Multivariate Spline Method for Scattered Data Fitting, Curve and Surface Reconstruction, and Numerical Solution to Poisson Equations via Domain Decomposition Method

by

Yidong Xu

(Under the Direction of Ming-Jun Lai)

Abstract

This work investigates three applications of bivariate and trivariate polynomial splines defined on triangulations and tetrahedralizations.

We first discuss a randomized block coordinate descent method, which can be further adapted to suit our spline applications. We split the domain into smaller ones randomly, solve the associated smaller problems while maintaining constraints, combine the solutions to get a global one, and repeat the whole process until certain stopping criterion is met.

The effectiveness of this method is then illustrated by three applications: scattered data fitting, curve and surface reconstruction, and spline solutions of Poisson equations. Each application has its own features. Numerical examples are presented.

Index words: multivariate splines, data fitting, curve and surface reconstruction, Poisson equations, domain decomposition method

Multivariate Spline Method for Scattered Data Fitting, Curve and Surface Reconstruction, and Numerical Solution to Poisson Equations via Domain Decomposition Method

by

Yidong Xu

B.S., Shanghai Jiao Tong University, 2011

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2019

Multivariate Spline Method for Scattered Data Fitting, Curve and Surface Reconstruction, and Numerical Solution to Poisson Equations via Domain Decomposition Method

by

Yidong Xu

Approved:

Major Professor: Ming-Jun Lai

Committee: Alexander Petukhov
Jingzhi Tie
Qing Zhang

Electronic Version Approved:

Ron Walcott
Interim Dean of the Graduate School
The University of Georgia
December 2019

Acknowledgments

I would like to thank my advisor Ming-Jun Lai without whose help and patience this work would not have been possible. I would also like to thank my parents for their understanding and support during my study.

Contents

Acknowledgments iv

List of Figures vii

List of Tables x

1 Introduction 1

2 Preliminary on Multivariate Splines 3

2.1 Bivariate Polynomials 3

2.2 Trivariate Polynomials 12

3 Triangulations and Tetrahedralizations 22

3.1 Introduction 22

3.2 Two-Dimensional Delaunay Triangulations 23

3.3 Three-Dimensional Delaunay Tetrahedralizations 35

4 Randomized Block Coordinate Descent Method 47

4.1 Introduction 47

4.2 Problem Formulation and Notations 48

4.3 Algorithm 48

4.4 Reduction of General Case 50

4.5 Convergence Analysis 53

4.6 Future Work 60

5 A Randomized Domain Decomposition Method for Computing Multivariate Spline Fits of Scattered Data 61

5.1 Introduction 61
5.2 Two-Dimensional Algorithm 63
5.3 Three-Dimensional Algorithm 74
5.4 Remarks and Future Work 80

6 Multivariate Splines for Curve and Surface Reconstruction 81
6.1 Introduction 81
6.2 Construction of 2D Smooth Curves 82
6.3 Construction of 3D Smooth Surfaces 102
6.4 Future Work 106

7 A Randomized Domain Decomposition Method for Spline Solutions of Poisson Equations 109
7.1 Introduction 109
7.2 Two-Dimensional Algorithm 110
7.3 Three-Dimensional Algorithm 118
7.4 Remarks and Future Work 126

Bibliography 127

List of Figures

3.1 A non-Delaunay triangulation. 26

3.2 A Delaunay triangulation. 26

3.3 A Delaunay triangulation of a convex polygon. 34

3.4 A refined triangulation of Figure 3.3. 34

3.5 A Delaunay triangulation of a non-convex polygonal domain with two holes. 35

3.6 A constrained Delaunay triangulation. 36

3.7 An example from [42] that doesn't have a constrained Delaunay tetrahedralization. 41

3.8 A Delaunay tetrahedralization of a cube. 43

3.9 A Delaunay tetrahedralization of a non-convex polyhedral domain. 44

3.10 A Delaunay tetrahedralization of a cube with two tunnels. 44

3.11 A Delaunay tetrahedralization of a torus-shape domain. 45

3.12 The left figure is a Delaunay tetrahedralization of a human head shape. The right one shows the inner structure. 45

5.1 $\text{star}^1(T)$. 65

5.2 $\text{star}^2(T)$. 65

5.3 Data points for Example 5.2.5. 69

5.4 Data points for Example 5.2.6. 70

5.5 Data points for Example 5.2.7. 71

5.6 Data points for Example 5.2.8. 72

5.7 Data points for Example 5.2.9. 73

5.8 Data points for Example 5.3.3. 78

5.9 Data points for Example 5.3.4. 79

5.10 Data points for Example 5.3.5. 80
6.1 One way to choose points for $\mathcal{A}_{-1}$, $\mathcal{A}_0$ and $\mathcal{A}_1$. 83
6.2 Another way to choose points for $\mathcal{A}_{-1}$, $\mathcal{A}_0$ and $\mathcal{A}_1$. 83
6.3 Illustration of computing contours. 86
6.4 Data points for Example 6.2.1. 87
6.5 The triangulation for Example 6.2.1. 88
6.6 The contour curve for Example 6.2.1. 88
6.7 Data points for Example 6.2.2. 89
6.8 The triangulation for Example 6.2.2. 90
6.9 The contour curve for Example 6.2.2. 90
6.10 Data points for Example 6.2.3. 91
6.11 The triangulation for Example 6.2.3. 91
6.12 The contour curve for Example 6.2.3. 92
6.13 A problem caused by C^{-1}. 93
6.14 The saddle point of $f(x, y) = x^2 - y^2$. 94
6.15 Generate a hole at the sharp corner. 95
6.16 A constrained triangulation with holes. 96
6.17 Data points for Example 6.2.4. 97
6.18 The triangulation for Example 6.2.4. 97
6.19 The contour curve for Example 6.2.4. 98
6.20 Data points for Example 6.2.5. 99
6.21 The triangulation for Example 6.2.5. 99
6.22 The contour curve for Example 6.2.5. 100
6.23 Data points for Example 6.2.6. 100
6.24 The triangulation for Example 6.2.6. 101
6.25 The contour curve for Example 6.2.6. 101
6.26 Data points and a tetrahedralization for Example 6.3.1. 104

6.27 The isosurface for Example 6.3.1. 105
6.28 Data points and a tetrahedralization for Example 6.3.2. 105
6.29 The isosurface for Example 6.3.2. 106
6.30 Data points and a tetrahedralization for Example 6.3.3. 107
6.31 The isosurface for Example 6.3.3. 107
6.32 The hollow interior of the isosurface in Fig. 6.31. 108
7.1 The triangulation for Example 7.2.3. 115
7.2 The triangulation for Example 7.2.4. 116
7.3 The triangulation for Example 7.2.5. 117
7.4 The tetrahedralization for Example 7.3.3. 122
7.5 The tetrahedralization for Example 7.3.4. 123
7.6 The tetrahedralization for Example 7.3.5. 124
7.7 The tetrahedralization for Example 7.3.6. 125

List of Tables

5.1 Approximation errors in Example 5.2.5 69

5.2 Approximation errors in Example 5.2.6 70

5.3 Approximation errors in Example 5.2.7 71

5.4 Approximation errors in Example 5.2.8 72

5.5 Approximation errors in Example 5.2.9 73

5.6 Approximation errors in Example 5.3.3 77

5.7 Approximation errors in Example 5.3.4 79

5.8 Approximation errors in Example 5.3.5 79

7.1 Approximation errors in Example 7.2.3 115

7.2 Approximation errors in Example 7.2.4 117

7.3 Approximation errors in Example 7.2.5 117

7.4 Approximation errors in Example 7.3.3 121

7.5 Approximation errors in Example 7.3.4 123

7.6 Approximation errors in Example 7.3.5 124

7.7 Approximation errors in Example 7.3.6 125

Chapter 1

Introduction

Multivariate spline functions are piecewise polynomial functions which have certain smoothness over given domains. They are useful in function approximation, surface design, and various other scenarios. Perhaps one of the most important applications is numerical solutions of partial differential equations (PDEs), where the success of the finite element method is unquestionable. There are many literatures devoted to these topics (cf. [48], [9], [8], and [7]). However, the implementation of the traditional finite element method with high-order smoothness is complicated and difficult.

The main focus of our work, polynomial splines on triangulations and tetrahedralizations, offers a different perspective and handles the smoothness elegantly. The key is to use the Bernstein representation of polynomials, which provides great flexibility and convenience. [27] gives a thorough analysis on this powerful tool. This approach may be summarized as follows. Each spline function s on its corresponding triangle/tetrahedron can be identified by its B-coefficient vector c. Since the spline function has certain smoothness, smoothness conditions need to be imposed and can be expressed by a linear system $Hc = 0$. In fact, as we shall see, many application problems can be reformulated into an optimization problem. Then the smoothness conditions serve as the constraints. The constrained optimization problem can be solved in many ways. For example, one can use the Lagrange multiplier method along with a matrix iterative method, as shown in [1]. There are many advantages of this approach. One of the most prominent characteristics is that we can use multivariate spline functions

of variable degrees and variable smoothness across any given polygonal/polyhedral domain with little change of the implementation. This provides a highly user-friendly platform.

The disadvantage is the necessity of solving linear systems of large size. In this work, we discuss a way to chop the original problem into smaller ones that are easy to handle. When the solutions to the subproblems are assembled together, it is a feasible solution to the original problem. Note that our method is an iterative method, and the decomposition is randomized.

The rest of the work is organized as follows. Chapter 2 gives some preliminaries on multivariate splines. Since the polynomial splines are based on triangulations and tetrahedralizations, Chapter 3 presents a brief introduction to this topic. Chapter 4 describes a randomized block coordinate descent method for linearly constrained convex optimization. This lays the foundation of our analysis. Chapter 5 presents a randomized domain decomposition method for solving large bivariate/trivariate scattered data fitting problems. Chapter 6 shows how to construct a smooth interpolatory or fitting curve of a given point set in the 2D setting, and a smooth surface in the 3D setting. It uses the algorithms from Chapter 5. Chapter 7 presents a randomized domain decomposition method for spline solutions of Poisson equations. In all three applications, numerical examples are given to illustrate the effectiveness.

All experiments were conducted on a Mac Pro with two 2.4 GHz quad-core Intel Xeon processors and 8 GB of memory.

Chapter 2

Preliminary on Multivariate Splines

In this chapter, we briefly review multivariate spline functions of any degree d and smoothness $0 \leq r < d$ over an arbitrary triangulation $\triangle$. Most of the discussion in this chapter can be found in [27].

2.1 Bivariate Polynomials

Given a nonnegative integer d, we write $\mathcal{P}_d$ for the space of bivariate polynomials of degree at most d. $\mathcal{P}_d$ is the linear space of all real-valued functions of the form

$$p(x,y) \;:=\; \sum_{0\leq i+j\leq d} c_{ij}x^i y^j, \tag{2.1}$$

where $\{c_{ij}\}_{0\leq i+j\leq d}$ are real numbers. It is easy to see that the monomials

$$\{x^i y^j\}_{0\leq i+j\leq d} \tag{2.2}$$

form a basis for $\mathcal{P}_d$. Thus, the dimension of $\mathcal{P}_d$ is $\binom{d+2}{2}$.

In this section we will construct a different basis for $\mathcal{P}_d$ and show that a bivariate polynomial can be written into a convenient form in terms of the barycentric coordinates associated with a triangle. This form enjoys many nice properties. Moreover, efficient and flexible computer programs can be designed to solve real-life problems.

2.1.1 Barycentric Coordinates

In this work, we are mostly working on polynomials on a triangulation. Barycentric coordinates are far more useful than Cartesian coordinates in this scenario. Recall that a non-degenerate triangle is one with nonzero area. Suppose T is a non-degenerate triangle in $\mathbb{R}^2$ with vertices

$$v_i := (x_i, y_i), \qquad i = 1, 2, 3. \tag{2.3}$$

Let's write $T := \langle v_1, v_2, v_3 \rangle$. We will assume that the vertices are numbered in counter-clockwise order. Barycentric coordinates give a way to represent the location of any point in $\mathbb{R}^2$ in terms of the three vertices of the triangle, as the following lemma shows (cf. [27]).

Lemma 2.1.1. *Let $T = \langle v_1, v_2, v_3 \rangle$ be a non-degenerate triangle. Then every point $v := (x, y) \in \mathbb{R}^2$ has a unique representation in the form*

$$v = b_1 v_1 + b_2 v_2 + b_3 v_3, \tag{2.4}$$

with

$$1 = b_1 + b_2 + b_3. \tag{2.5}$$

The numbers b_1, b_2, and b_3 are called the barycentric coordinates of the point v relative to T.

In matrix form, equations (2.4) and (2.5) become

$$\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}. \tag{2.6}$$

When T's vertices are ordered counter-clockwise, the area of T is given by

$$A_T = \frac{1}{2}\det(M), \tag{2.7}$$

where M is the 3×3 matrix in (2.6). Thus, M is nonsingular, and (2.6) then has a unique solution. By Crammer's Rule,

$$b_1 = \frac{1}{2A_T}\det\begin{bmatrix} 1 & 1 & 1 \\ x & x_2 & x_3 \\ y & y_2 & y_3 \end{bmatrix}, \tag{2.8}$$

$$b_2 = \frac{1}{2A_T}\det\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x & x_3 \\ y_1 & y & y_3 \end{bmatrix}, \tag{2.9}$$

$$b_3 = \frac{1}{2A_T}\det\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x \\ y_1 & y_2 & y \end{bmatrix}. \tag{2.10}$$

From (2.8), (2.9) and (2.10), it is clear that b_i's are functions of the Cartesian coordinates of $v = (x, y)$. More specifically, we have the following lemma (cf. [27]) which will be useful when we define the Bernstein basis polynomials in the next section.

Lemma 2.1.2. *For each $i = 1, 2, 3$, the function b_i is a linear polynomial in x and y which assumes value 1 at the vertex v_i and vanishes at all points on the edge of T opposite to v_i.*

Also note that a point v lies in the interior of T if and only if all three of its barycentric coordinates are positive.

2.1.2 Bernstein Basis Polynomials

Let $T = \langle v_1, v_2, v_3 \rangle$ be a fixed triangle, and d be a fixed positive integer. For each $v = (x, y) \in \mathbb{R}^2$, let b_1, b_2, b_3 be its barycentric coordinates relative to T. We are now ready to construct a new basis for $\mathcal{P}_d$. Given nonnegative integers i, j, k such that $i + j + k = d$, let

$$B^d_{ijk}(v) := \frac{d!}{i!j!k!} b_1^i b_2^j b_3^k. \tag{2.11}$$

By Lemma (2.1.2), each of b_i is a linear polynomial in x and y. It follows that B^d_{ijk} is a polynomial of degree d. These B^d_{ijk} are called the Bernstein basis polynomials of degree d relative to T.

Bernstein basis polynomials possess many important properties. One of them is that they form a partition of unity, i.e.,

$$\sum_{i+j+k=d} B^d_{ijk}(v) \equiv 1, \qquad \text{for all } v \in \mathbb{R}^2. \tag{2.12}$$

The other one worth mentioning is

$$0 \leq B^d_{ijk}(v) \leq 1, \qquad \text{for all } v \text{ in the triangle } T. \tag{2.13}$$

The next theorem shows that the set of Bernstein basis polynomials forms a basis for $\mathcal{P}_d$ (cf. [27]).

Theorem 2.1.3. *The set*

$$\mathcal{B}^d \;:=\; \{B^d_{ijk}\}_{i+j+k=d} \tag{2.14}$$

of Bernstein basis polynomials is a basis for the space $\mathcal{P}_d$ of polynomials of degree at most d.

2.1.3 B-form of Bivariate Polynomials

By Theorem 2.1.3, every bivariate polynomial p of degree at most d can be written uniquely in the form

$$p \;=\; \sum_{i+j+k=d} c_{ijk} B^d_{ijk}, \tag{2.15}$$

where B^d_{ijk} are the Bernstein basis polynomials associated with a fixed triangle T. The representation (2.15) is called the B-form of p relative to T. c_{ijk} are called the B-coefficients of p. By convention, these B-coefficients are ordered lexicographically and form a column vector c. For example, when $d = 3$,

$$c \;=\; [c_{300}, c_{210}, c_{201}, c_{120}, c_{111}, c_{102}, c_{030}, c_{021}, c_{012}, c_{003}]^\tau, \tag{2.16}$$

where the superscript τ means transpose.

Define the associated set of domain points to be

$$\mathcal{D}_{d,T} \;:=\; \{\xi_{ijk} := \frac{iv_1 + jv_2 + kv_3}{d}\}_{i+j+k=d}. \tag{2.17}$$

If the domain points are also ordered lexicographically, there is a one-to-one correspondence between each coefficient c_{ijk} and its associated domain point ξ_{ijk}.

One of the most important properties of the B-form representation is its stability, which means that the norm of p is comparable to the norm of its B-form coefficient vector.

Given a triangle T and any polynomial p of degree d, define

$$\|p\|_T \;=\; \sup_{v\in T} |p(v)|. \tag{2.18}$$

Write p in the B-form (2.15) with coefficient vector c. Measure c by

$$\|c\|_\infty \;=\; \max_{i+j+k=d} |c_{ijk}|. \tag{2.19}$$

Let $\{g_1, \ldots, g_n\}$ be the Bernstein basis polynomials of degree d, arranged in lexicographical order, and $\{t_1, \ldots, t_n\}$ be the associated domain points arranged in the same order, where $n := \binom{d+2}{2}$. Define the matrix

$$M \;:=\; [g_j(t_i)]_{i,j=1}^n. \tag{2.20}$$

Since M is nonsingular, let

$$K \;:=\; \|M^{-1}\|_\infty, \tag{2.21}$$

where $\|\cdot\|_\infty$ is the infinity matrix norm.

We are now ready to state the following theorem (cf. [27]).

Theorem 2.1.4. *Let p be a polynomial written in the B-form (2.15) with coefficient vector c. Then*

$$\frac{\|c\|_\infty}{K} \le \|p\|_T \le \|c\|_\infty, \tag{2.22}$$

where K is defined as (2.21), which depends only on d.

2.1.4 The de Casteljau Algorithm

The B-form representation of a polynomial p is very convenient for evaluation. The algorithm relies on the following theorem (cf. [27]).

Theorem 2.1.5. *Let p be a polynomial written in the B-form (2.15) with coefficients*

$$c_{ijk}^{(0)} := c_{ijk}, \qquad i+j+k=d. \tag{2.23}$$

Suppose v has barycentric coordinates $b := (b_1, b_2, b_3)$, and for all $l = 1, \ldots, d$, let

$$c_{ijk}^{(l)} \;:=\; b_1 c_{i+1,j,k}^{(l-1)} + b_2 c_{i,j+1,k}^{(l-1)} + b_3 c_{i,j,k+1}^{(l-1)}, \tag{2.24}$$

for $i+j+k = d-l$. Then

$$p(v) \;=\; \sum_{i+j+k=d-l} c_{ijk}^{(l)} B_{ijk}^{d-l}(v), \tag{2.25}$$

for all $0 \le l \le d$. In particular,

$$p(v) \;=\; c_{000}^{(d)}. \tag{2.26}$$

The de Casteljau algorithm now follows (cf. [27]).

Algorithm 2.1.6. *(de Casteljau)*

For $l = 1, \ldots, d$

For all $i+j+k = d-l$

$c_{ijk}^{(l)} := b_1 c_{i+1,j,k}^{(l-1)} + b_2 c_{i,j+1,k}^{(l-1)} + b_3 c_{i,j,k+1}^{(l-1)}$.

return $p(v) = c_{000}^{(d)}$.

2.1.5 Directional Derivatives

We first introduce the notation. Let f be a differentiable function on $\mathbb{R}^2$, and u be a vector in $\mathbb{R}^2$. Define the directional derivative of f at v with respect to u as

$$D_u f(v) \;:=\; \frac{d}{dt} f(v+tu)\Big|_{t=0}. \tag{2.27}$$

Since every point in $\mathbb{R}^2$ can be represented uniquely by its barycentric coordinates, the direction vector $u := v_1 - v_2$ can also be written uniquely by a triple (a_1, a_2, a_3) with

$$a_i := \alpha_i - \beta_i, \qquad i = 1, 2, 3, \tag{2.28}$$

where $(\alpha_1, \alpha_2, \alpha_3)$ and $(\beta_1, \beta_2, \beta_3)$ are the barycentric coordinates of the two points v_1 and v_2. The triple (a_1, a_2, a_3) is called the directional coordinates of u.

The next lemma gives the directional derivative of the Bernstein basis polynomials (cf. [27]).

Lemma 2.1.7. *Suppose u is a vector with directional coordinates $a := (a_1, a_2, a_3)$. Then for any $i + j + k = d$,*

$$D_u B^d_{ijk}(v) \;=\; d\left[a_1 B^{d-1}_{i-1,j,k}(v) + a_2 B^{d-1}_{i,j-1,k}(v) + a_3 B^{d-1}_{i,j,k-1}(v)\right], \tag{2.29}$$

with the convention that Bernstein basis polynomials with negative subscripts are taken to be identically zero.

In view of the B-form, the following theorem is straightforward (cf. [27]).

Theorem 2.1.8. *Let p be a polynomial of degree d written in the B-form (2.15) relative to a triangle T, and let u be a vector with directional coordinates $a := (a_1, a_2, a_3)$. Then the directional derivative at v of p in the direction u is given by*

$$D_u p(v) \;=\; d \sum_{i+j+k=d-1} c^{(1)}_{ijk}(a) B^{d-1}_{ijk}(v), \tag{2.30}$$

where $c^{(1)}_{ijk}(a)$ are the quantities obtained in the first step of the de Casteljau algorithm based on the triple a.

Combining Theorem 2.1.8 with Theorem 2.1.5, we can see that if v has barycentric coordinates $b = (b_1, b_2, b_3)$, to evaluate $D_u p(v)$, one can simply apply one step of the de Casteljau algorithm using a, followed by $d - 1$ steps using b.

Theorem 2.1.8 can be easily extended to higher-order directional derivatives (cf. [27]).

Theorem 2.1.9. *Let $1 \le m \le d$, and $u_1, \ldots, u_m$ be m directions described by the triples*

$$a^{(i)} \;:=\; (a_1^{(i)}, a_2^{(i)}, a_3^{(i)}) \tag{2.31}$$

for $i = 1, \ldots, m$. Then

$$D_{u_m} \cdots D_{u_1} p(v) \;=\; \frac{d!}{(d-m)!} \sum_{i+j+k=d-m} c_{ijk}^{(m)}(a^{(1)}, \ldots, a^{(m)}) B_{ijk}^{d-m}(v), \tag{2.32}$$

where $c_{ijk}^{(m)}(a^{(1)}, \ldots, a^{(m)})$ are the quantities obtained after carrying out m steps of the de Casteljau algorithm using $a^{(1)}, \ldots, a^{(m)}$ in order.

Corollary 2.1.10. *Let u be a direction described by the triple a. Then for $1 \le m \le d$,*

$$D_u^m p(v) := \underbrace{D_u \cdots D_u}_{m} p(v) \;=\; \frac{d!}{(d-m)!} \sum_{i+j+k=d-m} c_{ijk}^{(m)}(a) B_{ijk}^{d-m}(v), \tag{2.33}$$

where $c_{ijk}^{(m)}(a)$ are the quantities obtained after carrying out m steps of the de Casteljau algorithm using a.

2.1.6 Conditions for Smooth Joins of Polynomials

In our study, we deal with a triangulation $\triangle$ of a polygonal domain; see Chapter 3 for more about triangulations. A smooth join between two polynomials on adjoining triangles is essential for finding a satisfactory surface. The following theorem provides such a useful tool (cf. [27]).

Theorem 2.1.11. *Let $T := \langle v_1, v_2, v_3 \rangle$ and $\widetilde{T} := \langle v_4, v_3, v_2 \rangle$ be triangles sharing the edge $e := \langle v_2, v_3 \rangle$. Let*

$$p(v) \;:=\; \sum_{i+j+k=d} c_{ijk} B_{ijk}^d(v) \tag{2.34}$$

and

$$\widetilde{p}(v) \;:=\; \sum_{i+j+k=d} \widetilde{c}_{ijk} \widetilde{B}_{ijk}^d(v), \tag{2.35}$$

where $\{B^d_{ijk}\}$ *and* $\{\widetilde{B}^d_{ijk}\}$ *are the Bernstein basis polynomials associated with* T *and* $\widetilde{T}$*, respectively. Suppose* u *is any direction not parallel to* e*. Then*

$$D^n_u p(v) = D^n_u \widetilde{p}(v), \qquad \text{for all } v \in e \text{ and } n = 0, \ldots, r, \tag{2.36}$$

if and only if

$$\widetilde{c}_{njk} = \sum_{\nu+\mu+\kappa=n} c_{\nu,k+\mu,j+\kappa} B^n_{\nu\mu\kappa}(v_4), \qquad \text{for } j+k = d-n \text{ and } n = 0, \ldots, r. \tag{2.37}$$

2.1.7 Integrals and Inner Products of B-Polynomials

There exist explicit formulas for integrals and inner products of polynomials represented in B-form (cf. [27]).

Theorem 2.1.12. *Given a triangle* T*,*

$$\int_T B^d_{ijk}(x,y)dxdy \;=\; \frac{A_T}{\binom{d+2}{2}}, \tag{2.38}$$

for all $i+j+k=d$*, where* A_T *is the area of* T*.*

Theorem 2.1.13. *Given a triangle* T*, let* p *be a polynomial of degree* d *written in the B-form (2.15). Then*

$$\int_T p(x,y)dxdy \;=\; \frac{A_T}{\binom{d+2}{2}} \sum_{i+j+k=d} c_{ijk}, \tag{2.39}$$

where A_T *is the area of* T*.*

Theorem 2.1.14. *Given a triangle* T*,*

$$\int_T B^d_{ijk}(x,y) B^d_{\nu\mu\kappa}(x,y)dxdy \;=\; \frac{\binom{i+\nu}{i}\binom{j+\mu}{j}\binom{k+\kappa}{k} A_T}{\binom{2d}{d}\binom{2d+2}{2}}, \tag{2.40}$$

where A_T *is the area of* T*.*

Theorem 2.1.15. *Given a triangle* T*, let* p *and* $\widetilde{p}$ *be two polynomials of degree* d *written in the B-form (2.15) with coefficient vectors* c *and* $\widetilde{c}$ *respectively. Then*

$$\int_T p(x,y)\widetilde{p}(x,y)dxdy \;=\; \frac{A_T}{\binom{2d}{d}\binom{2d+2}{2}} \sum_{\substack{i+j+k=d \\ \nu+\mu+\kappa=d}} \binom{i+\nu}{i}\binom{j+\mu}{j}\binom{k+\kappa}{k} c_{ijk}\widetilde{c}_{\nu\mu\kappa}, \tag{2.41}$$

where A_T is the area of T. In fact, this inner product can be written in the matrix form

$$\int_T p(x,y)\widetilde{p}(x,y)dxdy \;=\; \frac{A_T}{\binom{2d}{d}\binom{2d+2}{2}} c^\tau G \widetilde{c}, \tag{2.42}$$

where G is a $\binom{d+2}{2}$ square matrix.

2.2 Trivariate Polynomials

Bivariate polynomials can be extended to trivariate ones. Although we will be using the same notation as in the previous section of bivariate polynomials, it shall cause no confusion in the context.

Given a nonnegative integer d, we write $\mathcal{P}_d$ for the space of trivariate polynomials of degree at most d. $\mathcal{P}_d$ is the linear space of all real-valued functions of the form

$$p(x,y,z) \;:=\; \sum_{0\le i+j+k\le d} c_{ijk} x^i y^j z^k, \tag{2.43}$$

where $\{c_{ijk}\}_{0\le i+j+k\le d}$ are real numbers. It is easy to see that the monomials

$$\{x^i y^j z^k\}_{0\le i+j+k\le d} \tag{2.44}$$

form a basis for $\mathcal{P}_d$. Thus, the dimension of $\mathcal{P}_d$ is $\binom{d+3}{3}$.

Just like the bivariate case, we can construct a different basis for $\mathcal{P}_d$ and a trivariate polynomial can be written into a convenient form in terms of the barycentric coordinates associated with a tetrahedron. Similar nice properties can be derived and efficient computer programs can be designed accordingly.

2.2.1 Barycentric Coordinates

In the rest of this section we are mostly working on polynomials on a tetrahedron. We begin with the definitions of non-degeneracy and the canonical order in a tetrahedron (cf. [27]).

Definition 2.2.1. *Suppose T is a non-degenerate tetrahedron in $\mathbb{R}^3$ with vertices*

$$v_i := (x_i, y_i, z_i), \qquad i = 1,2,3,4. \tag{2.45}$$

Write $T := \langle v_1, v_2, v_3, v_4 \rangle$. We say that T is non-degenerate provided that it has nonzero volume. We say that the vertices of T are in canonical order provided that if we rotate and translate T so that the triangular face $\langle v_1, v_2, v_3 \rangle$ lies in the x-y-plane with v_1, v_2, v_3 in counterclockwise order, then $z_4 > 0$.

On a tetrahedron, barycentric coordinates are far more useful than Cartesian coordinates. Barycentric coordinates present a way to represent the location of any point in $\mathbb{R}^3$ in terms of the four vertices of the tetrahedron, as the following lemma shows (cf. [27]).

Lemma 2.2.2. *Let $T = \langle v_1, v_2, v_3, v_4 \rangle$ be a non-degenerate tetrahedron. Then every point $v := (x, y, z) \in \mathbb{R}^3$ has a unique representation in the form*

$$v = b_1 v_1 + b_2 v_2 + b_3 v_3 + b_4 v_4, \tag{2.46}$$

with

$$1 = b_1 + b_2 + b_3 + b_4. \tag{2.47}$$

The numbers b_1, b_2, b_3 and b_4 are called the barycentric coordinates of the point v relative to T.

In matrix form, equations (2.46) and (2.47) become

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix}. \tag{2.48}$$

When T's vertices are in canonical order, the volume of T is given by

$$V_T = \frac{1}{6}\det(M), \tag{2.49}$$

where M is the 4×4 matrix in (2.48). Thus, M is nonsingular, and (2.48) then has a unique solution. By Crammer's Rule,

$$b_1 = \frac{1}{6V_T}\det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x & x_2 & x_3 & x_4 \\ y & y_2 & y_3 & y_4 \\ z & z_2 & z_3 & z_4 \end{bmatrix}, \tag{2.50}$$

$$b_2 = \frac{1}{6V_T}\det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x & x_3 & x_4 \\ y_1 & y & y_3 & y_4 \\ z_1 & z & z_3 & z_4 \end{bmatrix}, \tag{2.51}$$

$$b_3 = \frac{1}{6V_T}\det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x & x_4 \\ y_1 & y_2 & y & y_4 \\ z_1 & z_2 & z & z_4 \end{bmatrix}. \tag{2.52}$$

$$b_4 = \frac{1}{6V_T}\det\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x \\ y_1 & y_2 & y_3 & y \\ z_1 & z_2 & z_3 & z \end{bmatrix}. \tag{2.53}$$

From (2.50) to (2.53), it is clear that b_i's are functions of the Cartesian coordinates of $v = (x, y, z)$. More specifically, we have the following lemma (cf. [27]) which will be useful when we define the Bernstein basis polynomials in the next section.

Lemma 2.2.3. *For each $i = 1, 2, 3, 4$, the function b_i is a linear polynomial in x, y, and z which assumes value 1 at the vertex v_i and vanishes at all points on the face of T opposite to v_i.*

Also note that a point v lies in the interior of T if and only if all four of its barycentric coordinates are positive.

2.2.2 Bernstein Basis Polynomials

Let $T = \langle v_1, v_2, v_3, v_4 \rangle$ be a fixed tetrahedron, and d be a fixed positive integer. For each $v = (x, y, z) \in \mathbb{R}^3$, let b_1, b_2, b_3, b_4 be its barycentric coordinates relative to T. We can now construct a new basis for $\mathcal{P}_d$. Given nonnegative integers i, j, k, l such that $i+j+k+l = d$, let

$$B^d_{ijkl}(v) \;:=\; \frac{d!}{i!j!k!l!} b_1^i b_2^j b_3^k b_4^l. \tag{2.54}$$

By Lemma (2.2.3), each of b_i is a linear polynomial in x, y and z. It follows that B^d_{ijkl} is a polynomial of degree d. These B^d_{ijkl} are called the Bernstein basis polynomials of degree d relative to T.

These trivariate Bernstein basis polynomials have many important properties which are similar to those of the bivariate correspondents. To name a few,

$$\sum_{i+j+k+l=d} B^d_{ijkl}(v) \equiv 1, \qquad \text{for all } v \in \mathbb{R}^3, \tag{2.55}$$

and

$$0 \le B^d_{ijkl}(v) \le 1, \qquad \text{for all } v \text{ in the tetrahedron } T. \tag{2.56}$$

The next theorem shows that the set of Bernstein basis polynomials forms a basis for $\mathcal{P}_d$ (cf. [27]).

Theorem 2.2.4. *The set*

$$\mathcal{B}^d \;:=\; \{B^d_{ijkl}\}_{i+j+k+l=d} \tag{2.57}$$

of Bernstein basis polynomials is a basis for the space $\mathcal{P}_d$ of polynomials of degree at most d.

2.2.3 B-form of Trivariate Polynomials

By Theorem 2.2.4, every trivariate polynomial p of degree at most d can be written uniquely in the form

$$p \;=\; \sum_{i+j+k+l=d} c_{ijkl} B^d_{ijkl}, \tag{2.58}$$

where B^d_{ijkl} are the Bernstein basis polynomials associated with a fixed tetrahedron T. The representation (2.58) is called the B-form of p relative to T. c_{ijkl} are called the B-coefficients of p. As in the bivariate case, these B-coefficients are ordered lexicographically and form a column vector c. For example, when $d = 2$,

$$c \;=\; [c_{2000}, c_{1100}, c_{1010}, c_{1001}, c_{0200}, c_{0110}, c_{0101}, c_{0020}, c_{0011}, c_{0002}]^\tau, \tag{2.59}$$

where the superscript τ means transpose.

Define the associated set of domain points to be

$$\mathcal{D}_{d,T} \;:=\; \{\xi_{ijkl} := \frac{iv_1 + jv_2 + kv_3 + lv_4}{d}\}_{i+j+k+l=d}. \tag{2.60}$$

If the domain points are also ordered lexicographically, there is a one-to-one correspondence between each coefficient c_{ijkl} and its associated domain point ξ_{ijkl}.

One of the most important properties of the B-form representation is its stability, which means that the norm of p is comparable to the norm of its B-form coefficient vector.

Given a tetrahedron T and any polynomial p of degree d, define

$$\|p\|_T \;=\; \sup_{v\in T} |p(v)|. \tag{2.61}$$

Write p in the B-form (2.58) with coefficient vector c. Measure c by

$$\|c\|_\infty \;=\; \max_{i+j+k+l=d} |c_{ijkl}|. \tag{2.62}$$

Let $\{g_1, \ldots, g_n\}$ be the Bernstein basis polynomials of degree d, arranged in lexicographical order, and $\{t_1, \ldots, t_n\}$ be the associated domain points arranged in the same order, where $n := \binom{d+3}{3}$. Define the matrix

$$M \;:=\; [g_j(t_i)]^n_{i,j=1}. \tag{2.63}$$

Since M is nonsingular, let

$$K \;:=\; \|M^{-1}\|_\infty, \tag{2.64}$$

where $\|\cdot\|_\infty$ is the infinity matrix norm.

We are now ready to state the following theorem (cf. [27]).

Theorem 2.2.5. *Let p be a trivariate polynomial written in the B-form (2.58) with coefficient vector c. Then*

$$\frac{\|c\|_\infty}{K} \leq \|p\|_T \leq \|c\|_\infty, \tag{2.65}$$

where K is defined as (2.64), which depends only on d.

2.2.4 The de Casteljau Algorithm

The bivariate de Casteljau Algorithm can be easily extended to the trivariate case to evaluate a polynomial in its B-form representation. The algorithm relies on the following theorem (cf. [27]).

Theorem 2.2.6. *Let p be a trivariate polynomial written in the B-form (2.58) with coefficients*

$$c_{ijkl}^{(0)} := c_{ijkl}, \qquad i+j+k+l = d. \tag{2.66}$$

Suppose v has barycentric coordinates $b := (b_1, b_2, b_3, b_4)$, and for all $m = 1, \ldots, d$, let

$$c_{ijkl}^{(m)} \;:=\; b_1 c_{i+1,j,k,l}^{(m-1)} + b_2 c_{i,j+1,k,l}^{(m-1)} + b_3 c_{i,j,k+1,l}^{(m-1)} + b_4 c_{i,j,k,l+1}^{(m-1)}, \tag{2.67}$$

for $i+j+k+l = d-m$. Then

$$p(v) \;=\; \sum_{i+j+k+l=d-m} c_{ijkl}^{(m)} B_{ijkl}^{d-m}(v), \tag{2.68}$$

for all $0 \leq m \leq d$. In particular,

$$p(v) \;=\; c_{0000}^{(d)}. \tag{2.69}$$

The de Casteljau algorithm now follows (cf. [27]).

Algorithm 2.2.7. *(de Casteljau)*

For $m = 1, \ldots, d$

For all $i+j+k+l = d-m$

$$c_{ijkl}^{(m)} := b_1 c_{i+1,j,k,l}^{(m-1)} + b_2 c_{i,j+1,k,l}^{(m-1)} + b_3 c_{i,j,k+1,l}^{(m-1)} + b_4 c_{i,j,k,l+1}^{(m-1)}.$$

return $p(v) = c_{0000}^{(d)}$.

2.2.5 Directional Derivatives

Let f be a differentiable function on $\mathbb{R}^3$, and u be a vector in $\mathbb{R}^3$. Define the directional derivative of f at v with respect to u as

$$D_u f(v) \;:=\; \frac{d}{dt} f(v+tu)\Big|_{t=0}. \tag{2.70}$$

Since every point in $\mathbb{R}^3$ can be represented uniquely by its barycentric coordinates, the direction vector $u := v_1 - v_2$ can also be written uniquely by a quadruple (a_1, a_2, a_3, a_4) with

$$a_i := \alpha_i - \beta_i, \qquad i = 1, 2, 3, 4, \tag{2.71}$$

where $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and $(\beta_1, \beta_2, \beta_3, \beta_4)$ are the barycentric coordinates of the two points v_1 and v_2. The quadruple (a_1, a_2, a_3, a_4) is called the directional coordinates of u.

The next lemma gives the directional derivative of the trivariate Bernstein basis polynomials which is similar to the bivariate case (cf. [27]).

Lemma 2.2.8. *Suppose u is a vector with directional coordinates $a := (a_1, a_2, a_3, a_4)$. Then for any $i + j + k + l = d$,*

$$\begin{aligned} D_u B^d_{ijkl}(v) \;&=\; d\left[a_1 B^{d-1}_{i-1,j,k,l}(v) + a_2 B^{d-1}_{i,j-1,k,l}(v)\right. \\ &\qquad \left. + a_3 B^{d-1}_{i,j,k-1,l}(v) + a_4 B^{d-1}_{i,j,k,l-1}(v)\right], \end{aligned} \tag{2.72}$$

with the convention that Bernstein basis polynomials with negative subscripts are taken to be identically zero.

In view of the B-form, the following theorem is straightforward (cf. [27]).

Theorem 2.2.9. *Let p be a trivariate polynomial of degree d written in the B-form (2.58) relative to a tetrahedron T, and let u be a vector with directional coordinates $a := (a_1, a_2, a_3, a_4)$. Then the directional derivative at v of p in the direction u is given by*

$$D_u p(v) \;=\; d \sum_{i+j+k+l=d-1} c^{(1)}_{ijkl}(a) B^{d-1}_{ijkl}(v), \tag{2.73}$$

where $c^{(1)}_{ijkl}(a)$ are the quantities obtained in the first step of the de Casteljau algorithm based on the quadruple a.

Combining Theorem 2.2.9 with Theorem 2.2.6, we can see that if v has barycentric coordinates $b = (b_1, b_2, b_3, b_4)$, to evaluate $D_u p(v)$, one can simply apply one step of the de Casteljau algorithm using a, followed by $d-1$ steps using b.

Theorem 2.2.9 can be easily extended to higher-order directional derivatives (cf. [27]).

Theorem 2.2.10. *Let $1 \le m \le d$, and $u_1, \ldots, u_m$ be m directions described by the quadruples*

$$a^{(i)} \;:=\; (a_1^{(i)}, a_2^{(i)}, a_3^{(i)}, a_4^{(i)}) \tag{2.74}$$

for $i = 1, \ldots, m$. Then

$$D_{u_m} \cdots D_{u_1} p(v) \;=\; \frac{d!}{(d-m)!} \sum_{i+j+k+l=d-m} c_{ijkl}^{(m)}(a^{(1)}, \ldots, a^{(m)}) B_{ijkl}^{d-m}(v), \tag{2.75}$$

where $c_{ijkl}^{(m)}(a^{(1)}, \ldots, a^{(m)})$ are the quantities obtained after carrying out m steps of the de Casteljau algorithm using $a^{(1)}, \ldots, a^{(m)}$ in order.

Corollary 2.2.11. *Let u be a direction described by the quadruple a. Then for $1 \le m \le d$,*

$$D_u^m p(v) := \underbrace{D_u \cdots D_u}_{m} p(v) \;=\; \frac{d!}{(d-m)!} \sum_{i+j+k+l=d-m} c_{ijkl}^{(m)}(a) B_{ijkl}^{d-m}(v), \tag{2.76}$$

where $c_{ijkl}^{(m)}(a)$ are the quantities obtained after carrying out m steps of the de Casteljau algorithm using a.

2.2.6 Conditions for Smooth Joins of Trivariate Polynomials

In our study, we will deal with a tetrahedralization $\triangle$ of a polyhedral domain; see Chapter 3 for more about tetrahedralization. A smooth join between two trivariate polynomials on adjoining tetrahedrons is essential in many applications. The following theorem provides such a useful tool (cf. [27]).

Theorem 2.2.12. *Let $T := \langle v_1, v_2, v_3, v_4 \rangle$ and $\widetilde{T} := \langle v_5, v_2, v_4, v_3 \rangle$ be tetrahedrons sharing the face $F := \langle v_2, v_3, v_4 \rangle$. Let*

$$p(v) \;:=\; \sum_{i+j+k+l=d} c_{ijkl} B_{ijkl}^{d}(v) \tag{2.77}$$

and

$$\widetilde{p}(v) \;:=\; \sum_{i+j+k+l=d} \widetilde{c}_{ijkl}\widetilde{B}^d_{ijkl}(v), \tag{2.78}$$

where $\{B^d_{ijkl}\}$ *and* $\{\widetilde{B}^d_{ijkl}\}$ *are the trivariate Bernstein basis polynomials associated with* T *and* $\widetilde{T}$*, respectively. Then* p *and* $\widetilde{p}$ *join together with* C^r *continuity across the face* F *if and only if*

$$\widetilde{c}_{mijk} = \sum_{\nu+\mu+\kappa+\delta=m} c_{\nu,i+\mu,k+\kappa,j+\delta}B^m_{\nu\mu\kappa\delta}(v_5), \qquad \text{for } i+j+k = d-m \quad \text{and} \quad m = 0,\ldots,r. \tag{2.79}$$

2.2.7 Integrals and Inner Products of Trivariate B-Polynomials

There exist explicit formulas for integrals and inner products of trivariate polynomials represented in B-form (cf. [27]).

Theorem 2.2.13. *Given a tetrahedron* T*,*

$$\int_T B^d_{ijkl}(x,y,z)dxdydz \;=\; \frac{V_T}{\binom{d+3}{3}}, \tag{2.80}$$

for all $i+j+k+l=d$*, where* V_T *is the volume of* T*.*

Theorem 2.2.14. *Given a tetrahedron* T*, let* p *be a trivariate polynomial of degree* d *written in the B-form (2.58). Then*

$$\int_T p(x,y,z)dxdydz \;=\; \frac{V_T}{\binom{d+3}{3}} \sum_{i+j+k+l=d} c_{ijkl}, \tag{2.81}$$

where V_T *is the volume of* T*.*

Theorem 2.2.15. *Given a tetrahedron* T*,*

$$\int_T B^d_{ijkl}(x,y,z)B^d_{\nu\mu\kappa\delta}(x,y,z)dxdydz \;=\; \frac{\binom{i+\nu}{i}\binom{j+\mu}{j}\binom{k+\kappa}{k}\binom{l+\delta}{l}V_T}{\binom{2d}{d}\binom{2d+3}{3}}, \tag{2.82}$$

where V_T *is the volume of* T*.*

Theorem 2.2.16. *Given a tetrahedron T, let p and $\widetilde{p}$ be two trivariate polynomials of degree d written in the B-form (2.58) with coefficient vectors c and $\widetilde{c}$ respectively. Then*

$$\int_T p(x,y,z)\widetilde{p}(x,y,z)dxdydz \;=\; \frac{V_T}{\binom{2d}{d}\binom{2d+3}{3}} \sum_{\substack{i+j+k+l=d \\ \nu+\mu+\kappa+\delta=d}} \binom{i+\nu}{i}\binom{j+\mu}{j}\binom{k+\kappa}{k}\binom{l+\delta}{l} c_{ijkl}\widetilde{c}_{\nu\mu\kappa\delta}, \tag{2.83}$$

where V_T is the volume of T. In fact, this inner product can be written in the matrix form

$$\int_T p(x,y,z)\widetilde{p}(x,y,z)dxdydz \;=\; \frac{V_T}{\binom{2d}{d}\binom{2d+3}{3}} c^\intercal G\widetilde{c}, \tag{2.84}$$

where G is a $\binom{d+3}{3}$ square matrix.

Chapter 3

Triangulations and Tetrahedralizations

In order to use multivariate spline functions discussed in Chapter 2, a nice partition of the domain is crucial. In this chapter, we first introduce some basic concepts. Then we propose our algorithms for both two-dimensional and three-dimensional scenarios.

3.1 Introduction

In industry, triangulation is sometimes referred to as mesh generation. The automatic mesh generation problem is to divide a physical domain of a probably complicated geometry into small simple pieces, such as triangles or rectangles for two-dimensional geometries, and tetrahedrons or rectangular prisms for three-dimensional geometries. These small pieces are called elements of the mesh. Mesh generation has been one of the cutting-edge research fields for decades, and still remains active.

A nice mesh must satisfy some basic requirements. Here we list three of them. Of course, additional requirements may be imposed according to the practical applications.

First of all, it must conform to the shape of the domain of interest. For example, in the finite element method of solving PDE, the boundary condition makes essential the match of the triangulation and the domain (cf. [23]). When we compute eigenvalues and eigenfunctions associated with Laplace operator over a polygonal domain, a precise triangulation of the domain is necessary to guarantee a good approximation.

Secondly, the elements in a mesh shall be neither too large nor too small. Many applications use triangulations to interpolate a multivariate function whose true value might be unknown. Usually, we know the values of the function at points in a large sample, and want to approximate the values of the function at points not in the sample. One way to reduce the interpolation error is to use small elements. However, a small size of the elements results in a large number of them, hence escalating the scale of computational complexity. So picking a proper size is important from both theoretic and practical views. Sometimes an adaptive method can be employed (cf. [26]).

Last but not least, most of the elements shall have the right shape that is neither too long nor too thin. Large angles (near 180°) and small angles (near 0°) would accompany such bad shapes. Large angles can cause large interpolation errors which, in the finite element method, induce a large discretization error, the difference between the computed approximation and the true solution of PDE. On the other hand, small angles can cause the stiffness matrices associated with the finite element method to be ill-conditioned.

Since meshes find heavy use in hundreds of real-life applications, such as land surveying, image processing, and mechanical designs, there are numerous literatures devoted to this topic. The lecture notes [42] provide an elementary and thorough overview. Other excellent sources include [35] and [32]. Further surveys are supplied by [39], [4], [43], [6] and [16].

3.2 Two-Dimensional Delaunay Triangulations

In this section, we discuss two-dimensional Delaunay triangulations, constrained Delaunay triangulations, and their geometric properties.

3.2.1 Triangulations

We start with a very general definition of triangulations (cf. [27]).

Definition 3.2.1. *A collection $\triangle := \{T_1, \ldots, T_N\}$ of triangles in the plane is called a triangulation of $\Omega = \cup_{i=1}^{N} T_i$ provided that if a pair of triangles in $\triangle$ intersect, then their intersection is either a common vertex or a common edge.*

This definition is general enough to include most of the triangulations encountered in applications. Perhaps most importantly, it allows triangulations of a domain Ω with one or more holes. We will see some examples later in this section.

Definition 3.2.2. *The vertices of the triangles of $\triangle$ are called the vertices of the triangulation $\triangle$. If a vertex v is a boundary point of Ω, we say that it is a boundary vertex. Otherwise, it is called an interior vertex. Similarly, the edges of the triangles of $\triangle$ are called the edges of the triangulation $\triangle$. If an edge e lies on the boundary of Ω, we say that it is a boundary edge. Otherwise, it is called an interior edge.*

There are many methods to generate a triangulation. Below is a simple algorithm for constructing a triangulation $\triangle$ of the convex hull of a set $\mathcal{V}$ of vertices (cf. [27]). We call the resulting $\triangle$ a triangulation of $\mathcal{V}$.

If $\triangle$ is a triangulation and v is a point, we say that a vertex w of $\triangle$ is visible to v provided that it is possible to draw a line from v to w which does not cross any of the edges of $\triangle$.

Algorithm 3.2.3. *(Vertex Insertion Algorithm)*

Let $\mathcal{V} = \{v_i\}_{i=1}^{n}$ be a given set of points in $\mathbb{R}^2$.

1. *Connect v_{n-2}, v_{n-1}, v_n to form an initial triangulation $\triangle^{(0)}$ consisting of one triangle.*
2. ***for** $i = 1$ to $n - 3$ **do***
   (a) *If v_i is strictly inside some triangle T of $\triangle^{(i-1)}$, connect v_i to the three vertices of T. If v_i is on an edge e of a triangle T of $\triangle^{(i-1)}$ connect it to the opposite vertex of all triangles sharing the edge e.*
   (b) *Otherwise, connect v_i to all of the vertices of $\triangle^{(i-1)}$ which are visible to v_i.*

(c) Define $\triangle^{(i)}$ to be the new triangulation.

Algorithm 3.2.3 is simple and straightforward, but not very useful in practice, since it may produce unsatisfactory triangulations of bad shapes and it fails to conform to a general polygonal domain other than convex ones. In the next subsection, a much better triangulation with optimal properties will be discussed.

3.2.2 Delaunay Triangulations

As mentioned at the beginning of this chapter, a nice shape of the triangles in the triangulation is preferred. The Delaunay triangulation introduced in Year 1934 is one with most of the nice properties. To define it geometrically, we need to use circumcircles.

Definition 3.2.4. *The circumcircle of a triangle is the unique circle that passed through all three of its vertices. A circumcircle of an edge is any circle that passes through both its vertices.*

Definition 3.2.5. *(Delaunay) Suppose Ω is a convex polygonal domain and $\triangle$ is a triangulation of Ω. A triangle $T \in \triangle$ is Delaunay if there are no vertices of $\triangle$ inside the circumcircle of T. An edge e of $\triangle$ is Delaunay if it has at least one empty circumcircle. The triangulation $\triangle$ is Delaunay if every triangle in it is Delaunay.*

Figure 3.1 shows an example of a non-Delaunay triangulation, while Figure 3.2 is a Delaunay triangulation of the same four vertices.

There is a useful alternative characterization of the Delaunay triangulation as shown in the following definition (cf. [42]).

Definition 3.2.6. *(Locally Delaunay) Let e be an edge of a triangulation $\triangle$. If e is the edge of only one triangle, then e is said to be locally Delaunay. If e is the edge of two triangles T_1 and T_2, then e is locally Delaunay if it has a circumcircle enclosing no vertex of T_1 nor T_2. Equivalently, the circumcircle of T_1 encloses no vertex of T_2. Equivalently, the circumcircle of T_2 encloses no vertex of T_1.*

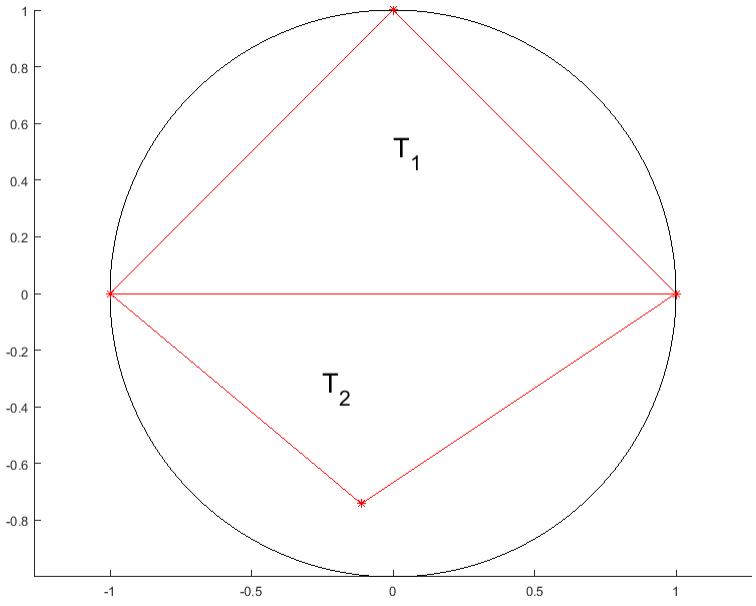

Figure 3.1: A non-Delaunay triangulation.

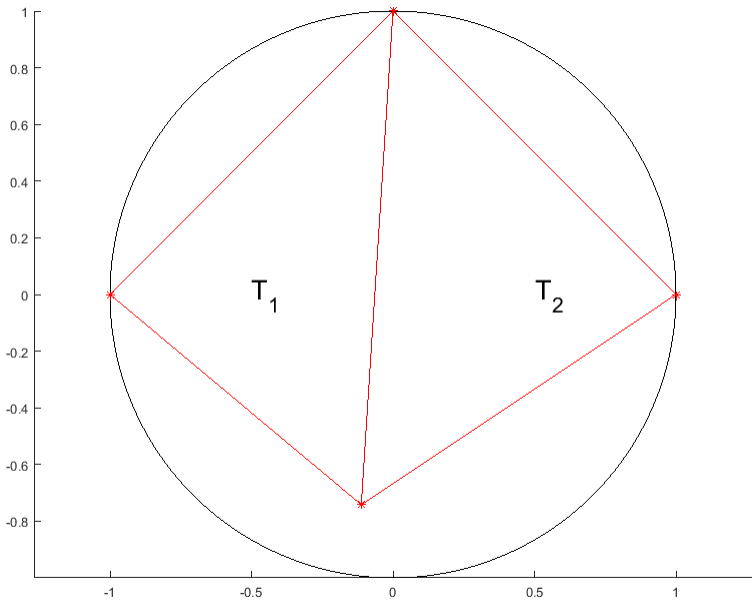

Figure 3.2: A Delaunay triangulation.

One of the most important results about Delaunay triangulation is the Delaunay Lemma, as shown below (cf. [42]).

Theorem 3.2.7. *(The Delaunay Lemma) Suppose Ω is a convex polygonal domain and $\triangle$ is a triangulation of Ω. The following three statements are equivalent.*

1. *Every triangle $T \in \triangle$ is Delaunay (i.e. $\triangle$ is Delaunay).*
2. *Every edge e of $\triangle$ is Delaunay.*
3. *Every edge e of $\triangle$ is locally Delaunay.*

The Delaunay Lemma tells us that in order to get a Delaunay triangulation, it suffices to make every edge locally Delaunay. If an edge e is not locally Delaunay, it must be a shared edge of two triangles. The union of the two triangles forms a quadrilateral, and e is a diagonal. The following lemma shows that we can flip the edge and the new edge is locally Delaunay (cf. [42]). Of course, an edge flip is legal only if the two diagonals cross each other, or equivalently, if the quadrilateral is convex. Note that unflippable edges are always locally Delaunay.

Lemma 3.2.8. *Let e be an edge of $\triangle$. Then either e is locally Delaunay, or e is flippable and the edge created by flipping e is locally Delaunay.*

Lemma 3.2.8 provides us a way to produce a Delaunay triangulation (cf. [27]).

Algorithm 3.2.9. *(Flip Algorithm)*

1. *Construct an initial triangulation $\triangle^{(0)}$ using any method, for example, Algorithm 3.2.3. Set $k := 0$.*
2. *Do until no longer possible:*
   Find an edge e of $\triangle^{(k)}$ which is not locally Delaunay, and flip it. Increase k by one, and let $\triangle^{(k)}$ be the new triangulation.

A natural question to ask is whether Algorithm 3.2.9 can terminate in a finite number of steps. The following theorem from [42] claims that the algorithm won't go on forever.

Theorem 3.2.10. *Given a triangulation $\triangle$ with n vertices, Algorithm 3.2.9 terminates after $O(n^2)$ edge flips, yielding a Delaunay triangulation.*

Given a set $\mathcal{V}$ of vertices, we can apply Algorithm 3.2.3 to generate a triangulation of the convex hull of $\mathcal{V}$. Then we can use Algorithm 3.2.9 to convert it to Delaunay. This implies the following corollary (cf. [42]).

Corollary 3.2.11. *Every finite set of points in $\mathbb{R}^2$ has a Delaunay triangulation.*

Although Delaunay triangulations may not be unique for a given set of vertices, they do optimize several geometric criteria, which is the reason why Delaunay triangulations are useful in practice. The following theorem illustrates those nice properties (cf. [42]), where the minimum angle in the triangulation $\triangle$ is the smallest angle among all the triangles in $\triangle$, and the min-containment circle of a triangle is the smallest circle that encloses it.

Theorem 3.2.12. *Among all the triangulations of a point set $\mathcal{V}$, every Delaunay triangulation of $\mathcal{V}$ maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle.*

3.2.3 Constrained Delaunay Triangulations

The Delaunay triangulations we discussed in the last subsection might not respect the domain's boundary, especially when the domain is not convex or has holes. One solution to this problem is to use a constrained Delaunay triangulation (CDT). To put it simple, the domain boundary is treated as constraints that are required to become an edge of the resulting triangulation. Apart from respecting the domain boundary, CDT still enjoys many optimal properties, similar to the Delaunay triangulation.

Recall that the Delaunay triangulation is defined upon a set $\mathcal{V}$ of vertices, and the outcome is always the convex hull of $\mathcal{V}$. In contrast, CDT is defined over a complex composed of

points, edges, and polygons. The points serve the same purpose as before; the edges must be contained in some triangles in CDT; the polygons specify the regions to be triangulated. Note that the polygons can be quite general; they are not necessarily convex, and may have holes.

The following definition from [42] formalizes the complex on which CDT is defined.

Definition 3.2.13. *(Piecewise Linear Complex) In $\mathbb{R}^2$, a piecewise linear complex (PLC) $\mathcal{X}$ is a finite set of vertices, edges, and polygons that satisfies the following properties.*

- *The vertices and edges in $\mathcal{X}$ form a simplicial complex. That is, $\mathcal{X}$ contains both vertices of every edge in $\mathcal{X}$, and the relative interior of an edge in $\mathcal{X}$ intersects no vertex in $\mathcal{X}$ nor any other edge in $\mathcal{X}$.*
- *For each polygon f in $\mathcal{X}$, the boundary of f is a union of edges in $\mathcal{X}$.*
- *If two polygons in $\mathcal{X}$ intersect, their intersection is a union of edges and vertices in $\mathcal{X}$.*

The underlying space of $\mathcal{X}$, denoted by $|\mathcal{X}|$, is the union of its polygons; that is, $|\mathcal{X}| = \cup_{f\in\mathcal{X}} f$.

Definition 3.2.14. *A vertex, edge, or polygon in a PLC $\mathcal{X}$ is called a linear cell in $\mathcal{X}$. The faces of a linear cell c in a PLC $\mathcal{X}$ are the linear cells in $\mathcal{X}$ that are subsets of c, including c itself.*

Definition 3.2.15. *(Triangulation of a PLC) Let $\mathcal{X}$ be a PLC in the $\mathbb{R}^2$ plane. A triangulation of $\mathcal{X}$ is a simplicial complex $\triangle$ such that*

- *$\mathcal{X}$ and $\triangle$ have the same vertices.*
- *$\triangle$ contains every edge in $\mathcal{X}$ (and perhaps additional edges).*
- *$|\triangle| = |\mathcal{X}|$.*

The following theorem guarantees the existence of a triangulation of a PLC (cf. [42]).

Theorem 3.2.16. *Every PLC in the $\mathbb{R}^2$ plane has a triangulation.*

Sometimes we need to allow the triangulation $\triangle$ to have extra vertices for the sake of higher quality of the triangles. And this leads to a Steiner triangulation (cf. [42]).

Definition 3.2.17. *(Steiner Triangulation of a PLC) Let $\mathcal{X}$ be a PLC in the $\mathbb{R}^2$ plane. A Steiner triangulation of $\mathcal{X}$ is a simplicial complex $\triangle$ such that*

- *$\triangle$ contains every vertex in $\mathcal{X}$ (and perhaps additional vertices).*
- *every edge in $\mathcal{X}$ is a union of edges in $\triangle$.*
- *$|\triangle| = |\mathcal{X}|$.*

From Theorem 3.2.16, we see that given a PLC, there is a triangulation of it, which respects the boundary of the domain. However, this preliminary triangulation might contain triangles of bad shapes. If we still want to maintain some of the advantages of Delaunay triangulations, the constrained Delaunay triangulation (CDT) needs to be introduced with the requirement that all triangles be Delaunay relaxed (cf. [42]).

Some necessary notions need to be brought up before defining CDT.

Definition 3.2.18. *A simplex σ respects a PLC $\mathcal{X}$ if $\sigma \subseteq |\mathcal{X}|$ and for every linear cell c in $\mathcal{X}$ that intersect σ, $c \cap \sigma$ is a union of faces of σ.*

Definition 3.2.19. *Two points a and b are visible to each other if the line segment ab respects $\mathcal{X}$. A linear cell in $\mathcal{X}$ that intersects the relative interior of ab but does not include ab is said to occlude the visibility between a and b.*

Definition 3.2.20. *(Constrained Delaunay) Given a PLC $\mathcal{X}$, a simplex σ is constrained Delaunay if it satisfies the following three conditions.*

- *$\mathcal{X}$ contains all vertices of σ.*
- *σ respects $\mathcal{X}$.*
- *There is a circumcircle of σ that encloses no vertex in $\mathcal{X}$ that is visible from any point in the relative interior of σ.*

Definition 3.2.21. *(Constrained Delaunay Triangulation) A constrained Delaunay triangulation (CDT) of a PLC $\mathcal{X}$ is a triangulation of $\mathcal{X}$ in which every triangle is constrained Delaunay.*

Theorem 3.2.7 can be generalized to the constrained case (cf. [42]).

Theorem 3.2.22. *(Constrained Delaunay Lemma) Let $\triangle$ be a triangulation of a PLC $\mathcal{X}$. The following three statements are equivalent.*

- *Every triangle $T \in \triangle$ is constrained Delaunay (i.e. $\triangle$ is constrained Delaunay).*
- *Every edge e of $\triangle$ that is not in $\mathcal{X}$ is constrained Delaunay.*
- *Every edge e of $\triangle$ that is not in $\mathcal{X}$ is locally Delaunay.*

Algorithm 3.2.9 can be modified to construct a constrained Delaunay triangulation. We only need to make sure that the algorithm never flips an edge that is in $\mathcal{X}$.

Algorithm 3.2.23. *(Constrained Flip Algorithm)*

1. *Construct an initial triangulation $\triangle^{(0)}$ of $\mathcal{X}$ using any method, for example, [10, Chapter 3]. Set $k := 0$.*
2. *Do until no longer possible:*
   Find an edge e of $\triangle^{(k)}$ which is not in $\mathcal{X}$ and not locally Delaunay, and flip it. Increase k by one, and let $\triangle^{(k)}$ be the new triangulation.

Algorithm 3.2.23 can terminate in a finite number of steps, as indicated in the following theorem (cf. [42]).

Theorem 3.2.24. *Given a triangulation of a PLC having n vertices, Algorithm 3.2.23 terminates after $O(n^2)$ edge flips, yielding a constrained Delaunay triangulation.*

Corollary 3.2.25. *Every PLC has a constrained Delaunay triangulation.*

As expected, CDT may not be unique. The following theorem claims that the CDT has the almost same optimal properties as the Delaunay triangulation (cf. [42]).

Theorem 3.2.26. *Among all the triangulations of a PLC, every constrained Delaunay triangulation maximizes the minimum angle in the triangulation, minimizes the largest circumcircle, and minimizes the largest min-containment circle.*

3.2.4 Algorithms for Constructing Constrained Delaunay Triangulations

The flip algorithms 3.2.9 and 3.2.23 can be slow in practice. Fortunately, many other efficient methods are available. There are basically three classic types of algorithms. The first type is the gift-wrapping algorithms (cf. [17]). Gift-wrapping algorithms construct Delaunay triangles one at a time. The second type is the divide-and-conquer algorithm, which partitions a set of points into two halves, recursively computes the Delaunay triangulation of each subset, and merges them into one (cf. [12]). The last type is the incremental insertion algorithms which insert vertices into a Delaunay triangulation one at a time while maintaining the Delaunay properties (cf. [29, 47]). This type of methods is perhaps the most widely used. All three types can be extended to constrained Delaunay triangulations.

MATLAB offers a fast built-in function to compute the CDT of a PLC $\mathcal{X}$. However, the output is always the convex hull of the vertices in $\mathcal{X}$. In order to generate a triangulation respecting the domain's boundary, and improve its quality, we present the following algorithm.

Algorithm 3.2.27. *(2D Constrained Delaunay Triangulation)*

1. *Given a set $\mathcal{V}$ of vertices, the boundary $\mathcal{B}$ of a domain, and a set $\mathcal{E}$ of constrained edges, pick additional vertices inside the domain and on the constrained edges as appropriate. Let $\widetilde{\mathcal{V}}$ be the set of these additional vertices.*

2. *Construct a constrained Delaunay triangulation of* $\mathcal{V} \cup \widetilde{\mathcal{V}}$ *with constraints* $\mathcal{B}$ *and* $\mathcal{E}$ *using any algorithm.*

3. *Delete those triangles outside the domain.*

In the first step of Algorithm 3.2.27, we pick some extra vertices to make sure that the triangles have the desired size and shape. There are many strategies as to how to choose these vertices, depending on the specific application. For example, if one needs a finer triangulation in a certain subdomain, he can pick more vertices in that area. In general, we can just pick uniformly distributed vertices over the entire domain. Note that sometimes we also need to pick some vertices on the edges in $\mathcal{B}$ and $\mathcal{E}$ to partition them so that the quality of the triangulation nearby can be improved. This idea of adding vertices comes from the Steiner triangulation (Definition 3.2.17).

In the second step, one option is to use MATLAB's built-in function delaunayTriangulation.m. The output triangulation is the convex hull of all the vertices, which might not respect the boundary of the domain.

In the last step, we delete all those triangles which are outside the domain. The resulting triangulation must respect the boundary because we included $\mathcal{B}$ in the constraints in the second step. In order to determine whether a triangle lies inside a polygonal domain, we can first find the center point of the triangle, and then check if this point is in the polygonal domain. Thus, it reduces to a point-in-polygon problem. Again, MATLAB offers a built-in function inpolygon.m to do that, of which the idea stems from [19].

3.2.5 Examples

Example 3.2.28. *Fig. 3.3 shows a Delaunay triangulation of a convex polygon. Figure 3.4 is a triangulation which is refined from Figure 3.3 by splitting each triangle into four subtriangles by connecting the midpoints of the edges with straight lines.*

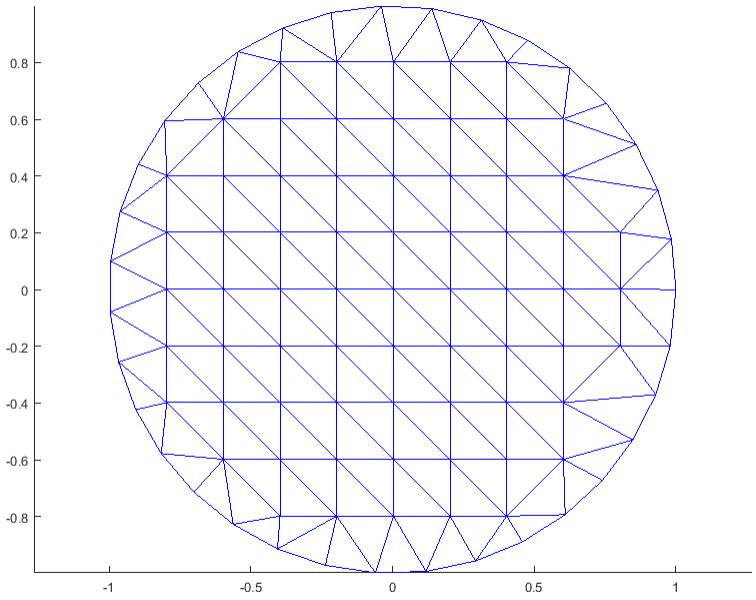

Figure 3.3: A Delaunay triangulation of a convex polygon.

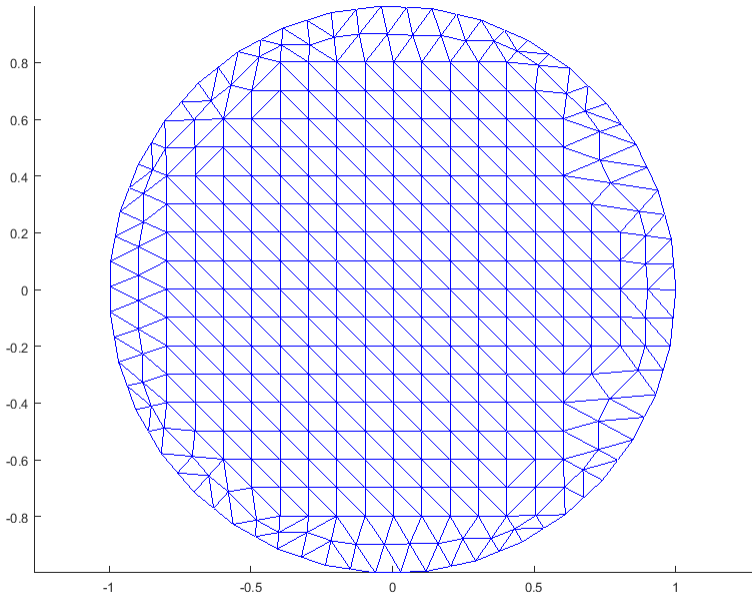

Figure 3.4: A refined triangulation of Figure 3.3.

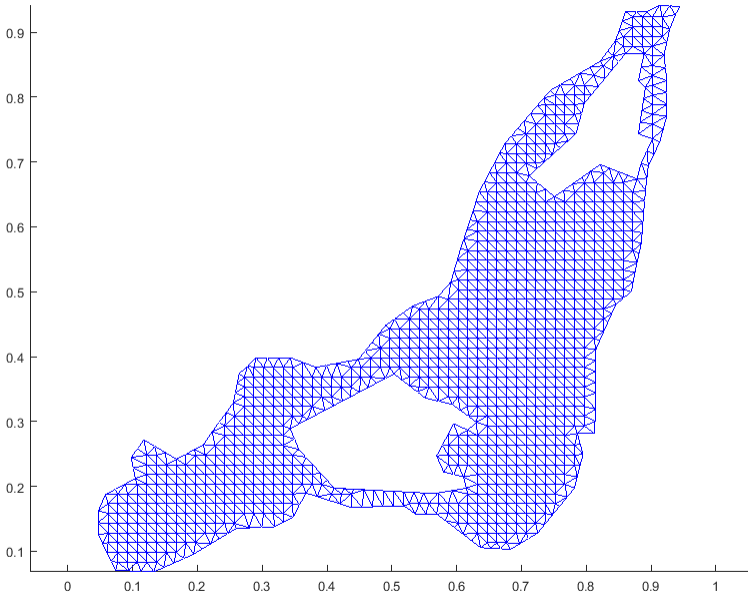

Figure 3.5: A Delaunay triangulation of a non-convex polygonal domain with two holes.

Example 3.2.29. *Fig. 3.5 shows a Delaunay triangulation of a non-convex polygonal domain with two holes.*

Example 3.2.30. *Fig. 3.6 shows a constrained Delaunay triangulation, where the red line segments are constrained edges. Observe how the triangles are assembled around those constrained edges.*

3.3 Three-Dimensional Delaunay Tetrahedralizations

In this section, we discuss three-dimensional Delaunay tetrahedralizations.

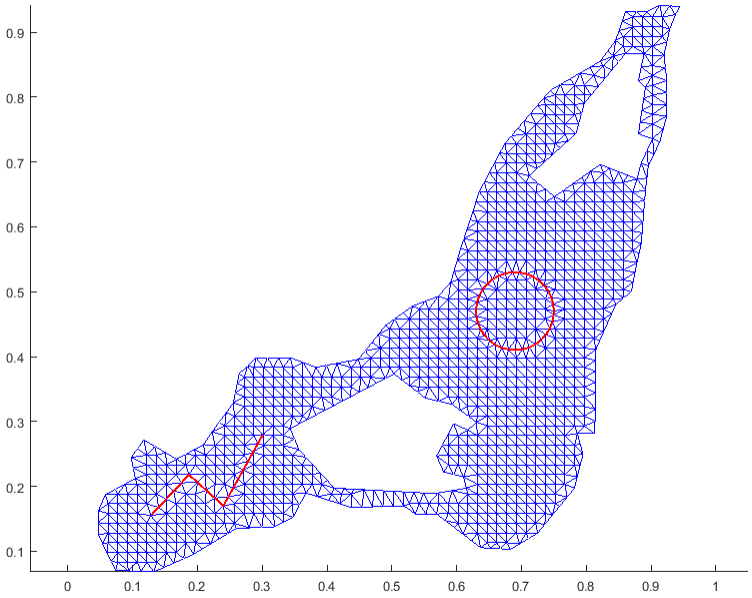

Figure 3.6: A constrained Delaunay triangulation.

3.3.1 Tetrahedralizations

Three-dimensional triangulations are sometimes called tetrahedralizations, since the elements of the mesh are now tetrahedrons. We start with a very general definition of tetrahedralizations (cf. [27]).

Definition 3.3.1. *A collection* $\triangle := \{T_1, \ldots, T_N\}$ *of tetrahedrons in* $\mathbb{R}^3$ *is called a tetrahedralization of* $\Omega = \cup_{i=1}^{N} T_i$ *provided that if a pair of tetrahedrons in* $\triangle$ *intersect, then their intersection is either a common vertex, a common edge, or a common triangular face.*

This definition is general enough to include most of the tetrahedralizations encountered in applications. Perhaps most importantly, it allows tetrahedralizations of a domain Ω with one or more holes and cavities.

Definition 3.3.2. *The vertices of the tetrahedrons of $\triangle$ are called the vertices of the tetrahedralization $\triangle$. If a vertex v is a boundary point of Ω, we say that it is a boundary vertex. Otherwise, it is called an interior vertex. Similarly, the edges and faces of the tetrahedrons of $\triangle$ are called the edges and faces of the tetrahedralization $\triangle$. If an edge or face lies on the boundary of Ω, we say that it is a boundary edge or boundary face. Otherwise, it is called an interior edge or interior face.*

3.3.2 Delaunay Tetrahedralizations

As in the two-dimensional case, a nice shape of the tetrahedrons in the tetrahedralizations is preferred. The Delaunay tetrahedralization is one with some of the nice properties. To define it geometrically, we need to use circumspheres.

Definition 3.3.3. *The circumsphere of a tetrahedron is the unique sphere that passed through all four of its vertices. A circumsphere of a triangular face is any sphere that passes through all three of its vertices. A circumsphere of an edge is any sphere that passes through both its vertices.*

Definition 3.3.4. *(Delaunay) Suppose Ω is a convex polyhedral domain and $\triangle$ is a tetrahedralization of Ω. A tetrahedron $T \in \triangle$ is Delaunay if there are no vertices of $\triangle$ inside the circumsphere of T. A face or edge of $\triangle$ is Delaunay if it has at least one empty circumsphere. The tetrahedralization $\triangle$ is Delaunay if every tetrahedron in it is Delaunay.*

There is a useful alternative characterization of the Delaunay tetrahedralization as shown in the following definition (cf. [42]).

Definition 3.3.5. *(Locally Delaunay) Let f be a face of a tetrahedralization $\triangle$. If f is the face of only one tetrahedron, then f is said to be locally Delaunay. If f is the face of two tetrahedrons T_1 and T_2, then f is locally Delaunay if it has a circumsphere enclosing no vertex of T_1 nor T_2. Equivalently, the circumsphere of T_1 encloses no vertex of T_2. Equivalently, the circumsphere of T_2 encloses no vertex of T_1.*

One of the most important results about Delaunay tetrahedralization is the Delaunay Lemma, as shown below (cf. [42]).

Theorem 3.3.6. *(The Delaunay Lemma) Suppose Ω is a convex polyhedral domain and $\triangle$ is a tetrahedralization of Ω. The following three statements are equivalent.*

1. *Every tetrahedron $T \in \triangle$ is Delaunay (i.e. $\triangle$ is Delaunay).*
2. *Every face f of $\triangle$ is Delaunay.*
3. *Every face f of $\triangle$ is locally Delaunay.*

Given a set $\mathcal{V}$ of vertices in $\mathbb{R}^3$, there exists a Delaunay tetrahedralization of the convex hull of $\mathcal{V}$, as implied by the following theorem (cf. [42]).

Theorem 3.3.7. *Every finite set of points in $\mathbb{R}^3$ has a Delaunay tetrahedralization.*

Although Delaunay tetrahedralization may not be unique for a given set of vertices, they do optimize several geometric criteria. The following theorem illustrates one nice property (cf. [42]), where the min-containment sphere of a tetrahedron is the smallest sphere that encloses it.

Theorem 3.3.8. *Among all the tetrahedralizations of a point set $\mathcal{V}$, every Delaunay tetrahedralization of $\mathcal{V}$ minimizes the largest min-containment sphere.*

3.3.3 Constrained Delaunay Tetrahedralizations

The Delaunay tetrahedralizations we discussed in the last subsection might not respect the domain's boundary, especially when the domain is not convex or has holes or cavities. One solution to this problem is to use a constrained Delaunay tetrahedralization (CDT). To put it simple, the domain boundary is treated as constraints that are required to become a face of the resulting tetrahedralization. Apart from respecting the domain boundary, CDT still enjoys many optimal properties, similar to the Delaunay tetrahedralization.

Recall that the Delaunay tetrahedralization is defined upon a set $\mathcal{V}$ of vertices, and the outcome is always the convex hull of $\mathcal{V}$. In contrast, CDT is defined over a complex composed of points, edges, polygons, and polyhedrons. The points serve the same purpose as before; the edges must be contained in some tetrahedrons in CDT; the polygons must be a union of faces of tetrahedrons; the polyhedrons specify the regions to be tetrahedralized. Note that the polyhedrons can be quite general; they are not necessarily convex, and may have holes or cavities.

The following definition from [42] formalizes the complex on which CDT is defined.

Definition 3.3.9. *(Piecewise Linear Complex) In $\mathbb{R}^3$, a piecewise linear complex (PLC) $\mathcal{X}$ is a finite set of linear cells (vertices, edges, polygons, and polyhedrons) that satisfies the following properties.*

- *The vertices and edges in $\mathcal{X}$ form a simplicial complex.*
- *For each linear cell in $\mathcal{X}$, its boundary is a union of linear cells in $\mathcal{X}$.*
- *If two distinct linear cells f and g in $\mathcal{X}$ intersect, their intersection is a union of linear cells in $\mathcal{X}$, all having lower dimension than at least one of f or g.*

The underlying space of $\mathcal{X}$, denoted by $|\mathcal{X}|$, is the union of its polyhedrons; that is, $|\mathcal{X}| = \cup_{p\in\mathcal{X}} p$.

Definition 3.3.10. *The faces of a linear cell c in a PLC $\mathcal{X}$ are the linear cells in $\mathcal{X}$ that are subsets of c, including c itself.*

Definition 3.3.11. *(Tetrahedralization of a PLC) Let $\mathcal{X}$ be a PLC in $\mathbb{R}^3$. A tetrahedralization of $\mathcal{X}$ is a simplicial complex $\triangle$ such that*

- *$\mathcal{X}$ and $\triangle$ have the same vertices.*
- *Every linear cell in $\mathcal{X}$ is a union of simplices in $\triangle$.*
- *$|\triangle| = |\mathcal{X}|$.*

Some necessary notions need to be brought up before defining CDT.

Definition 3.3.12. *A simplex σ respects a PLC $\mathcal{X}$ if $\sigma \subseteq |\mathcal{X}|$ and for every linear cell c in $\mathcal{X}$ that intersect σ, $c \cap \sigma$ is a union of faces of σ.*

Definition 3.3.13. *Two points a and b are visible to each other if $|\mathcal{X}|$ includes the open line segment connecting the two points, but no linear cell in $\mathcal{X}$ intersects only part of that open line segment. A linear cell in $\mathcal{X}$ that intersects the open line segment but does not entirely include it is said to occlude the visibility between a and b.*

Definition 3.3.14. *(Constrained Delaunay) Given a PLC $\mathcal{X}$, a simplex σ is constrained Delaunay if it satisfies the following three conditions.*

- *$\mathcal{X}$ contains all vertices of σ.*
- *σ respects $\mathcal{X}$.*
- *There is a circumsphere of σ that encloses no vertex in $\mathcal{X}$ that is visible from any point in the relative interior of σ.*

Definition 3.3.15. *(Constrained Delaunay Tetrahedralization) A constrained Delaunay tetrahedralization (CDT) of a PLC $\mathcal{X}$ is a tetrahedralization of $\mathcal{X}$ in which every tetrahedron is constrained Delaunay.*

Unlike the two-dimensional case, not all PLCs have constrained Delaunay tetrahedralizations. Figure 3.7 is one such example from [42].

Sometimes we can solve this problem by adding more vertices. The result would be a Steiner CDT.

Definition 3.3.16. *(Steiner Tetrahedralization of a PLC) Let $\mathcal{X}$ be a PLC in $\mathbb{R}^3$. A Steiner tetrahedralization of $\mathcal{X}$ is a simplicial complex $\triangle$ such that*

- *$\triangle$ contains every vertex in $\mathcal{X}$ (and perhaps additional vertices).*
- *Every linear cell in $\mathcal{X}$ is a union of simplices in $\triangle$.*
- $|\triangle| = |\mathcal{X}|$.

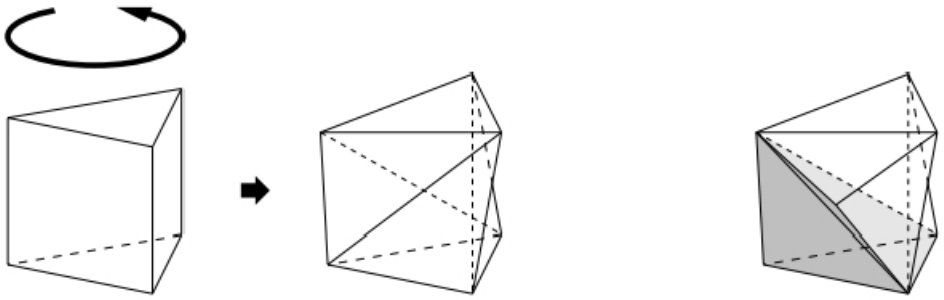

Figure 3.7: An example from [42] that doesn't have a constrained Delaunay tetrahedralization.

3.3.4 Algorithms for Constructing Constrained Delaunay Tetrahedralizations

Incremental insertion and gift-wrapping algorithms are the most popular algorithms for constructing Delaunay tetrahedralizations of the convex hull of a finite set of points (cf. [42]). Constrained Delaunay tetrahedralizations, however, is much more difficult, partly because not all PLCs have CDTs.

MATLAB offers a fast built-in function to compute Delaunay tetrahedralizations. However, the output is always the convex hull of a finite set of points. In order to generate a tetrahedralization respecting the domain's boundary, we present the following algorithm.

Algorithm 3.3.17. *(3D Constrained Delaunay Tetrahedralization)*

1. *Given a set of vertices, the boundary faces $\mathcal{B}$ of a domain, and a set $\mathcal{C}$ of constrained faces, pick additional vertices inside the domain and on the constrained faces as appropriate. Let $\mathcal{V}$ be the set of the original vertices and those additional vertices.*

2. *Construct a Delaunay tetrahedralization $\triangle$ of $\mathcal{V}$. Let $\widetilde{\mathcal{V}} := \emptyset$.*

3. *For each edge e of $\triangle$, if e intersects a face $f \in \mathcal{B} \cup \mathcal{C}$ at a point $v \notin \mathcal{V}$, put v into $\widetilde{\mathcal{V}}$.*

4. *For each edge e of $\mathcal{B} \cup \mathcal{C}$, if e intersects a triangular face f of $\triangle$ at a point $v \notin \mathcal{V}$, put v into $\widetilde{\mathcal{V}}$.*

5. *If $\widetilde{\mathcal{V}} \neq \emptyset$, set $\mathcal{V} := \mathcal{V} \cup \widetilde{\mathcal{V}}$ and go back to Step 2. Otherwise, delete those tetrahedrons outside the domain, and output the result.*

In the first step of Algorithm 3.3.17, we pick some extra vertices to make sure that the tetrahedrons have the desired size and shape. There are many strategies as to how to choose these vertices, depending on the specific application. For example, if one needs a finer tetrahedralization in a certain subdomain, he can pick more vertices in that area. In general, we can just pick uniformly distributed vertices over the entire domain. Note that sometimes we also need to pick some vertices on the faces in $\mathcal{B}$ and $\mathcal{C}$ to partition them so that the quality of the tetrahedralization nearby can be improved. This idea of adding vertices comes from the Steiner tetrahedralization (Definition 3.3.16).

In the second step, one option is to use MATLAB's built-in function delaunayTriangulation.m. The output tetrahedralization is the convex hull of all the vertices, which might not respect the boundary of the domain.

In the third and fourth step, we find all the intersection points which are not in $\mathcal{V}$. Those intersection points indicate that there exist tetrahedrons which don't respect the boundary or constrained faces.

In the fifth step, put those points into $\mathcal{V}$, and go back to Step 2. If no such points are detected, delete all those tetrahedrons which are outside the domain, then exit. In order to determine whether a tetrahedron lies inside a polyhedral domain, we can first find the center point of the tetrahedron, and then check if this point is in the polyhedral domain. Thus, it reduces to a point-in-polyhedron problem. One solution is to follow [31].

3.3.5 Examples

Example 3.3.18. *Fig. 3.8 is a Delaunay tetrahedralization of a standard cube.*

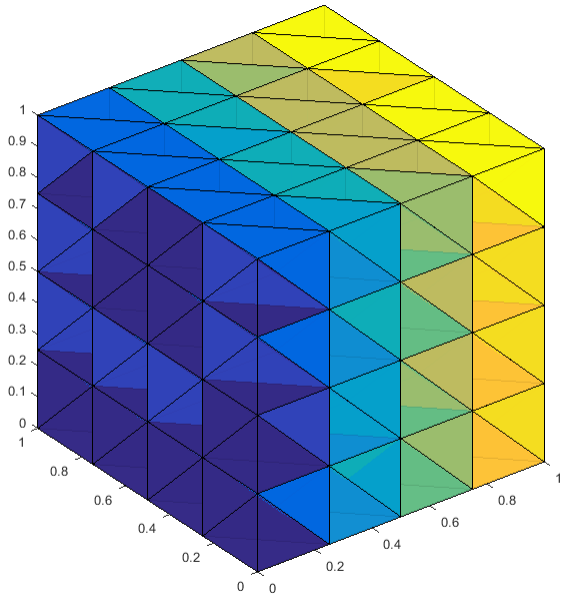

Figure 3.8: A Delaunay tetrahedralization of a cube.

Example 3.3.19. *Fig. 3.9 is a Delaunay tetrahedralization of a non-convex polyhedral domain.*

Example 3.3.20. *Fig. 3.10 is a Delaunay tetrahedralization of a cube with two tunnels.*

Example 3.3.21. *Fig. 3.11 is a Delaunay tetrahedralization of a torus-shape domain.*

Example 3.3.22. *Fig. 3.12 is a Delaunay tetrahedralization of a human head shape.*

3.3.6 Remarks

Algorithm 3.3.17 only works for domains with simple structures. It depends heavily on MATLAB's built-in function delaunayTriangulation.m. There are many literatures about

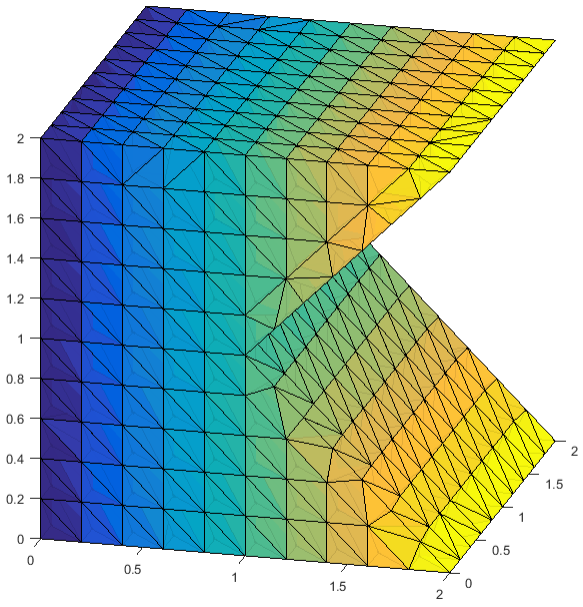

Figure 3.9: A Delaunay tetrahedralization of a non-convex polyhedral domain.

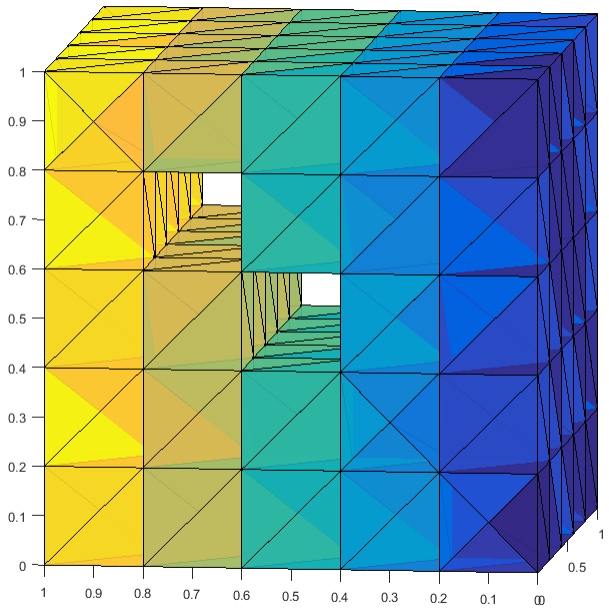

Figure 3.10: A Delaunay tetrahedralization of a cube with two tunnels.

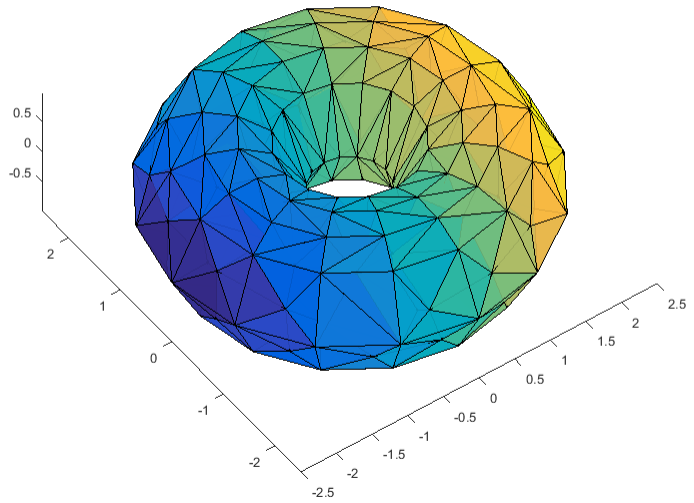

Figure 3.11: A Delaunay tetrahedralization of a torus-shape domain.

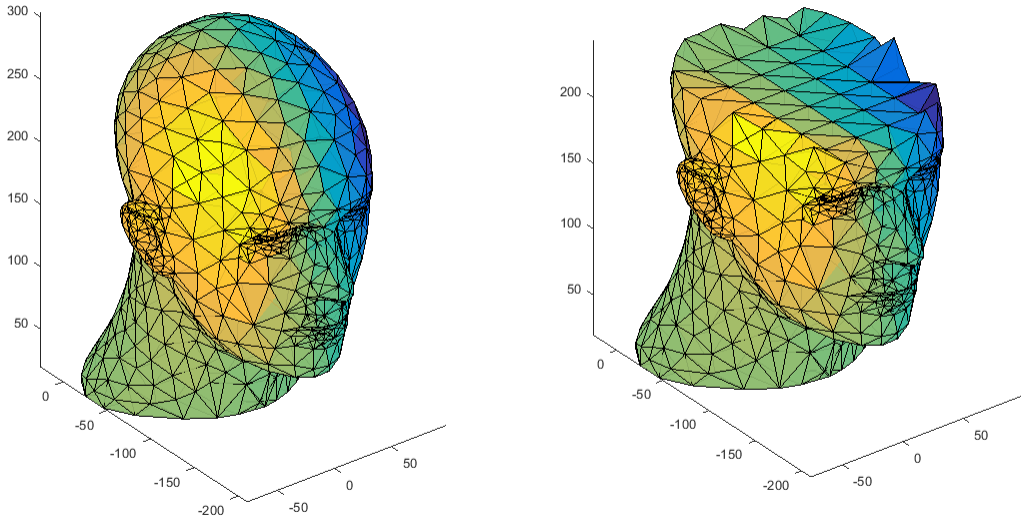

Figure 3.12: The left figure is a Delaunay tetrahedralization of a human head shape. The right one shows the inner structure.

constrained Delaunay tetrahedralizations. It would be beneficial to investigate some of them and find a good algorithm which suits our spline applications. Further study is needed in this area.

Chapter 4

Randomized Block Coordinate Descent Method

In this chapter, we introduce a randomized block coordinate descent method for minimizing convex problems with linearly coupled constraints. This method lays the foundation of our applications in later chapters.

4.1 Introduction

Coordinate descent (CD) methods are one of the simplest schemes for optimization, and have been studied for many decades. These days with the advance of parallel and distributed computing, CD methods are gaining renewed attention. Many applications have witnessed the new development. See [21], [20], [22], [38], and [41] for some successful examples.

Randomization plays a non-negligible role in the theoretical and practical success of CD methods. [34] proposed a randomized block CD method for minimizing a separable objective convex problem with linearly coupled constraints. They also extended their main algorithm, where two blocks are updated per iteration, to a more general version, where more than two blocks are updated per iteration. [37] developed a randomized block CD method for a composite objective convex problem with non-separable linear constraints. But at each iteration, only two blocks are updated.

In this chapter, we combine the ideas from [34] and [37]. Our algorithm works for a composite objective convex problem with non-separable linear constraints. We update $M \geq 2$ blocks per iteration. As we shall see in the later chapters, our algorithm is quite suitable for our applications.

4.2 Problem Formulation and Notations

We work in the space $\mathbb{R}^n$ composed by column vectors.

The optimization problem we are solving is

$$\min_x \quad f(x) \tag{4.1}$$

$$\text{s.t.} \quad Ax = 0, \tag{4.2}$$

where $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m \times n}$.

We assume that the entire space $\mathbb{R}^n$ is decomposed into N blocks, i.e.,

$$x \;=\; [x_1^\tau, \ldots, x_N^\tau]^\tau, \tag{4.3}$$

where τ is matrix transpose, $x_i \in \mathbb{R}^{n_i}$ for $i = 1, \ldots, N$, and $n = \sum_{i=1}^N n_i$.

For any matrix B with n columns, we use B_i to denote the columns of B corresponding to x_i. If $\Psi \subseteq \{1, \ldots, N\}$, then B_Ψ represents the columns of B corresponding to the coordinate blocks $\{x_i : i \in \Psi\}$. We use U to denote the $n \times n$ identity matrix and hence U_i is a matrix that places an n_i dimensional vector into the corresponding block of an n dimensional vector.

For a differentiable function f, we use f_i and $\nabla_i f$ to denote the restriction of the function and its gradient to the block x_i. And for $\Psi \subseteq \{1, \ldots, N\}$, we use f_Ψ and $\nabla_\Psi f$ to denote the restriction of the function and its gradient to the coordinate blocks $\{x_i : i \in \Psi\}$.

4.3 Algorithm

In this section, we present the randomized block CD method for the optimization problem (4.1-4.2).

We assume that the function f has L-Lipschitz continuous gradient ∇f, i.e.,

$$\|\nabla f(x) - \nabla f(x+d)\| \quad \leq \quad L\|d\|, \tag{4.4}$$

for all $x, d \in \mathbb{R}^n$.

The following result is standard (cf. [3]).

Lemma 4.3.1. *(The Descent Lemma) For any function $f : \mathbb{R}^n \to \mathbb{R}$ with L-Lipschitz continuous gradient ∇f, we have*

$$f(x+d) \quad \leq \quad f(x) + \langle \nabla f(x), d\rangle + \frac{L}{2}\|d\|^2 \tag{4.5}$$

for any $x, d \in \mathbb{R}^n$.

We start with a feasible point $x^{(0)}$. For example, take $x^{(0)} = 0$. Pick a positive integer $M \leq N$. Then, at each iteration we update M blocks in the vector x. For simplicity of the exposition, we use the following notation: given the current iterate x, denote the next iterate by $x^\ddagger$. Similar notations apply to the other variables.

To be more specific, randomly choose $\Psi = \{i_1, \ldots, i_M\}$ with $i_l \in \{1, \ldots, N\}$ with probability p_Ψ. By Lemma 4.3.1, the following inequality holds:

$$f\left(x + \sum_{i\in\Psi} U_i d_i\right) \quad \leq \quad f(x) + \langle \nabla_\Psi f(x), d_\Psi\rangle + \frac{L}{2}\|d_\Psi\|^2, \tag{4.6}$$

where $d_\Psi = [d_{i_1}^\tau, \ldots, d_{i_M}^\tau]^\tau$. Based on the inequality (4.6), let the next iterate be chosen as

$$x^\ddagger \quad = \quad x + \sum_{i\in\Psi} U_i d_i, \tag{4.7}$$

where the direction d_Ψ is determined by requiring the next iterate $x^\ddagger$ to be also feasible and minimizing the right-hand side of (4.6), i.e.,

$$d_\Psi \quad = \quad \arg\min_{A_\Psi d_\Psi = 0} f(x) + \langle \nabla_\Psi f(x), d_\Psi\rangle + \frac{L}{2}\|d_\Psi\|^2. \tag{4.8}$$

We can apply the Lagrange multiplier method (cf. [3]) to get an explicit solution to (4.8), as shown in the following lemma.

Lemma 4.3.2. *The solution to (4.8) is*

$$d_i = -\frac{1}{L}\nabla_i f(x) + \frac{1}{L}A_i^\tau \lambda \tag{4.9}$$

for each $i \in \Psi$*, where*

$$\lambda = \left(\sum_{j\in\Psi} A_j A_j^\tau\right)^+ \left(\sum_{j\in\Psi} A_j \nabla_j f(x)\right) \tag{4.10}$$

with $+$ *denoting the pseudo-inverse.*

To sum up, we obtain the following randomized block coordinate descent method.

Algorithm 4.3.3. *(Randomized Block Coordinate Descent Method)*

1. *Take an initial value* $x^{(0)}$ *such that* $Ax^{(0)} = 0$
2. ***for*** $k \geq 0$ ***do***
3. *Choose an* M*-tuple* $\Psi^{(k)} = \{i_1^{(k)}, \ldots, i_M^{(k)}\}$ *with probability* $p_{\Psi^{(k)}}$
4. *Solve (4.8) to get* $d_{\Psi^{(k)}}$ *by applying (4.9)*
5. *Set* $x^{(k+1)} := x^{(k)} + U_{\Psi^{(k)}} d_{\Psi^{(k)}}$
6. *k:=k+1*
7. ***end for***

4.4 Reduction of General Case

Before we analyze convergence, it would be convenient if we can reduce the original problem (4.1-4.2) to a simpler one, which is the topic of this section. We will follow the idea from [37].

We write the constraint (4.2) into

$$\sum_{i=1}^{N} A_i x_i = 0, \tag{4.11}$$

where $A = [A_1, \ldots, A_N]$ with $A_i \in \mathbb{R}^{m\times n_i}$ for $i = 1, \ldots, N$.

For each $i = 1, \ldots, N$, define $y_i = A_i x_i$ so that $\sum_{i=1}^N y_i = 0$. Use singular value decomposition to write $A_i = P_i \Sigma_i Q_i^*$. Then

$$Q_i^* x_i = \Sigma_i^+ P_i^* y_i + W_i z_i \tag{4.12}$$

for some z_i and W_i with orthonormal columns satisfying range(W_i) = kernel(Σ_i), where $+$ denotes the pseudo-inverse. Hence, we have

$$x_i \;\; = \;\; A_i^+ y_i + \widetilde{A_i} z_i, \tag{4.13}$$

where $A_i^+ = Q_i \Sigma_i^+ P_i^*$ and $\widetilde{A_i} = Q_i W_i$. Note that $\widetilde{A_i}$ has orthonormal columns and range$(\widetilde{A_i})$ = kernel(A_i). Also note that each $y_i \in \mathbb{R}^m$ but the dimension of z_i may vary for different i. Nonetheless, by adding all-zero columns to $\widetilde{A_i}$, we can still assume that the dimensions of z_i are the same for all i. Let $y = [y_1^\tau, \ldots, y_N^\tau]^\tau$ and $z = [z_1^\tau, \ldots, z_N^\tau]^\tau$.

The problem (4.1-4.2) can then be written as

$$\min_{y,z} \quad g(y,z) \tag{4.14}$$

$$\text{s.t.} \quad \sum_{i=1}^N y_i = 0, , \tag{4.15}$$

where

$$g(y,z) \;\; = \;\; f\left(\sum_{i=1}^N U_i (A_i^+ y_i + \widetilde{A_i} z_i) \right). \tag{4.16}$$

The next lemma follows from the property of convex functions.

Lemma 4.4.1. *Let $f(x)$ be a convex function. Let $g(y,z)$ be the function defined in (4.16). Then $g(y,z)$ is convex.*

The following lemma can be found in [37].

Lemma 4.4.2. *Let $f(x)$ be a function with L-Lipschitz continuous gradient ∇f. Let $g(y,z)$ be the function defined in (4.16). Then we have*

$$\|\nabla_{y_i} g(y,z) - \nabla_{y_i} g(y',z)\| \;\; \leq \;\; \frac{L}{\sigma_{\min}^2(A_i)} \|y_i - y_i'\|, \tag{4.17}$$

$$\|\nabla_{z_i} g(y,z) - \nabla_{z_i} g(y',z)\| \;\; \leq \;\; L \|y_i - y_i'\|, \tag{4.18}$$

where $\sigma^2_{\min}(A_i)$ *denotes the minimum non-zero singular value of* A_i.

Here and thereafter, we will focus on the following reformulation of the original problem (4.1-4.2):

$$\min_{y,z} \quad f(y,z) \tag{4.19}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i = 0,, \tag{4.20}$$

where $y_i \in \mathbb{R}^{n_y}$ and $z_i \in \mathbb{R}^{n_z}$.

Let $x = [y^\tau, z^\tau]^\tau \in \mathbb{R}^{N(n_y+n_z)}$. Take

$$A = [\underbrace{I_{n_y\times n_y}, \ldots, I_{n_y\times n_y}}_{N}, \underbrace{O_{n_y\times n_z}, \ldots, O_{n_y\times n_z}}_{N}], \tag{4.21}$$

where $I_{n_y\times n_y}$ is the $n_y \times n_y$ identity matrix and $O_{n_y\times n_z}$ is the $n_y \times n_z$ zero matrix. Then the constraint (4.20) can be written as

$$\sum_{i=1}^{N} y_i = A \begin{bmatrix} y \\ z \end{bmatrix} = Ax = 0. \tag{4.22}$$

Under this reformulation, we can derive some useful results for Algorithm 4.3.3. Since $x = [y^\tau, z^\tau]^\tau$, we can write $x_i = [y_i^\tau, z_i^\tau]^\tau$ and $d_i = [d_{y_i}^\tau, d_{z_i}^\tau]^\tau$ for each $i \in \Psi$. Also note that the matrix A has a special structure as indicated in (4.21). Thus, we can apply the Lagrange multiplier method to get an explicit solution to (4.8), as shown in the following lemma.

Lemma 4.4.3. *The solution to (4.8) is*

$$d_{y_i} = -\frac{1}{LM} \sum_{j\in\Psi} \left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right), \tag{4.23}$$

$$d_{z_i} = -\frac{1}{L} \nabla_{z_i} f(x), \tag{4.24}$$

for each $i \in \Psi$.

4.5 Convergence Analysis

In this section, we discuss the convergence of Algorithm 4.3.3 for the reformulation (4.19-4.20). First, we have the following lemma.

Lemma 4.5.1. *Algorithm 4.3.3 is a descent method.*

Proof. Plugging (4.23) and (4.24) into (4.6), we get

$$\begin{aligned} f(x^{\ddagger}) &\leq f(x) - \frac{1}{LM}\sum_{i\in\Psi}\left\langle \nabla_{y_i} f(x), \sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\rangle - \frac{1}{L}\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2 \\ &\quad + \frac{1}{2LM^2}\sum_{i\in\Psi}\left\|\sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\|^2 + \frac{1}{2L}\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2 \\ &= f(x) - \frac{1}{LM^2}\sum_{i\in\Psi}\left\|\sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\|^2 - \frac{1}{L}\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2 \\ &\quad + \frac{1}{2LM^2}\sum_{i\in\Psi}\left\|\sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\|^2 + \frac{1}{2L}\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2 \\ &= f(x) - \frac{1}{2LM^2}\sum_{i\in\Psi}\left\|\sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\|^2 - \frac{1}{2L}\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2. \end{aligned} \tag{4.25}$$

This shows that the method is a descent method. □

After k iterations of the algorithm, we generate a random output $(x, f(x))$, which depends on the observed history of block selections:

$$\eta = \{\Psi^{(0)}, \ldots, \Psi^{(k-1)}\}. \tag{4.26}$$

Taking expectation over the choice of Ψ given η on both sides of (4.25) yields

$$\begin{aligned} \mathbb{E}[f(x^{\ddagger})|\eta] &\leq f(x) - \frac{1}{2LM^2}\sum_{\Psi} p_\Psi \left(\sum_{i\in\Psi}\left\|\sum_{j\in\Psi}\left(\nabla_{y_i} f(x) - \nabla_{y_j} f(x)\right)\right\|^2\right) \\ &\quad - \frac{1}{2L}\sum_{\Psi} p_\Psi \left(\sum_{i\in\Psi}\|\nabla_{z_i} f(x)\|^2\right) \\ &= f(x) - \frac{1}{2L}\nabla f(x)^{\tau}\mathcal{G}\nabla f(x), \end{aligned} \tag{4.27}$$

where the matrix $\mathcal{G}$ is defined as follows.

Define $\iota_\Psi \in \mathbb{R}^N$ to be a vector with components

$$\iota_i = \begin{cases} 1 & \text{if } i \in \Psi \\ 0 & \text{otherwise.} \end{cases} \tag{4.28}$$

Let $\text{diag}(\iota)$ denote the diagonal matrix with entries ι_i on the diagonal. Define

$$G_1 = \sum_\Psi p_\Psi \left(\text{diag}(\iota_\Psi) - \frac{1}{M} \iota_\Psi \iota_\Psi^\tau \right), \tag{4.29}$$

$$G_2 = \sum_\Psi p_\Psi \text{diag}(\iota_\Psi). \tag{4.30}$$

Finally define

$$\mathcal{G} = \begin{bmatrix} G_1 \otimes I_{n_y \times n_y} & 0 \\ 0 & G_2 \otimes I_{n_z \times n_z} \end{bmatrix}. \tag{4.31}$$

where $\otimes$ denotes the Kronecker product. It is easy to verify that this $\mathcal{G}$ satisfies (4.27).

We observe that G_1 can be viewed as a Laplacian matrix of a weighted graph. We can assume that this graph is connected, which is usually satisfied in our applications in later chapters. Properties of Laplacian matrices can be found in [36]. In particular, G_1 is positive semidefinite, and has a simple eigenvalue 0 with the associated eigenvector $e \in \mathbb{R}^N$ of which all entries are equal to 1. Since G_2 is positive definite, it is easy to deduce that the matrix $\mathcal{G}$ is positive semidefinite.

As in [34], we define the extended primal norm induced by the matrix $\mathcal{G}$ as

$$\|w\|_\mathcal{G} := \sqrt{w^\tau \mathcal{G} w}, \tag{4.32}$$

for any $w \in \mathbb{R}^{N(n_y+n_z)}$.

Theorem 4.5.2. *The extended norm defined by (4.32) is a seminorm. Moreover, if we write* $w = [u^\tau, v^\tau]^\tau$ *where* $u = [u_1^\tau, \ldots, u_N^\tau]^\tau \in \mathbb{R}^{Nn_y}$ *and* $v \in \mathbb{R}^{Nn_z}$, *then* $\|w\|_\mathcal{G} = 0$ *if and only if* $u_1 = u_2 = \cdots = u_N$ *and* $v = 0$.

Proof. It is straightforward to show that (4.32) is a seminorm. As to the second statement, suppose $w^\tau \mathcal{G} w = 0$. Then $v = 0$ follows from the positive definiteness of G_2. Based on the structure of G_1 in (4.29), we will now show that $a^\tau \left(\text{diag}(\iota_\Psi) - \frac{1}{M}\iota_\Psi \iota_\Psi^\tau\right) a = 0$ implies $a_1 = a_2 = \cdots = a_M$, from which $u_1 = u_2 = \cdots = u_N$ follows. Indeed,

$$\begin{aligned} 0 &= a^\tau \left(\text{diag}(\iota_\Psi) - \frac{1}{M}\iota_\Psi \iota_\Psi^\tau\right) a \\ &= \sum_{i\in\Psi} a_i \left(a_i - \frac{1}{M}\sum_{j\in\Psi} a_j\right) \\ &= \sum_{i\in\Psi} a_i^2 - \frac{1}{M}\left(\sum_{i\in\Psi} a_i\right)^2. \end{aligned}$$

Linking this with the Cauchy-Schwarz inequality, we must have $a_1 = a_2 = \cdots = a_M$. □

Define the subspace $S = \{x \in \mathbb{R}^{N(n_y+n_z)} : Ax = 0\}$ where A is defined in (4.21). The extended dual norm can be defined on S as

$$\|x\|_{\mathcal{G}}^* := \max_{w:\|w\|_{\mathcal{G}}\le 1} \langle x, w\rangle, \tag{4.33}$$

for any $x \in S$.

By the definition of dual norm, the Cauchy-Schwarz inequality holds:

$$\langle x, w\rangle \le \|x\|_{\mathcal{G}}^* \|w\|_{\mathcal{G}}, \tag{4.34}$$

for any $x \in S$ and $w \in \mathbb{R}^{N(n_y+n_z)}$.

In fact, this dual norm can be computed in the same way as in [34]. Recall that we can write $x = [y^\tau, z^\tau]^\tau$ and $w = [u^\tau, v^\tau]^\tau$. In the block form, $y = [y_1^\tau, \ldots, y_N^\tau]^\tau$, similarly for z, u, and v. Also recall that e was defined as a vector in $\mathbb{R}^N$ with all entries equal to 1. For any $x \in S$, we have

$$
\begin{aligned}
\|x\|_{\mathcal{G}}^{*} &= \max_{w:\|w\|_{\mathcal{G}}\le 1} \langle x, w\rangle \\
&= \max_{u,v:\|[u^\tau,v^\tau]^\tau\|_{\mathcal{G}}\le 1} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau\rangle \\
&= \max_{u,v:\|[u^\tau,v^\tau]^\tau\|_{\mathcal{G}}\le 1} \left\langle \begin{bmatrix} y \\ z \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} e\otimes \frac{1}{N}\sum_{i=1}^N u_i \\ 0 \end{bmatrix} \right\rangle \qquad (4.35) \\
&= \max_{u,v} \left\langle \begin{bmatrix} y \\ z \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} e\otimes \frac{1}{N}\sum_{i=1}^N u_i \\ 0 \end{bmatrix} \right\rangle \\
&\quad \text{s.t.} \left\| \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} e\otimes \frac{1}{N}\sum_{i=1}^N u_i \\ 0 \end{bmatrix} \right\|_{\mathcal{G}} \le 1 \qquad (4.36) \\
&= \max_{u,v:\ \substack{\|[u^\tau,v^\tau]^\tau\|_{\mathcal{G}}\le 1, \\ \sum_{i=1}^N u_i = 0}} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau\rangle \\
&= \max_{u,v:\ \substack{\|[u^\tau,v^\tau]^\tau\|_{\mathcal{G}}\le 1, \\ Bu = 0}} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau\rangle \qquad (4.37) \\
&= \max_{u,v:\ \substack{\|[u^\tau,v^\tau]^\tau\|_{\mathcal{G}}\le 1, \\ u^\tau B^\tau B u \le 0}} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau\rangle \\
&= \min_{\lambda\ge 0,\xi\ge 0} \max_{u,v} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau\rangle + \lambda(1 - \|[u^\tau, v^\tau]^\tau\|_{\mathcal{G}}^2) - \xi u^\tau B^\tau B u \qquad (4.38) \\
&= \min_{\lambda\ge 0,\xi\ge 0} \lambda + \frac{1}{4}\left\langle x, \left[\lambda\mathcal{G} + \xi \begin{bmatrix} B^\tau B & 0 \\ 0 & 0 \end{bmatrix}\right]^+ x \right\rangle \qquad (4.39) \\
&= \min_{\lambda\ge 0,\xi\ge 0} \lambda + \frac{1}{4\lambda}\left\langle x, \left[\mathcal{G} + \frac{\xi}{\lambda} \begin{bmatrix} B^\tau B & 0 \\ 0 & 0 \end{bmatrix}\right]^+ x \right\rangle \\
&= \min_{\zeta\ge 0}\min_{\lambda\ge 0} \lambda + \frac{1}{4\lambda}\left\langle x, \left[\mathcal{G} + \zeta \begin{bmatrix} B^\tau B & 0 \\ 0 & 0 \end{bmatrix}\right]^+ x \right\rangle \\
&= \min_{\zeta\ge 0} \sqrt{\left\langle x, \left[\mathcal{G} + \zeta \begin{bmatrix} B^\tau B & 0 \\ 0 & 0 \end{bmatrix}\right]^+ x \right\rangle} \qquad (4.40) \\
&= \sqrt{\langle x, \mathcal{G}^+ x\rangle}. \qquad (4.41)
\end{aligned}
$$

(4.35) follows by noting that $\langle y, e\otimes \frac{1}{N}\sum_{i=1}^N u_i\rangle = 0$ since $\sum_{i=1}^N y_i = 0$.

(4.36) follows from the fact that G_1 is a Laplacian matrix so that $G_1\left(e\otimes\frac{1}{N}\sum_{i=1}^{N}u_i\right)=0$. In (4.37), we define $B=[\underbrace{I_{n_y\times n_y},\ldots,I_{n_y\times n_y}}_{N}]$ as a submatrix of A.

The derivation of (4.38) will be explained later.

Maximizing the objective function with respect to u,v in (4.38) gives $w=[u^\tau,v^\tau]^\tau=\frac{1}{2}\left[\lambda\mathcal{G}+\xi\begin{bmatrix}B^\tau B & 0\\ 0 & 0\end{bmatrix}\right]^{+}[y^\tau,z^\tau]^\tau$, which in turn yields (4.39).

To compute (4.40), it suffices to analyze $\left[\mathcal{G}+\zeta\begin{bmatrix}B^\tau B & 0\\ 0 & 0\end{bmatrix}\right]^{+}$. Recall that G_1 is a Laplacian matrix of a weighted connected graph. It is positive semidefinite, and has a simple eigenvalue 0 with the associated eigenvector $e\in\mathbb{R}^N$ of which all entries are equal to 1. We can write $G_1=P\mathrm{diag}(\lambda_1,\lambda_2,\ldots,\lambda_N)P^\tau$, where $0=\lambda_1<\lambda_2\leq\lambda_3\leq\ldots\leq\lambda_N$ are the eigenvalues and $P=[e/\|e\|,\eta_2,\ldots,\eta_N]$ is an orthogonal matrix. Then $G_1=\lambda_2\eta_2\eta_2^\tau+\ldots,+\lambda_N\eta_N\eta_N^\tau$. Hence, we have

$$
\begin{aligned}
&\left[\mathcal{G}+\zeta\begin{bmatrix}B^\tau B & 0\\ 0 & 0\end{bmatrix}\right]^{+}\\
=&\begin{bmatrix}G_1\otimes I_{n_y\times n_y}+\zeta B^\tau B & 0\\ 0 & G_2\otimes I_{n_z\times n_z}\end{bmatrix}^{+}\\
=&\begin{bmatrix}(G_1+\zeta ee^\tau)\otimes I_{n_y\times n_y} & 0\\ 0 & G_2\otimes I_{n_z\times n_z}\end{bmatrix}^{+}\\
=&\begin{bmatrix}(\lambda_2\eta_2\eta_2^\tau+\ldots,+\lambda_N\eta_N\eta_N^\tau+\zeta ee^\tau)\otimes I_{n_y\times n_y} & 0\\ 0 & G_2\otimes I_{n_z\times n_z}\end{bmatrix}^{+}\\
=&\begin{bmatrix}(P\mathrm{diag}(\zeta\|e\|^2,\lambda_2,\ldots,\lambda_N)P^\tau)\otimes I_{n_y\times n_y} & 0\\ 0 & G_2\otimes I_{n_z\times n_z}\end{bmatrix}^{+}\\
=&\begin{bmatrix}(P\mathrm{diag}(\zeta\|e\|^2,\lambda_2,\ldots,\lambda_N)^{+}P^\tau)\otimes I_{n_y\times n_y} & 0\\ 0 & G_2^{-1}\otimes I_{n_z\times n_z}\end{bmatrix},
\end{aligned}
\tag{4.42}
$$

from which it is straightforward to see that the minimizer of (4.40) is $\zeta = 0$. Therefore, (4.41) has been justified.

It remains to justify (4.38). Since the Slater constraint qualification is not satisfied, we cannot apply the Strong Duality Theorem (cf. [5]). Instead, we use the optimality conditions in [5, Proposition 6.1.5], which guarantees the correctness of (4.38) if the following claims can be established:

- Primal Feasibility: $(\widetilde{u}, \widetilde{v})$ satisfies $\|[\widetilde{u}^\tau, \widetilde{v}^\tau]^\tau\|_{\mathcal{G}} \leq 1$, and $\widetilde{u}^\tau B^\tau B \widetilde{u} \leq 0$,
- Dual Feasibility: $\widetilde{\lambda} \geq 0$ and $\widetilde{\xi} \geq 0$,
- Lagrangian Optimality:
  $(\widetilde{u}, \widetilde{v}) \in \arg\max_{u,v} \langle [y^\tau, z^\tau]^\tau, [u^\tau, v^\tau]^\tau \rangle + \widetilde{\lambda}(1 - \|[u^\tau, v^\tau]^\tau\|_{\mathcal{G}}^2) - \widetilde{\xi} u^\tau B^\tau B u$,
- Complementary Slackness: $\widetilde{\lambda}(1 - \|[\widetilde{u}^\tau, \widetilde{v}^\tau]^\tau\|_{\mathcal{G}}^2) = 0$ and $\widetilde{\xi} \widetilde{u}^\tau B^\tau B \widetilde{u} = 0$,

where $\widetilde{\lambda} = \frac{1}{2}\sqrt{x^\tau \mathcal{G}^+ x}$, $\widetilde{\xi} = 0$, and $[\widetilde{u}^\tau, \widetilde{v}^\tau]^\tau = \frac{1}{2}\left[\widetilde{\lambda}\mathcal{G} + \widetilde{\xi}\begin{bmatrix} B^\tau B & 0 \\ 0 & 0 \end{bmatrix}\right]^+ [y^\tau, z^\tau]^\tau$.

In fact, some algebraic calculations give $\|[\widetilde{u}^\tau, \widetilde{v}^\tau]^\tau\|_{\mathcal{G}} = 1$. And by [18, Lemma 2], we can deduce $B\widetilde{u} = 0$, which then implies $\widetilde{u}^\tau B^\tau B \widetilde{u} \leq 0$. Lagrangian Optimality can be verified by taking gradient and by noting that x is in the range of $\mathcal{G}$. Hence, it is now easy to see that all four claims above hold.

We are now ready to state the convergence result. The proof is similar to that in [37].

Theorem 4.5.3. *For the optimization problem (4.19-4.20), assume that f is convex and has L-Lipschitz continuous gradient. Let f^* denote the optimal value and X^* be the set of optimal solutions. If $\{x^{(k)}\}_{k\geq 0}$ is generated by Algorithm 4.3.3, then we have the following rate of convergence for the expected values of the objective function:*

$$\mathbb{E}[f(x^{(k)})] - f^* \quad \leq \quad \frac{2LR^2(x^{(0)})}{k}, \tag{4.43}$$

where

$$R(x^{(0)}) \quad = \quad \max_{\substack{x: \\ f(x)\leq f(x^{(0)})}} \min_{\substack{Ax=0 \\ x^* \in X^*}} \|x - x^*\|_{\mathcal{G}}^*. \tag{4.44}$$

Proof. From the convexity of f and the Cauchy-Schwarz inequality (4.34) of the norm $\|\cdot\|_{\mathcal{G}}$, we have

$$\begin{aligned} f(x^{(k)}) - f^* &\leq \langle \nabla f(x^{(k)}), x^{(k)} - x^* \rangle \\ &\leq \|x^{(k)} - x^*\|_{\mathcal{G}}^* \|\nabla f(x^{(k)})\|_{\mathcal{G}} \\ &\leq R(x^{(0)}) \|\nabla f(x^{(k)})\|_{\mathcal{G}}. \end{aligned} \tag{4.45}$$

Combining this inequality with (4.27), we obtain

$$\mathbb{E}[f(x^{(k+1)})|\eta^{(k)}] \leq f(x^{(k)}) - \frac{1}{2L} \frac{(f(x^{(k)}) - f^*)^2}{R^2(x^{(0)})}. \tag{4.46}$$

Taking expectation of both sides of this inequality with respect to $\eta^{(k)}$ gives

$$\begin{aligned} \mathbb{E}[f(x^{(k+1)})] &\leq \mathbb{E}[f(x^{(k)})] - \frac{1}{2L} \frac{\mathbb{E}[(f(x^{(k)}) - f^*)^2]}{R^2(x^{(0)})} \\ &\leq \mathbb{E}[f(x^{(k)})] - \frac{1}{2L} \frac{(\mathbb{E}[f(x^{(k)})] - f^*)^2}{R^2(x^{(0)})}, \end{aligned} \tag{4.47}$$

where (4.47) follows from Jensen's inequality.

Denote $\Theta_k = \mathbb{E}[f(x^{(k)})] - f^*$. (4.47) now leads to

$$\Theta_{k+1} \leq \Theta_k - \frac{1}{2L} \frac{\Theta_k^2}{R^2(x^{(0)})}. \tag{4.48}$$

Dividing both sides with $\Theta_k \Theta_{k+1}$ leads to

$$\begin{aligned} \frac{1}{\Theta_k} &\leq \frac{1}{\Theta_{k+1}} - \frac{1}{2LR^2(x^{(0)})} \frac{\Theta_k}{\Theta_{k+1}} \\ &\leq \frac{1}{\Theta_{k+1}} - \frac{1}{2LR^2(x^{(0)})}, \end{aligned} \tag{4.49}$$

since $\Theta_{k+1} \leq \Theta_k$.

Adding these inequalities for k steps, we obtain $0 \leq \frac{1}{\Theta_0} \leq \frac{1}{\Theta_k} - \frac{k}{2LR^2(x^{(0)})}$ from which the statement of the theorem follows.

□

Note that the reformulation of (4.19-4.20) is presented only for the ease of theoretical analysis. In practice, we don't need to do the conversion.

4.6 Future Work

[11] presents a parallel multi-block alternating direction method of multipliers (ADMM) with $o(1/k)$ convergence. One possible future work is to apply the method in [11] to improve our result.

Chapter 5

A Randomized Domain Decomposition Method for Computing Multivariate Spline Fits of Scattered Data

A randomized domain decomposition method for solving large bivariate/trivariate scattered data fitting problems is described in this chapter. This method is based on randomly splitting the domain into smaller ones, solving the associated smaller fitting problems with boundary conditions, and iterating.

5.1 Introduction

Suppose f is a smooth function defined on a domain Ω in $\mathbb{R}^2$ or $\mathbb{R}^3$. Given the values $\{f_i := f(v_i)\}_{i=1}^N$ of f at some set of scattered points $\{v_1, \ldots, v_N\} \subset \Omega$, we consider the problem of computing a smooth function s that approximates the data, i.e.,

$$s(v_i) \approx f_i, \qquad i = 1, \ldots, N. \tag{5.1}$$

There are many methods for solving this problem, such as the minimal energy (ME) method, the discrete least-squares (DLS) method, and the penalized least-squares (PLS) method. These methods have been extensively studied in the literature (cf. [1, 45, 46, 25, 40]). It is well known that all three work well on fitting smooth functions. The major drawback is that they are global methods, which means that the coefficients of a fitting spline must be

computed from a single linear system of equations. Nowadays many real-life problems can easily contain a very large scale of data. Thus, it is usually a formidable task to find the global solution in one shot.

Some researchers have tried to solve global fitting problems by dividing the domain into smaller subdomains, computing fits on each subdomain, and then blending the patches together with some blending functions. These blending functions might produce a result that is not close to the global fit. In [28], a better domain decomposition method for bivariate splines is constructed. However, their method assumes that there are enough data points in each subdomain to uniquely determine a solution, which is too strict for real-life practice.

In this chapter, we present a different domain decomposition method. There are three major differences between our algorithm and that in [28]:

- Our algorithm is iterative while [28] is not.
- [28] fixes a decomposition of Ω at the beginning while our algorithm uses a new random decomposition at each iteration, and there might be overlapping in our decomposition.
- [28] uses a stable local minimal determining set to ensure that the final result satisfies the required smoothness. In contrast, our algorithm uses boundary conditions as constraints to enforce the smoothness.

As we shall see, our method

- is easy to implement,
- is able to deal with large data fitting problems,
- allows the solution of fitting sparse data,
- produces a spline s with the same smoothness as the global fit,
- does not use a minimal determining set (which may not be easy to find in practice).

5.2 Two-Dimensional Algorithm

In this section, we describe the two-dimensional randomized domain decomposition method for bivariate scattered data fitting problem. The algorithm is presented along with its convergence analysis. Several numerical examples are given at the end.

5.2.1 Problem Formulation

Let $\Omega \subseteq \mathbb{R}^2$ be a polygonal domain. Suppose we are given values $\{f_i := f(x_i, y_i)\}_{i=1}^N$ at points in $\mathcal{A} := \{v_i = (x_i, y_i)\}_{i=1}^N \subset \Omega$. We wish to compute a smooth function s that approximates the data. To be more specific, given $0 \leq r < d$ and a triangulation $\triangle$ of Ω, let

$$\mathcal{S}_d^r(\triangle) \;:=\; \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{5.2}$$

be the associated space of bivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of bivariate polynomials of degree at most d, as defined in Section 2.1. Also fix a constant $\lambda > 0$. The corresponding penalized least-squares (PLS) spline (cf. [1]) is defined to be:

$$s^* \;:=\; \arg\min_{s \in \mathcal{S}_d^r(\triangle)} \|s - f\|_{\mathcal{A}}^2 + \lambda E(s), \tag{5.3}$$

where

$$\|s - f\|_{\mathcal{A}}^2 \;:=\; \sum_{i=1}^N (s(x_i, y_i) - f_i)^2, \tag{5.4}$$

and

$$E(s) \;:=\; \int_\Omega \left[\left(\frac{\partial^2 s}{\partial x^2} \right)^2 + 2 \left(\frac{\partial^2 s}{\partial x \partial y} \right)^2 + \left(\frac{\partial^2 s}{\partial y^2} \right)^2 \right] dxdy. \tag{5.5}$$

Note that $E(s)$ is the well-known thin-plate energy of s.

Since $s|_T$ is a polynomial of degree d on each triangle $T \in \triangle$, we can use the B-form representation (2.15):

$$s|_T \;=\; \sum_{i+j+k=d} c_{ijk}^T B_{ijk}^d. \tag{5.6}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c \;:=\; [c^T_{i,j,k}, i+j+k=d, T\in\triangle], \tag{5.7}$$

where c is a column vector of dimension $m\binom{d+2}{2}$ with m denoting the number of triangles in $\triangle$. Moreover, using the B-form representation (2.15), we obtain

$$\|s-f\|^2_{\mathcal{A}} \;=\; \|Ac-b\|^2 \tag{5.8}$$

for some matrix A and $b=[f_1,\ldots,f_N]^\tau$. We also have

$$E(s) \;=\; c^\tau Kc, \tag{5.9}$$

for some matrix K. Smoothness conditions from Theorem 2.1.11 helps us to write the constraint $s\in\mathcal{S}^r_d(\triangle)$ into a linear constraint:

$$Hc \;=\; 0 \tag{5.10}$$

for some matrix H.

Thus, the optimization problem (5.3) can be reformulated into the following form:

$$\min_c \qquad \|Ac-b\|^2+\lambda c^\tau Kc \tag{5.11}$$

$$\text{s.t.} \qquad Hc=0. \tag{5.12}$$

5.2.2 Our Algorithm

Instead of finding all the entries of the coefficient vector c at once, we solve a collection of smaller problems at each iteration. To state our algorithm formally, we need some notations. For any subset $\delta\subset\triangle$, we set $\text{star}^0(\delta)=\delta$, and for all $k\geq 1$, recursively define

$$\text{star}^k(\delta) \;=\; \cup\{T\in\triangle : T\cap\text{star}^{k-1}(\delta)\neq\emptyset\}. \tag{5.13}$$

Fig. 5.1 is an illustration of $\text{star}^1(T)$, where T is the red triangle. $\text{star}^1(T)$ consists of both the red triangle and the green triangles. Fig. 5.2 is $\text{star}^2(T)$.

The algorithm is as follows.

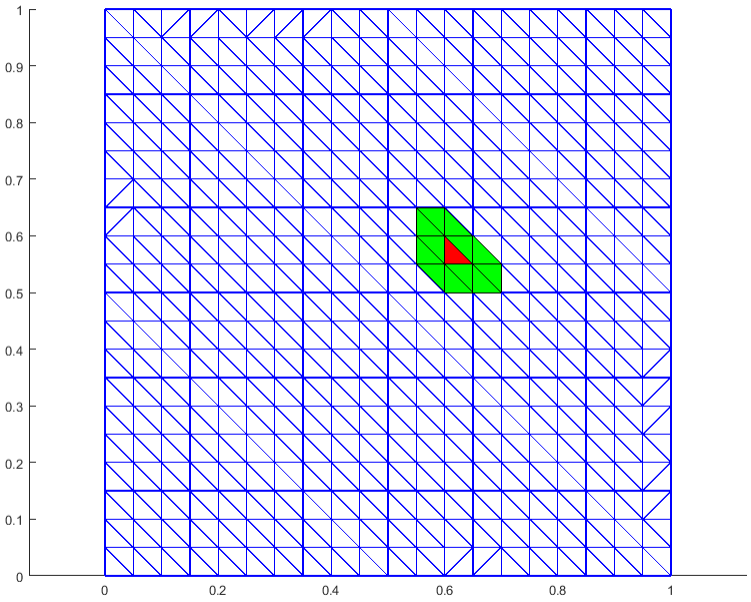

Figure 5.1: $\mathrm{star}^1(T)$.

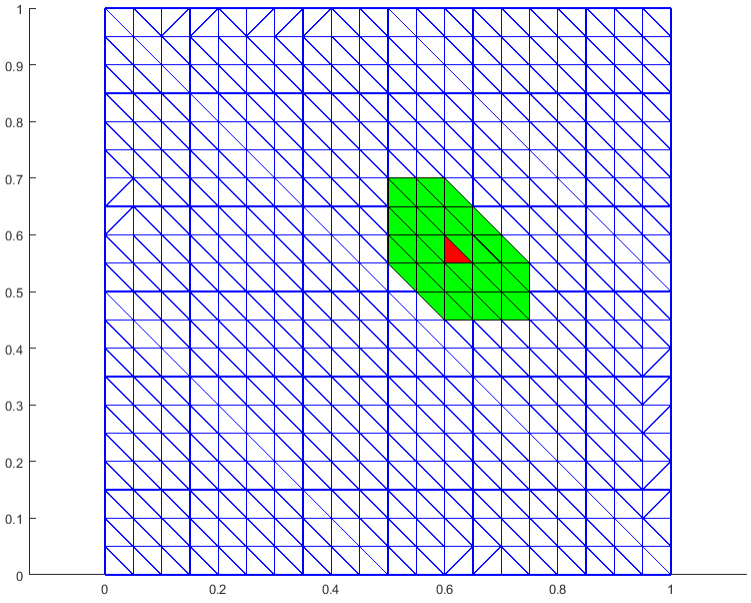

Figure 5.2: $\mathrm{star}^2(T)$.

Algorithm 5.2.1. *(Randomized Domain Decomposition Method (2D))*

1. *Fix a triangulation $\triangle$ of Ω. Choose $k > 0$. Initialize the coefficient vector c (e.g. $c := \mathbf{0}$).*

2. *Randomly select a triangle $T \in \triangle$. Find $\Omega_T^k := \text{star}^k(T)$.*

3. *Let $s_T^k \in \mathcal{S}_d^r(\triangle)|_{\Omega_T^k}$ be the spline fit based on the data in Ω_T^k such that after updating $c_{i,j,k}^t = s_{i,j,k}^t$ for all $i+j+k = d$ and $t \in \Omega_T^k$, the resulting spline satisfies the required smoothness condition.*

4. *If certain stopping criterion is met, quit; otherwise, go back to step 2.*

Let's take a look at how to find the local fit s_T^k in Step 3 above.

Write the coefficient vector c as $[c_1^\tau, c_2^\tau]^\tau$, where c_1 is the coefficient vector associated with the triangles in Ω_T^k, and c_2 is the rest. Write the smoothness matrix H in (5.12) as $[H_1, H_2]$ such that

$$Hc \;=\; Hc_1 + Hc_2. \tag{5.14}$$

Let A_1, K_1 and b_1 be the submatrices of A, K and b related to the triangles in Ω_T^k respectively.

Now finding the local fit s_T^k is equivalent to solving the following minimization problem

$$\min_{c_1} \quad \|A_1c_1 - b_1\|^2 + \lambda c_1^\tau K_1 c_1 \tag{5.15}$$

$$\text{s.t.} \quad H_1c_1 = -H_2c_2. \tag{5.16}$$

Note that the right-hand side of the constraint (5.16) is constant since c_2 is fixed here.

There are many methods that one can use to solve the problem (5.15-5.16). Lagrange multiplier is one of the popular ones. Letting

$$\mathcal{L}(c_1, \xi) \;=\; \|A_1c_1 - b_1\|^2 + \lambda c_1^\tau K_1 c_1 + \xi^\tau(H_1c_1 + H_2c_2), \tag{5.17}$$

there exists ξ such that

$$2(A_1^\tau A_1 + \lambda K_1)c_1 + H_1^\tau \xi \;=\; 2A_1^\tau b_1, \tag{5.18}$$

$$H_1c_1 \;=\; -H_2c_2. \tag{5.19}$$

In general, the linear system above is not invertible. So we need to solve it using the method of least squares. MATLAB provides a class of efficient algorithms for least squares. Alternatively, the iterative method described in [1] can be used. Assuming the existence of an optimal solution, any least-squares solution would satisfy the above linear system exactly and is an optimal solution to (5.15-5.16).

5.2.3 Convergence Analysis

In this section, we analyze the convergence of Algorithm 5.2.1.

First of all, the existence and uniqueness of the minimization problem (5.3) is guaranteed by the following theorem described in [1].

Theorem 5.2.2. *Suppose that there exist three data sites, say* $(x_i, y_i), i = 1, 2, 3$*, which are not collinear. Then there exists a unique* s^* *in* $\mathcal{S}_d^r(\triangle)$ *solving the minimization problem (5.3).*

Let $W_\infty^m(\Omega)$ be the Sobolev space of all functions whose m-th derivatives are essentially bounded over Ω. $|f|_{m,\infty,\Omega}$ is the maximal norm of all m-th order derivatives of f over Ω. $|\triangle|$ is the longest edge length of the triangles in $\triangle$. The following result shows how the surface s^* resembles the given data, and can be found in [25].

Theorem 5.2.3. *Let* s^* *be the minimization solution of (5.3) with* $d \geq 3r + 2$*. Suppose that* $f \in W_\infty^{m+1}(\Omega)$ *with* $1 \leq m \leq d$*. Then there exists a constant* C *such that*

$$\|s^* - f\|_{L_\infty(\Omega)} \quad \leq \quad C(|\triangle|^{m+1}|f|_{m+1,\infty,\Omega} + \lambda|f|_{2,\infty,\Omega}). \tag{5.20}$$

Since (5.15-5.16) is essentially the minimization problem (5.11-5.12) with respect to c_1 with c_2 fixed, it can be easily seen that Algorithm 5.2.1 is a randomized block coordinate descent method. Thus, we can use the results from Chapter 4 to analyze the convergence.

Theorem 5.2.4. *For the optimization problem (5.11-5.12), let*

$$g(c) := \|Ac - b\|^2 + \lambda c^\tau K c \tag{5.21}$$

be the objective function with optimal value g^*. If $\{c^{(n)}\}_{n\geq 0}$ is generated by Algorithm 5.2.1, then we have the following rate of convergence for the expected values of g:

$$\mathbb{E}[g(c^{(n)})] - g^* \quad \leq \quad \frac{M}{n} \tag{5.22}$$

for some constant $M > 0$.

Proof. The only difference between Algorithm 5.2.1 and Algorithm 4.3.3 is that instead of using (4.9), Algorithm 5.2.1 computes the minimum of the left-hand side of (4.6) directly by the method of least squares mentioned in the last subsection. However, this little difference would make no difference since the inequality (4.25) still holds. Thus, the proof of Theorem 4.5.3 passes through. Now this theorem follows. □

5.2.4 Numerical Examples

Example 5.2.5. *This is a very basic example. Consider* 4867 *points uniformly distributed over a convex polygon as shown in Fig. 5.3 with the red dots representing the data points. There are* 167 *triangles in the triangulation $\triangle$. Let $\{(x_i, y_i, f(x_i, y_i)), i = 1, \ldots, 4867\}$ be a scattered data set, where*

$$f(x, y) \quad = \quad x^4 + 2y^3 - xy + 2y - 1 \tag{5.23}$$

is a polynomial. We use Algorithm 5.2.1 to find a spline function $s \in \mathcal{S}_d^r(\triangle)$ to fit the data. In the algorithm, we choose $k = 2$. The maximum errors are measured on 31242 *points uniformly distributed over the domain. The results are summarized in Table 5.1. As can been seen, the best result is splines of degree* 4*, the same degree as f.*

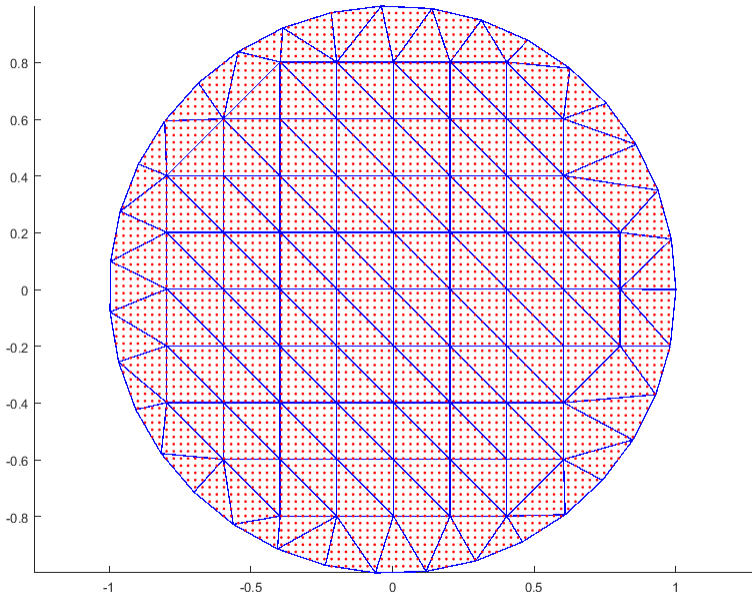

Figure 5.3: Data points for Example 5.2.5.

Table 5.1: Approximation errors in Example 5.2.5

	Errors	**CPU**		**Errors**	**CPU**
$\mathcal{S}_3^0$	1.05e-04	1.18s	$\mathcal{S}_3^1$	3.11e-02	20.87s
$\mathcal{S}_4^0$	7.17e-05	1.41s	$\mathcal{S}_4^1$	9.93e-06	23.01s
$\mathcal{S}_5^0$	9.32e-04	1.95s	$\mathcal{S}_5^1$	2.08e-05	4.27s

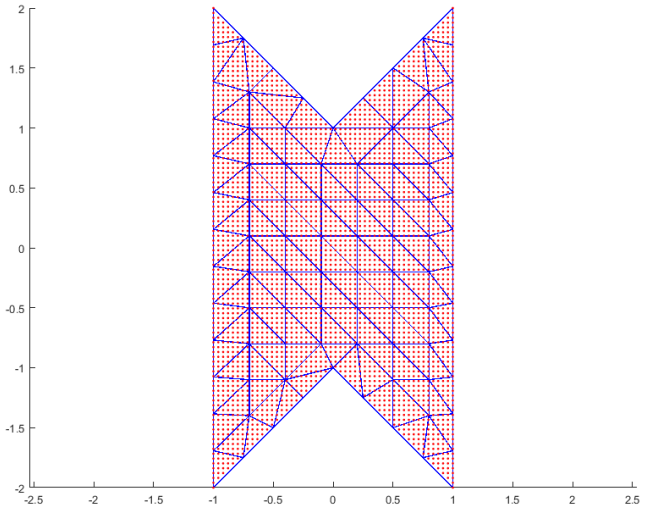

Figure 5.4: Data points for Example 5.2.6.

Table 5.2: Approximation errors in Example 5.2.6

	Errors	**CPU**		**Errors**	**CPU**
$\mathcal{S}_4^0$	2.10e-01	1.19s	$\mathcal{S}_4^1$	1.45e-01	8.97s
$\mathcal{S}_5^0$	2.99e-02	1.55s	$\mathcal{S}_5^1$	3.15e-02	4.90s
$\mathcal{S}_6^0$	4.81e-02	3.02s	$\mathcal{S}_6^1$	1.91e-02	5.52s

Example 5.2.6. *Consider* 3704 *points uniformly distributed over a non-convex polygon as shown in Fig. 5.4 with the red dots representing the data points. There are* 136 *triangles. Let* $\{(x_i, y_i, f(x_i, y_i)), i = 1, \ldots, 3704\}$ *be a scattered data set, where*

$$f(x, y) \quad = \quad \sin(\pi(x^2 + 2y^2)). \tag{5.24}$$

Again, we choose $k = 2$ *in the algorithm. The maximum errors are measured on* 40401 *points uniformly distributed over the domain. The results are summarized in Table 5.2.*

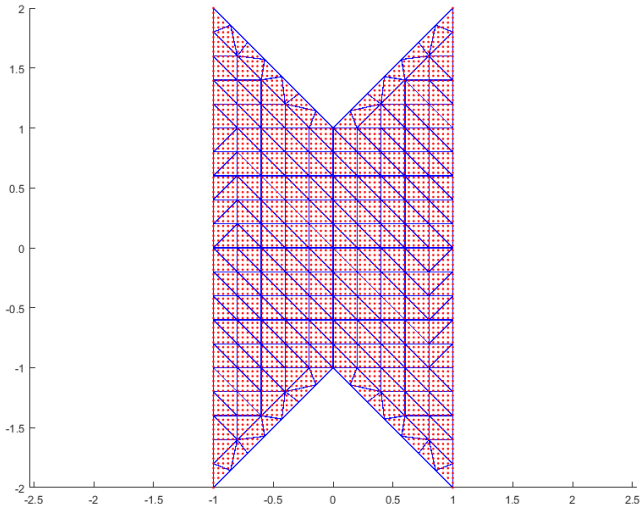

Figure 5.5: Data points for Example 5.2.7.

Table 5.3: Approximation errors in Example 5.2.7

	Errors	**CPU**		**Errors**	**CPU**
$\mathcal{S}_4^0$	1.20e-02	3.46s	$\mathcal{S}_4^1$	2.35e-02	49.48s
$\mathcal{S}_5^0$	7.50e-03	40.11s	$\mathcal{S}_5^1$	4.61e-03	9.75s
$\mathcal{S}_6^0$	3.03e-02	5.29s	$\mathcal{S}_6^1$	8.11e-04	98.83s

Example 5.2.7. *We use the same data set as in Example 5.2.6. But the triangulation is more refined as shown in Fig. 5.5. We now have* 308 *triangles. The results summarized in Table 5.3 are better as expected.*

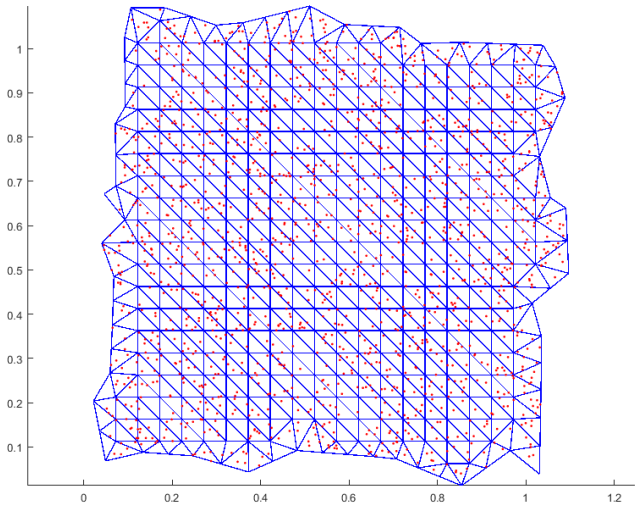

Figure 5.6: Data points for Example 5.2.8.

Table 5.4: Approximation errors in Example 5.2.8

	Errors	**CPU**		**Errors**	**CPU**
$\mathcal{S}_4^0$	5.53e+00	7.15s	$\mathcal{S}_4^1$	5.18e-02	185.27s
$\mathcal{S}_5^0$	19.79e+00	12.33s	$\mathcal{S}_5^1$	6.18e-02	19.83s
$\mathcal{S}_6^0$	7.94e+00	22.49s	$\mathcal{S}_6^1$	7.25e-02	26.65s

Example 5.2.8. *Consider* 1681 *points randomly chosen over a non-convex polygon as shown in Fig. 5.6 with the red dots representing the data points. There are* 758 *triangles. Let* $\{(x_i, y_i, f(x_i, y_i)), i = 1, \ldots, 1681\}$ *be a scattered data set, where*

$$f(x, y) \quad = \quad \sin(\pi(x^2 + 2y^2)) + 2e^{-(x-0.5)^2-(y-0.3)^2}. \tag{5.25}$$

We choose $k = 4$ *in the algorithm. The maximum errors are measured on* 9701 *points uniformly distributed over the domain. The results are summarized in Table 5.4.*

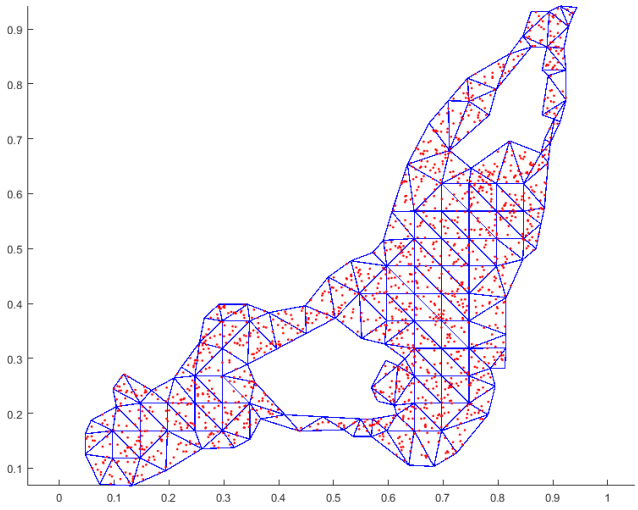

Figure 5.7: Data points for Example 5.2.9.

Table 5.5: Approximation errors in Example 5.2.9

	Errors	**CPU**		**Errors**	**CPU**
$\mathcal{S}_4^0$	2.33e-02	0.86s	$\mathcal{S}_4^1$	7.91e-03	2.41s
$\mathcal{S}_5^0$	2.80e-02	1.15s	$\mathcal{S}_5^1$	8.30e-03	3.36s
$\mathcal{S}_6^0$	2.95e-02	2.24s	$\mathcal{S}_6^1$	8.80e-03	3.95s

Example 5.2.9. *Consider* 1583 *points randomly chosen over a domain with two holes as shown in Fig. 5.7 where the red dots represent the data points. There are* 212 *triangles. Let* $\{(x_i, y_i, f(x_i, y_i)), i = 1, \ldots, 1583\}$ *be a scattered data set, where*

$$f(x, y) \quad = \quad \sin(\pi(x^2 + 2y^2)) + x^3 - 2x^4y^2. \tag{5.26}$$

We choose $k = 3$ *in the algorithm. The maximum errors are measured on* 2396 *points uniformly distributed over the domain. The results are summarized in Table 5.5.*

5.3 Three-Dimensional Algorithm

The three-dimensional randomized domain decomposition method for trivariate scattered data fitting is essentially the same as the two-dimensional one. We briefly present the algorithm and illustrate its effectiveness by several examples.

5.3.1 Problem Formulation

Let $\Omega \subseteq \mathbb{R}^3$ be a polyhedral domain. Suppose we are given values $\{f_i := f(x_i, y_i, z_i)\}_{i=1}^N$ at points in $\mathcal{A} := \{v_i = (x_i, y_i, z_i)\}_{i=1}^N \subset \Omega$. We wish to compute a smooth function s that approximates the data. To be more specific, given $0 \le r < d$ and a tetrahedralization $\triangle$ of Ω, let

$$\mathcal{S}_d^r(\triangle) \;:=\; \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{5.27}$$

be the associated space of trivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of trivariate polynomials of degree at most d, as defined in Section 2.2. Also fix a constant $\lambda > 0$. The corresponding penalized least-squares (PLS) spline (cf. [1]) is defined to be:

$$s^* \;:=\; \arg\min_{s \in \mathcal{S}_d^r(\triangle)} \|s - f\|_{\mathcal{A}}^2 + \lambda E(s), \tag{5.28}$$

where

$$\|s - f\|_{\mathcal{A}}^2 \;:=\; \sum_{i=1}^N (s(x_i, y_i, z_i) - f_i)^2, \tag{5.29}$$

and

$$E(s) \;:=\; \int_\Omega \sum_{\substack{\alpha,\beta,\gamma \ge 0 \\ \alpha+\beta+\gamma=2}} \left(\frac{\partial^2 s}{\partial x^\alpha \partial y^\beta \partial z^\gamma} \right)^2 dxdydz. \tag{5.30}$$

Since $s|_T$ is a polynomial of degree d on each tetrahedron $T \in \triangle$, we can use the B-form representation (2.58):

$$s|_T \;=\; \sum_{i+j+k+l=d} c_{ijkl}^T B_{ijkl}^d. \tag{5.31}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c \;:=\; [c^T_{i,j,k,l}, i+j+k+l=d, T\in\triangle], \tag{5.32}$$

where c is a column vector of dimension $m\binom{d+3}{3}$ with m denoting the number of tetrahedrons in $\triangle$. Moreover, using the B-form representation (2.58), we obtain

$$\|s-f\|^2_{\mathcal{A}} \;=\; \|Ac-b\|^2 \tag{5.33}$$

for some matrix A and $b=[f_1,\ldots,f_N]^\tau$. We also have

$$E(s) \;=\; c^\tau Kc, \tag{5.34}$$

for some matrix K. Smoothness conditions from Theorem 2.2.12 helps us to write the constraint $s\in\mathcal{S}^r_d(\triangle)$ into a linear constraint:

$$Hc \;=\; 0 \tag{5.35}$$

for some matrix H.

Thus, the optimization problem (5.28) can be reformulated into the following form:

$$\min_c \qquad \|Ac-b\|^2+\lambda c^\tau Kc \tag{5.36}$$

$$\text{s.t.} \qquad Hc=0. \tag{5.37}$$

5.3.2 Our Algorithm

Instead of finding all the entries of the coefficient vector c at once, we solve a collection of smaller problems at each iteration.

For any subset $\delta\subset\triangle$, we set $\text{star}^0(\delta)=\delta$, and for all $k\geq 1$, recursively define

$$\text{star}^k(\delta) \;=\; \cup\{T\in\triangle : T\cap\text{star}^{k-1}(\delta)\neq\emptyset\}. \tag{5.38}$$

The algorithm is as follows.

Algorithm 5.3.1. *(Randomized Domain Decomposition Method (3D))*

1. *Fix a tetrahedralization $\triangle$ of Ω. Choose $k > 0$. Initialize the coefficient vector c (e.g. $c := \mathbf{0}$).*

2. *Randomly select a tetrahedron $T \in \triangle$. Find $\Omega_T^k := \text{star}^k(T)$.*

3. *Let $s_T^k \in \mathcal{S}_d^r(\triangle)|_{\Omega_T^k}$ be the spline fit based on the data in Ω_T^k such that after updating $c_{i,j,k,l}^t = s_{i,j,k,l}^t$ for all $i+j+k+l = d$ and $t \in \Omega_T^k$, the resulting spline satisfies the required smoothness condition.*

4. *If certain stopping criterion is met, quit; otherwise, go back to step 2.*

The local fit s_T^k in Step 3 above can be computed in the same way as in the two-dimensional case.

Write the coefficient vector c as $[c_1^\tau, c_2^\tau]^\tau$, where c_1 is the coefficient vector associated with the tetrahedrons in Ω_T^k, and c_2 is the rest. Write the smoothness matrix H in (5.37) as $[H_1, H_2]$ such that

$$Hc \;=\; Hc_1 + Hc_2. \tag{5.39}$$

Let A_1, K_1 and b_1 be the submatrices of A, K and b related to the tetrahedrons in Ω_T^k respectively.

Now finding the local fit s_T^k is equivalent to solving the following minimization problem

$$\min_{c_1} \qquad \|A_1c_1 - b_1\|^2 + \lambda c_1^\tau K_1 c_1 \tag{5.40}$$

$$\text{s.t.} \qquad H_1c_1 = -H_2c_2. \tag{5.41}$$

5.3.3 Convergence Analysis

The convergence of Algorithm 5.3.1 is essentially the same as that of Algorithm 5.2.1. So we give the following theorem without proof.

Theorem 5.3.2. *For the optimization problem (5.36-5.37), let*

$$g(c) := \|Ac - b\|^2 + \lambda c^\tau Kc \tag{5.42}$$

Table 5.6: Approximation errors in Example 5.3.3

	Errors	**CPU**
$\mathcal{S}_3^1$	1.36e+00	940.29s
$\mathcal{S}_4^1$	8.87e-06	1594.51s
$\mathcal{S}_5^1$	3.91e-06	1771.87s

be the objective function with optimal value g^*. If $\{c^{(n)}\}_{n\geq 0}$ is generated by Algorithm 5.3.1, then we have the following rate of convergence for the expected values of g:

$$\mathbb{E}[g(c^{(n)})] - g^* \quad \leq \quad \frac{M}{n} \tag{5.43}$$

for some constant $M > 0$.

5.3.4 Numerical Examples

Example 5.3.3. *Consider* 8000 *points uniformly distributed over a cube as shown in Fig. 5.8 with the red dots representing the data points. There are* 384 *tetrahedrons in the tetrahedralization $\triangle$. Let $\{(x_i, y_i, z_i, f(x_i, y_i, z_i)), i = 1, \ldots, 8000\}$ be a scattered data set, where*

$$f(x, y, z) \quad = \quad x^4 + 2y^3 + xyz - z^2 + 3xz - yz + 2x - 1 \tag{5.44}$$

is a polynomial. We use Algorithm 5.3.1 to find a spline function $s \in \mathcal{S}_d^r(\triangle)$ to fit the data. In the algorithm, we choose $k = 2$. The maximum errors are measured on 125000 *points uniformly distributed over the domain. The results are summarized in Table 5.6.*

Example 5.3.4. *Consider* 20700 *points uniformly distributed over a non-convex domain as shown in Fig. 5.9 with the red dots representing the data points. There are* 619 *tetrahedrons. Let $\{(x_i, y_i, z_i, f(x_i, y_i, z_i)), i = 1, \ldots, 20700\}$ be a scattered data set, where*

$$f(x, y, z) \quad = \quad \sin(\pi(x^2 + y^2 + z^2)) + xy - z. \tag{5.45}$$

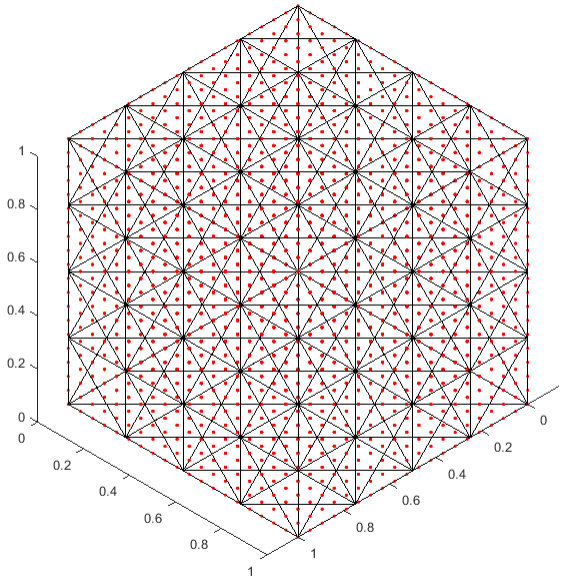

Figure 5.8: Data points for Example 5.3.3.

We choose $k = 2$ *in the algorithm. The maximum errors are measured on* 95000 *points uniformly distributed over the domain. The results are summarized in Table 5.7.*

Example 5.3.5. *Consider* 24952 *points uniformly distributed over a torus-shape domain as shown in Fig. 5.10 with the yellow dots representing the data points. There are* 588 *tetrahedrons. Let* $\{(x_i, y_i, z_i, f(x_i, y_i, z_i)), i = 1, \ldots, 24952\}$ *be a scattered data set, where*

$$f(x, y, z) \quad = \quad 2e^{-(x-1.5)^2-y^2-z^2} + x^2 - yz. \tag{5.46}$$

We choose $k = 2$ *in the algorithm. The maximum errors are measured on* 49296 *points uniformly distributed over the domain. The results are summarized in Table 5.8.*

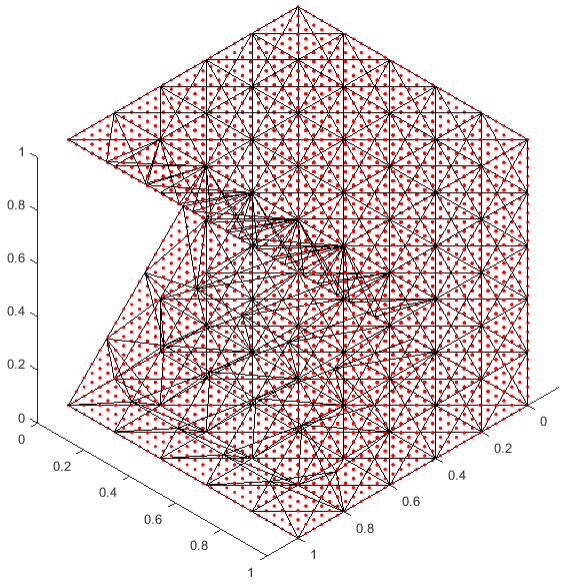

Figure 5.9: Data points for Example 5.3.4.

Table 5.7: Approximation errors in Example 5.3.4

	Errors	**CPU**
$\mathcal{S}_3^1$	2.51e+00	901.58s
$\mathcal{S}_4^1$	4.27e-02	1867.81s
$\mathcal{S}_5^1$	2.40e-03	2829.66s

Table 5.8: Approximation errors in Example 5.3.5

	Errors	**CPU**
$\mathcal{S}_3^1$	1.19e-01	713.08s
$\mathcal{S}_4^1$	6.58e-04	949.69s
$\mathcal{S}_5^1$	5.60e-04	1085.77s

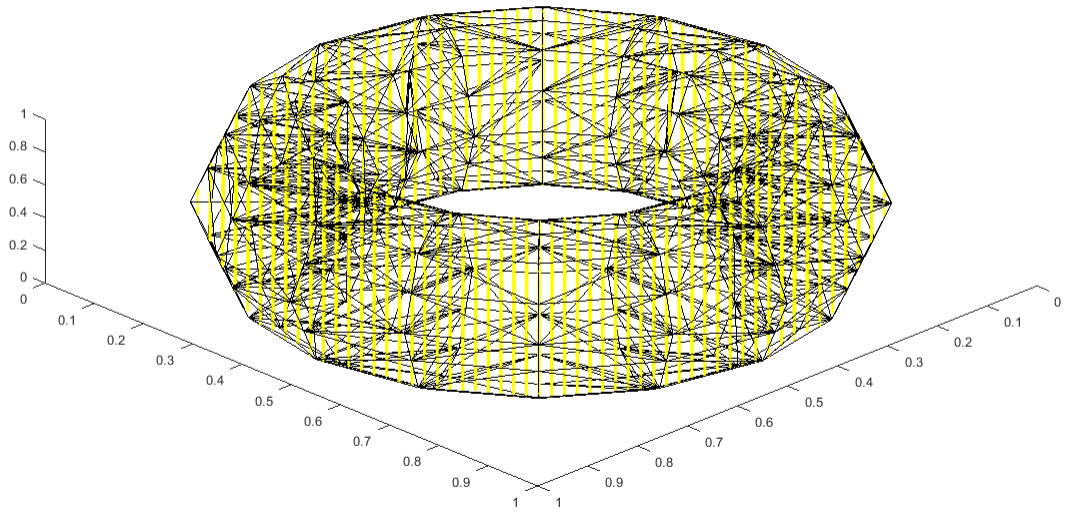

Figure 5.10: Data points for Example 5.3.5.

5.4 Remarks and Future Work

Since the data fitting problems discussed in this chapter have been formulated into optimization problems, and the energy terms (5.5) and (5.30) are included, our algorithm can handle sparse data set. In fact, this can be seen from Examples 5.2.8 and 5.2.9 where some triangles contain very few data points.

The goal of data fitting is to find a good approximation of the given data. With that in mind, we can mix in some empirical ways to choose blocks. For example, for every odd iteration, we randomly pick a triangle/tetrahedron T and update the spline on $\text{star}^k(T)$, while for every even iteration, we pick T which has the largest fitting error. This alternating method will not undermine the convergence since it is still a descent method.

One possible future work would be to use parallel or distributed computing to accelerate the computation. Extra care is needed due to the coupled constraints and overlapping blocks.

Chapter 6

Multivariate Splines for Curve and Surface Reconstruction

6.1 Introduction

In this chapter, we construct a smooth interpolatory or fitting curve of a given point set in the 2D setting, and a smooth surface in the 3D setting. This will have some important applications in real life. For example, when designing airplanes, one would like to connect the wings to the plane body in C^2 fashion. The smoothness can help reduce air friction and hence increase fuel efficiency.

There have been a lot of theories and methods for computing smooth curves and surfaces. One typical method is to use tensor products of B-splines. Some aerospace and car companies use the nonuniform rational B-splines (NURB) (cf. [15]). The concepts of G^1 continuity instead of C^1 continuity for connecting smooth surfaces was introduced and studied (cf. [33]). Another popular method is subdivision schemes (cf. [13]). However, these tools are not flexible enough to connect two surface patches together in a smooth fashion.

In this chapter, we use multivariate spline functions to construct curves and surfaces with smoothness C^r, where $r \geq 1$. The intuition is that any smooth enough curve can be viewed as a contour or a part of a contour of a smooth spline function defined on a polygonal domain, and any smooth enough surface can be treated as an isosurface of a smooth spline function defined on a polyhedral domain.

6.2 Construction of 2D Smooth Curves

In this section we describe the algorithm to generate smooth curves in the 2D setting.

6.2.1 Our Algorithm

Suppose we are given points $\mathcal{A}_0 := \{(x_i, y_i)\}_{i=1}^{N_0}$ from some unknown planar closed smooth curve C. We first construct a polygonal domain $\Omega \subseteq \mathbb{R}^2$ which contains all the given points in its interior. For simplicity, we can just use a rectangular bounding box. Then we find a triangulation $\triangle$ of Ω. If there are some known connectivity relationships of the data points, e.g., piecewise linear interpolations, we can adopt a constrained triangulation to include those line segments.

We wish to compute a smooth function s whose contour at value 0 approximates the desired curve. In order for the curve to stand out, we choose auxiliary points in the following sense. Pick points $\mathcal{A}_{-1} := \{(x_i, y_i)\}_{i=1}^{N_{-1}} \subset \Omega$ outside the closed curve, and assign them function value -1. Pick points $\mathcal{A}_1 := \{(x_i, y_i)\}_{i=1}^{N_1} \subset \Omega$ inside the closed curve, and assign them function value 1. There are no absolute rules of how to choose these auxiliary points. According to our experiments, there are two ways that usually work well. The first way is to pick $\mathcal{A}_{-1}$ near the boundary of Ω and $\mathcal{A}_1$ near the center of the curve. Another way is to pick points on both sides of the curve, maintaining nearly equal distances of $\mathcal{A}_{-1}$ and $\mathcal{A}_1$ from the curve. Fig. 6.1 and 6.2 illustrate these two ways, where the red points are on the curve, the blue ones are inside the curve, and the black ones are outside the curve.

The next task is to do the data fitting as discussed in Chapter 5. Given $0 \leq r < d$ and a triangulation $\triangle$, let

$$\mathcal{S}_d^r(\triangle) \;:=\; \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{6.1}$$

be the associated space of bivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of bivariate polynomials of degree at most d, as defined in Section 2.1. Also fix a constant $\lambda > 0$. The corresponding penalized least-squares (PLS) spline (cf. [1]) is defined

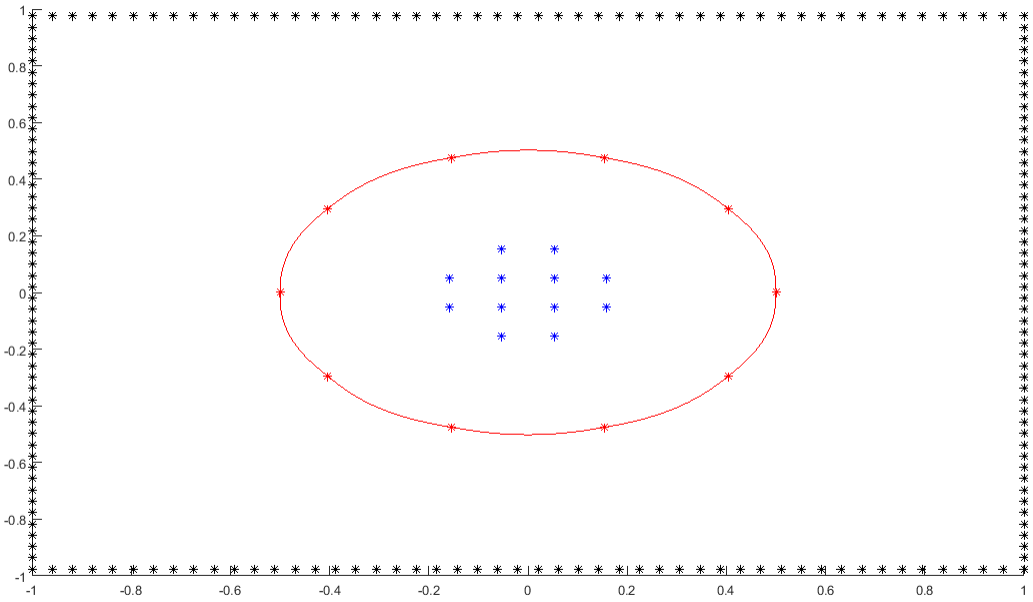

Figure 6.1: One way to choose points for $\mathcal{A}_{-1}$, $\mathcal{A}_0$ and $\mathcal{A}_1$.

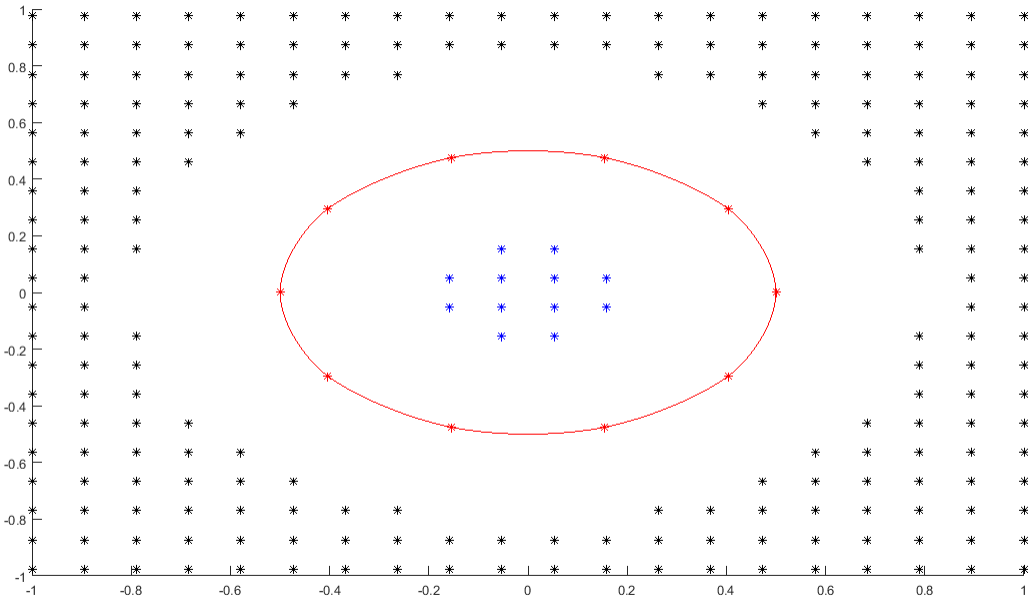

Figure 6.2: Another way to choose points for $\mathcal{A}_{-1}$, $\mathcal{A}_0$ and $\mathcal{A}_1$.

to be:

$$s^* \;:=\; \arg\min_{s\in\mathcal{S}_d^r(\triangle)} \|s-(-1)\|_{\mathcal{A}_{-1}}^2 + \|s-0\|_{\mathcal{A}_0}^2 + \|s-1\|_{\mathcal{A}_1}^2 + \lambda E(s), \tag{6.2}$$

where

$$\|s-t\|_{\mathcal{A}}^2 \;:=\; \sum_{(x_i,y_i)\in\mathcal{A}} (s(x_i,y_i)-t)^2, \tag{6.3}$$

and

$$E(s) \;:=\; \int_\Omega \left[\left(\frac{\partial^2 s}{\partial x^2}\right)^2 + 2\left(\frac{\partial^2 s}{\partial x\partial y}\right)^2 + \left(\frac{\partial^2 s}{\partial y^2}\right)^2\right] dxdy. \tag{6.4}$$

Since $s|_T$ is a polynomial of degree d on each triangle $T \in \triangle$, we can use the B-form representation (2.15):

$$s|_T \;=\; \sum_{i+j+k=d} c_{ijk}^T B_{ijk}^d. \tag{6.5}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c \;:=\; [c_{i,j,k}^T, i+j+k=d, T\in\triangle], \tag{6.6}$$

where c is a column vector of dimension $m\binom{d+2}{2}$ with m denoting the number of triangles in $\triangle$. Moreover, using the B-form representation (2.15), we obtain

$$\|s-(-1)\|_{\mathcal{A}_{-1}}^2 + \|s-0\|_{\mathcal{A}_0}^2 + \|s-1\|_{\mathcal{A}_1}^2 \;=\; \|Ac-b\|^2 \tag{6.7}$$

for some matrix A and $b = [-1,\ldots,-1,0,\ldots,0,1,\ldots,1]^\tau$. We also have

$$E(s) \;=\; c^\tau K c, \tag{6.8}$$

for some matrix K. Smoothness conditions from Theorem 2.1.11 helps us to write the constraint $s \in \mathcal{S}_d^r(\triangle)$ into a linear constraint:

$$Hc \;=\; 0 \tag{6.9}$$

for some matrix H.

Thus, the optimization problem (6.2) can be reformulated into the following form:

$$\min_{c} \quad \|Ac - b\|^2 + \lambda c^\tau K c \tag{6.10}$$

$$\text{s.t.} \quad Hc = 0. \tag{6.11}$$

In fact, our spline method is so flexible that it supports not only fitting but also interpolation. To be more specific, if we want to make sure that the points in some set $\mathcal{A}_0' := \{(x_i, y_i)\}_{i=1}^{N_0'}$ are on the resulting curve, we can express them as a linear equation $Bc = 0$ and put the equation into the constraint. This gives us the following more general form:

$$\min_{c} \quad \|Ac - b\|^2 + \lambda c^\tau K c \tag{6.12}$$

$$\text{s.t.} \quad Hc = 0, \tag{6.13}$$

$$Bc = 0. \tag{6.14}$$

It is recommended that the triangulation $\triangle$ be manipulated so that the points in $\mathcal{A}_0'$ are at the vertices of $\triangle$.

If the scale of the problem is very large, then instead of finding all the entries of the coefficient vector c at once, we can apply the randomized domain decomposition method presented in Chapter 5 with a little modification to handle (6.14).

6.2.2 Finding Contours

After finding the spline function, we need to figure out how to get the contour curve of value 0. We need an algorithm that is capable of computing points on the contour. Moreover, the density of these points can be set arbitrarily. Note that it is very important to be able to get as many points on the contour as wanted, especially in applications like computer graphics.

First we need to assume that the points in the set $\mathcal{A}_0'$ which are guaranteed to be on the contour by the constraint (6.14) are dense enough such that for any two adjacent points $u, v \in \mathcal{A}_0'$, the straight line segment connecting u and v is close enough to the contour. We will see later what we mean by "close enough".

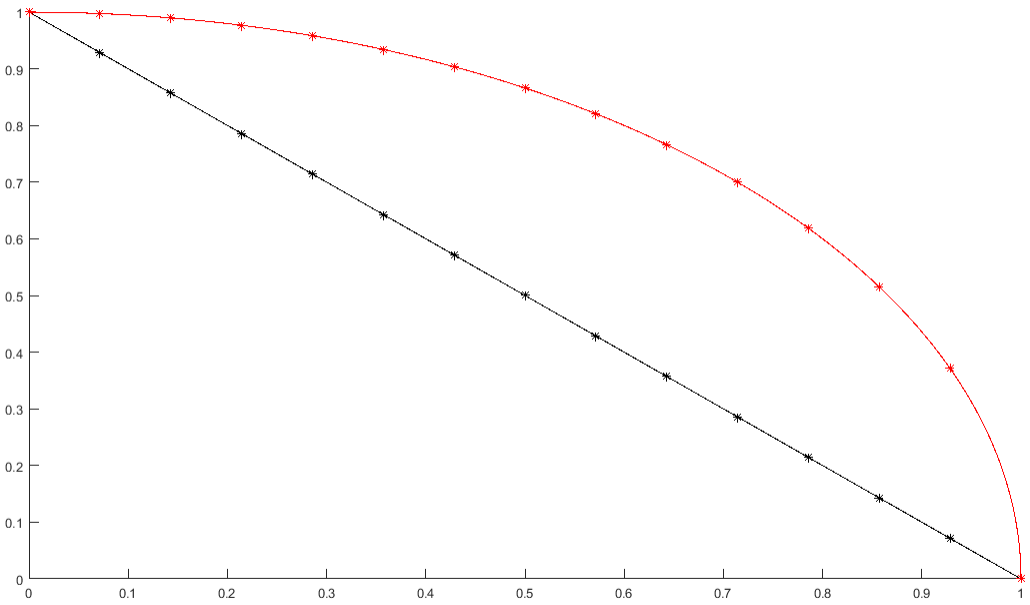

Figure 6.3: Illustration of computing contours.

Under this assumption, on each line segment connecting adjacent u and v, pick points as needed. For example, pick 20 points uniformly. For every such point, fix one of its coordinate, and use the well-known root-finding Newton's method to produce the other coordinate, using the original coordinate as the initial estimate, such that the resulting point is on the contour.

This process is illustrated in Fig. 6.3. In the picture, the red curve is the contour we want. We already know the two endpoints since they are in $\mathcal{A}'_0$. Take as many points as needed on the connecting line segment. For each point, fix its x-coordinate, and use its y-coordinate as the initial guess to apply Newton's method. The iteration formula is

$$y_{n+1} \quad = \quad y_n - \frac{s(x, y_n)}{s_y(x, y_n)}, \tag{6.15}$$

where $s_y = \frac{\partial s}{\partial y}$. The resulting point would have the same x-coordinate and lie on the contour curve.

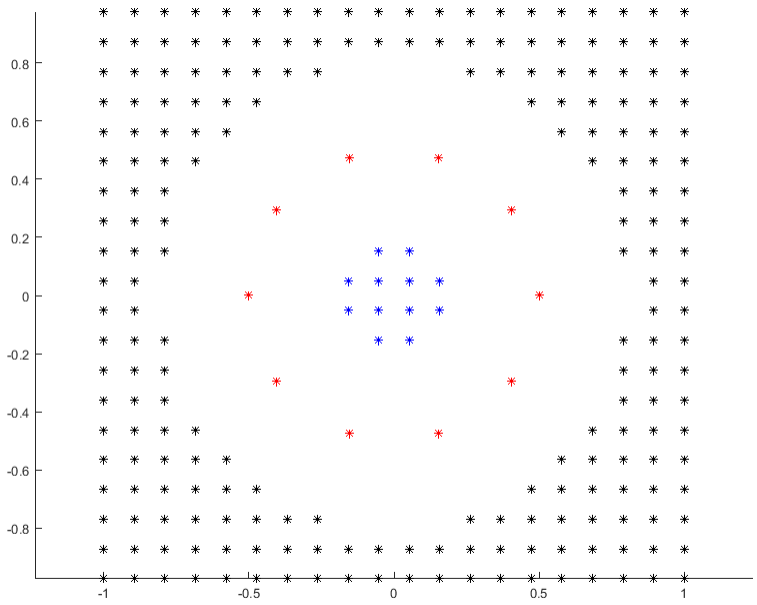

Figure 6.4: Data points for Example 6.2.1.

In order for the Newton's method to converge to the desired root, the initial guess needs to be close enough to the root. This is where the assumption we mentioned earlier comes from.

6.2.3 Examples

Example 6.2.1. *We are given some points sampled from a circle as shown by the red points in Fig. 6.4. These points form a piecewise linear closed curve. We select some other points inside and outside the closed curve as shown by the blue and black dots in the figure. We then find a constrained triangulation* $\triangle$ *where the piecewise linear curve serves as the constraint. See Fig. 6.5. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. The result is shown in Fig. 6.6.*

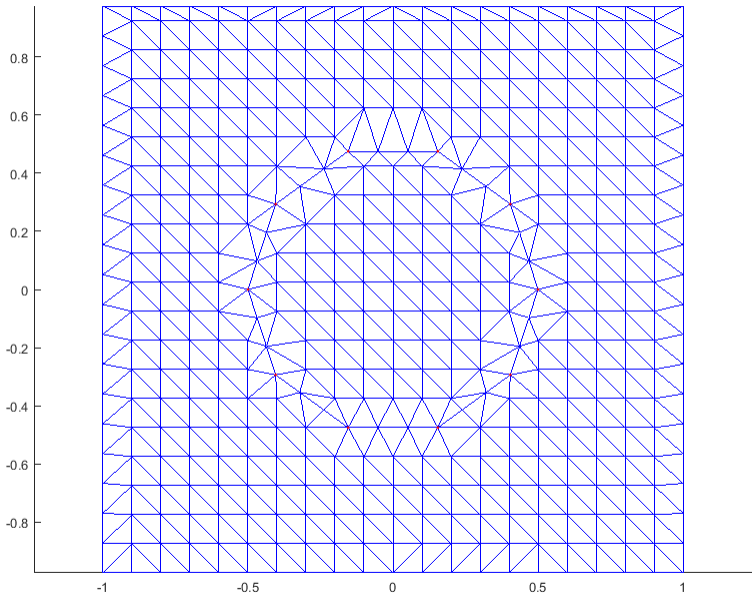

Figure 6.5: The triangulation for Example 6.2.1.

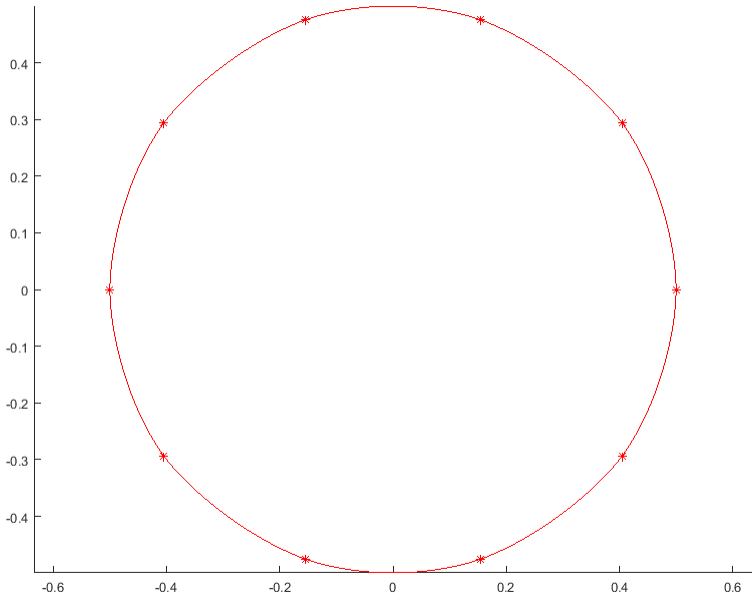

Figure 6.6: The contour curve for Example 6.2.1.

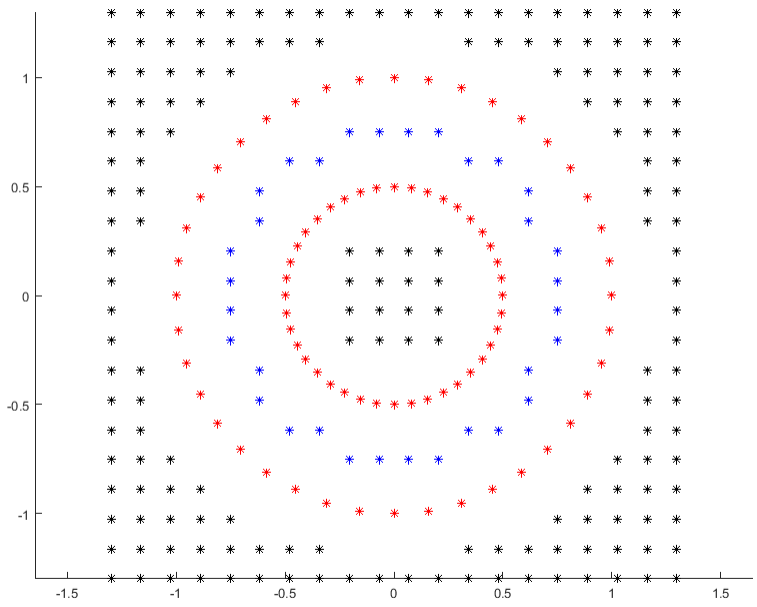

Figure 6.7: Data points for Example 6.2.2.

Example 6.2.2. *In this example, points are sampled from two circles as shown in Fig. 6.7. The constrained triangulation* $\triangle$ *is shown in Fig. 6.8. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. The result is shown in Fig. 6.9.*

Example 6.2.3. *We are given some points on a curve as shown by the red points in Fig. 6.10. The constrained triangulation* $\triangle$ *is shown in Fig. 6.11. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. The result is shown in Fig. 6.12.*

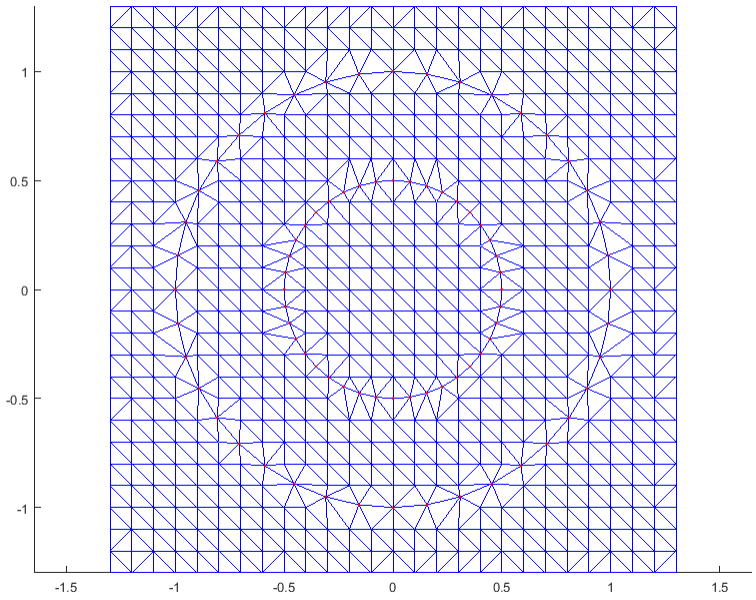

Figure 6.8: The triangulation for Example 6.2.2.

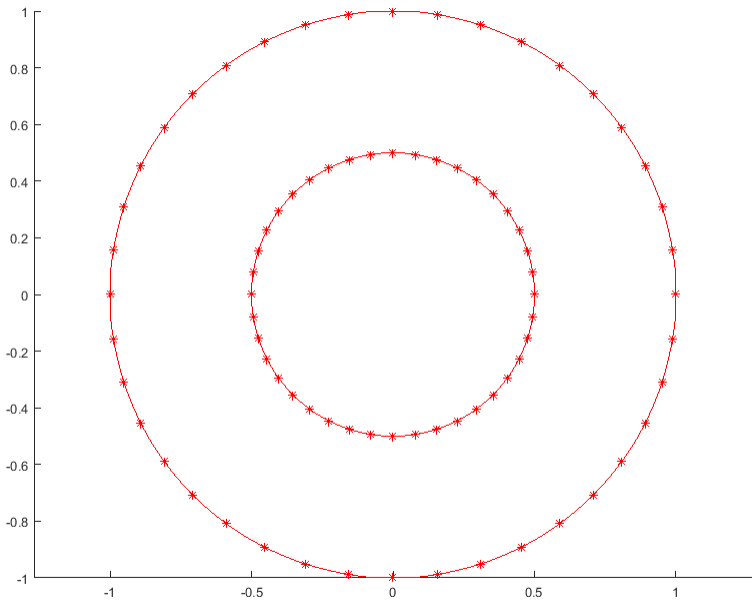

Figure 6.9: The contour curve for Example 6.2.2.

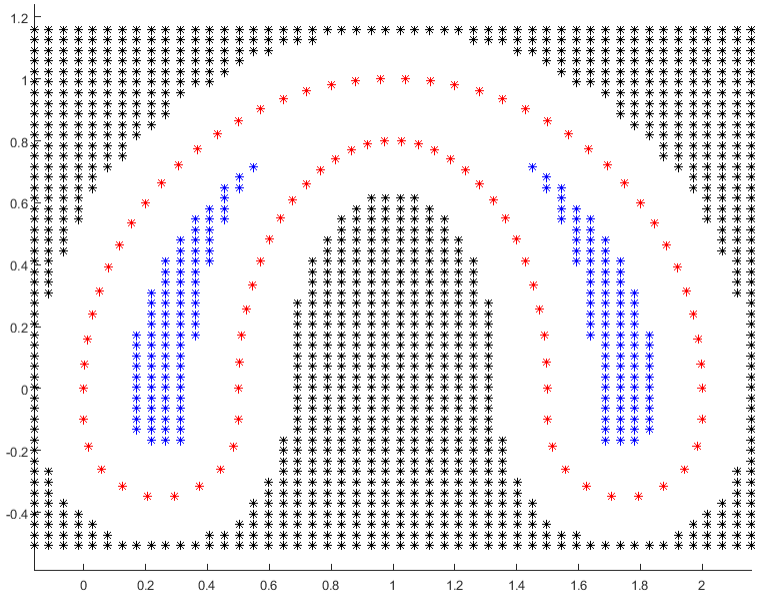

Figure 6.10: Data points for Example 6.2.3.

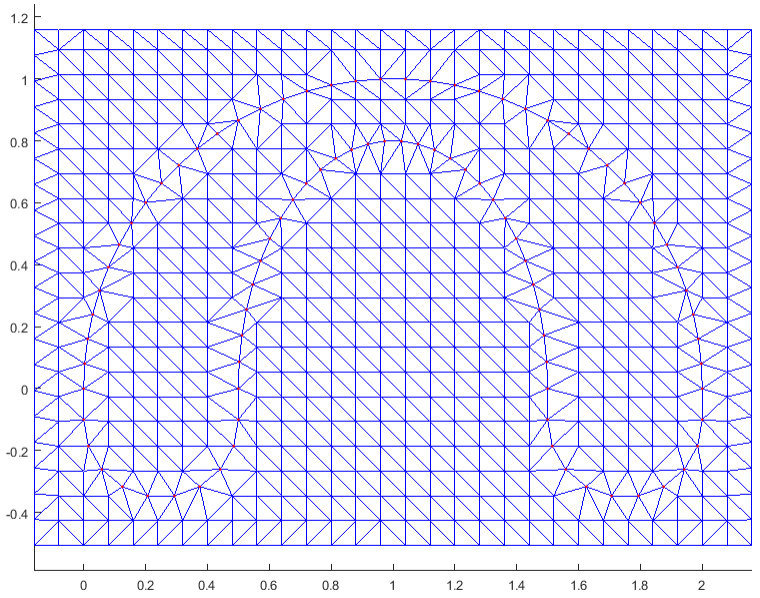

Figure 6.11: The triangulation for Example 6.2.3.

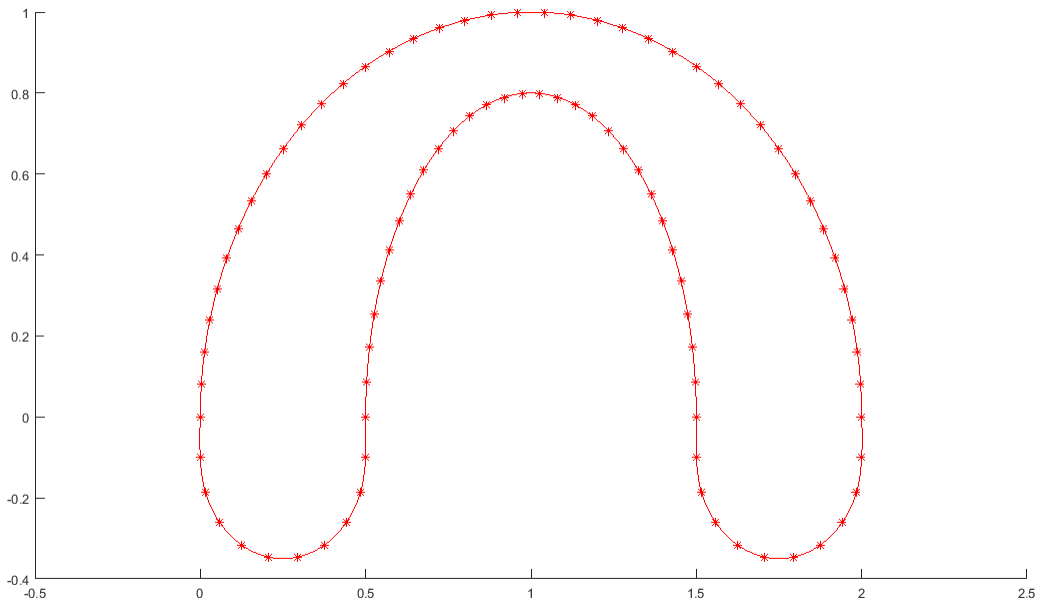

Figure 6.12: The contour curve for Example 6.2.3.

6.2.4 Curves with Sharp Corners

Previously we only talked about smooth curves. In practice, it is inevitable to come across curves with sharp corners at a couple of places while smooth elsewhere. In this section, we present a method to deal with this important situation.

As is known, no C^1 spline function has contours with sharp corners. So naturally, splines which are C^0 near the corner might be a candidate. However, due to the energy term in (6.2), it is very difficult, if not impossible, to enforce a corner, since a corner usually doesn't have the minimum energy.

One might resort to C^{-1} instead. But this would cause technical problems. As is known to all, floating-point arithmetic is used in computers, and it is inevitably susceptible to loss of accuracy . Fig. 6.13 is a simple example to illustrate this problem. Suppose we want to evaluate the spline function at a point p in T_1, and p is very close to one of T_1's edges.

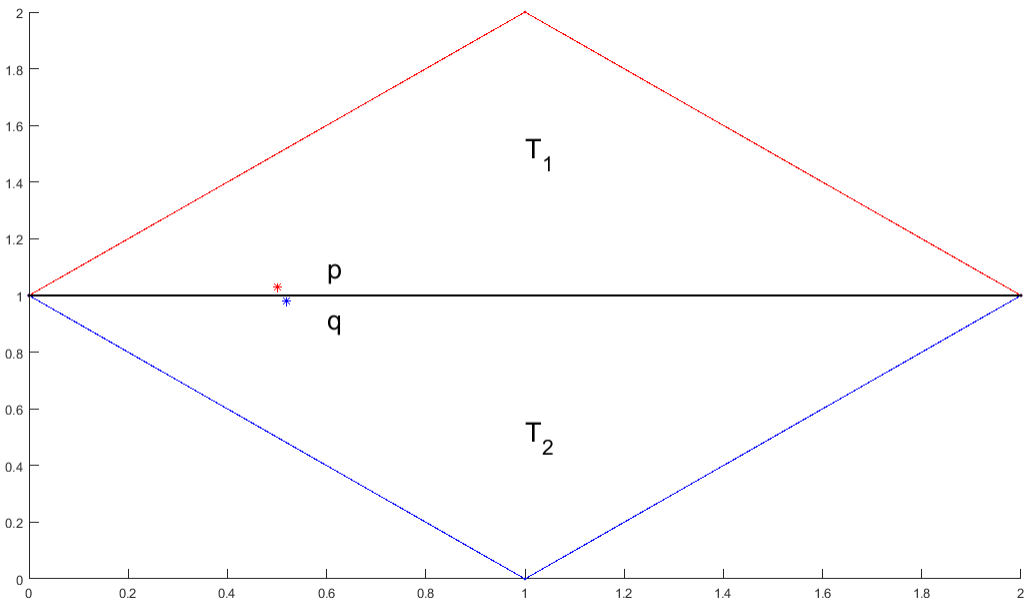

Figure 6.13: A problem caused by C^{-1}.

When the coordinates of p is represented as a floating-point number in computer, inaccuracy of machine precision might actually assign the coordinates of another point q to p. Quite unfortunately, q lies in T_2 which shares a common edge with T_1. Now, if the spline function is C^{-1} on this common edge, the function values at p and q can be quite different. This would cause huge problems. In contrast, if the function is continuous on the common edge, the error would be acceptably small.

The argument above indicates that some other techniques need to be considered to generate a corner which is computationally tractable. And hopefully, the spline function is still smooth to some extent. One inspiration is the contour of the function $f(x, y) = x^2 - y^2$. This function has a saddle point at the origin, and the contours at value 0 form a cross at the origin. See Fig. 6.14.

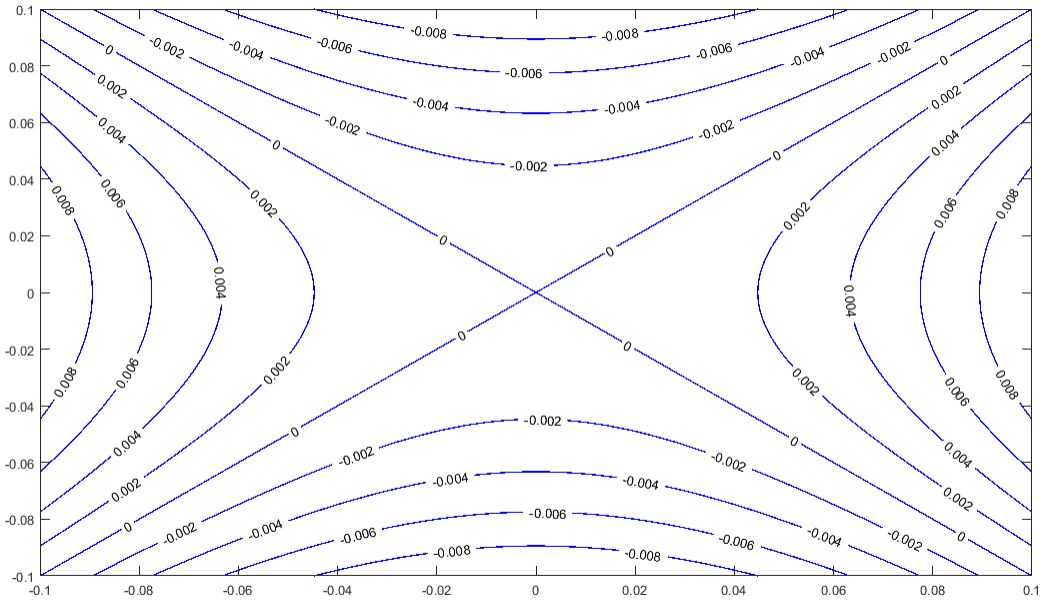

Figure 6.14: The saddle point of $f(x, y) = x^2 - y^2$.

Observe that if we can eliminate half of the cross, we get a sharp corner. So the basic idea is to form a saddle point on the boundary of the domain Ω.

In order to achieve this, we need to assume that the tip of the corner is formed by two short straight line segments. Note that this is a pretty mild assumption.

On the other side of the corner, we remove a hole from the domain to make sure that the corner point is on the boundary of the domain Ω. This can be done easily. In fact, we have a program to generate holes automatically given the corner tip vertex and two additional points on its two sides. Figure 6.15 shows the process. First find the angle bisector L_1. Then find the line L_2 perpendicular to L_1 and passing through the tip vertex. Finally a rectangular hole is generated along L_2. The position of the hole is to guarantee that the two sides of the corner are completely cut off right there. Also note that the shape of the hole can be

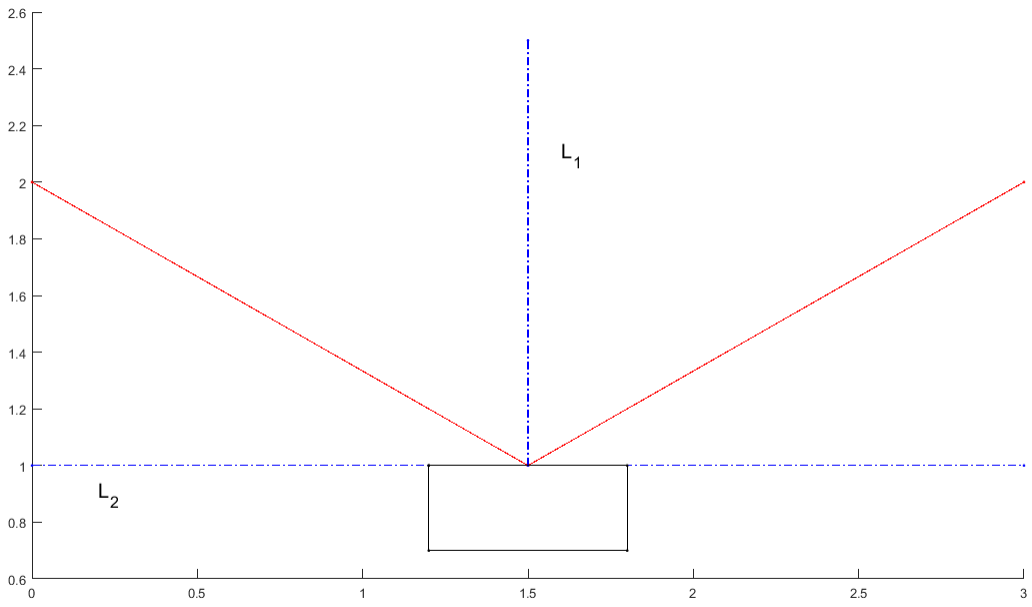

Figure 6.15: Generate a hole at the sharp corner.

arbitrary. It can be a rectangle, a triangle, or any polygon, as long as it can intercept the two sides.

Next is to find a triangulation of the domain Ω with the hole produced above. The triangulation is constrained by the two short line segments of the sides of the corner. This is the reason why we assumed that the tip of the corner is formed by two short line segments. Fig. 6.16 is such a constrained triangulation. The wanted contour is a triangle. So the triangulation has three holes corresponding to the three corners.

The final step is to compute the desired spline function as described in the previous subsections. The important thing is that the spline function must maintain constant 0 on the two short line segments at each corner. For example, in Fig. 6.16, the spline function should be constant 0 on all the red line segments. This can be achieved by manipulating the

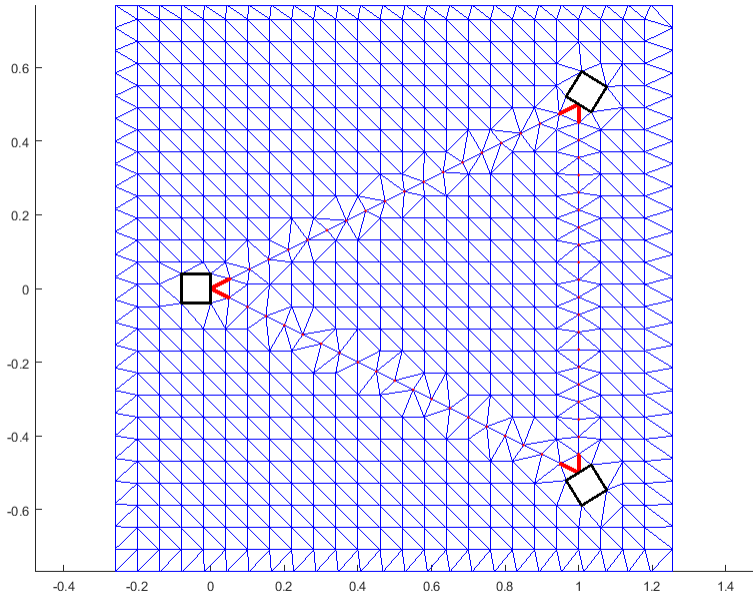

Figure 6.16: A constrained triangulation with holes.

corresponding domain points and putting them into the constraint (6.14). Also note that we can still enforce smoothness of the spline function, say, C^1 everywhere.

6.2.5 More Examples

Example 6.2.4. *We are given some points sampled from some curve with several sharp corners as shown by the red points in Fig. 6.17. A constrained triangulation* $\triangle$ *with holes is shown in Fig. 6.18. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. Note that the spline function needs to be constant* 0 *on the line segments that form the sharp corners. The result is shown in Fig. 6.19.*

Example 6.2.5. *We are given some points sampled from a rose curve as shown by the red points in Fig. 6.20. A constrained triangulation* $\triangle$ *is shown in Fig. 6.21. Note that there is*

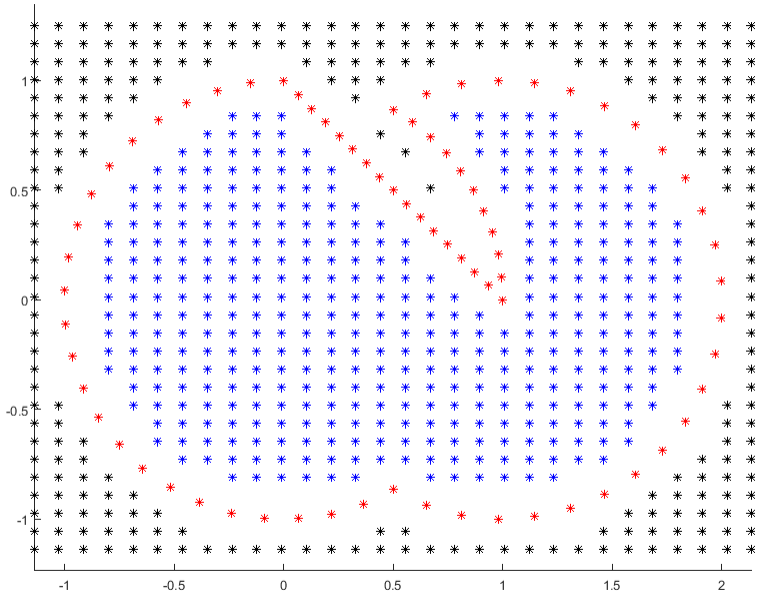

Figure 6.17: Data points for Example 6.2.4.

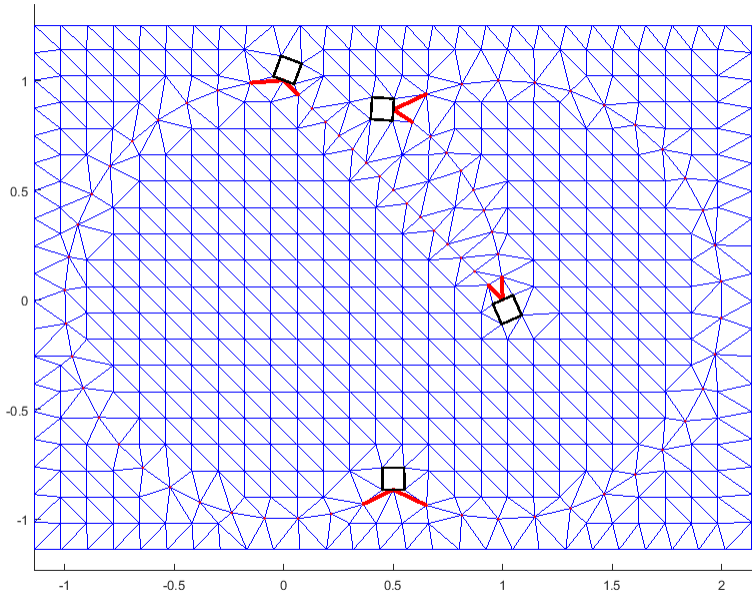

Figure 6.18: The triangulation for Example 6.2.4.

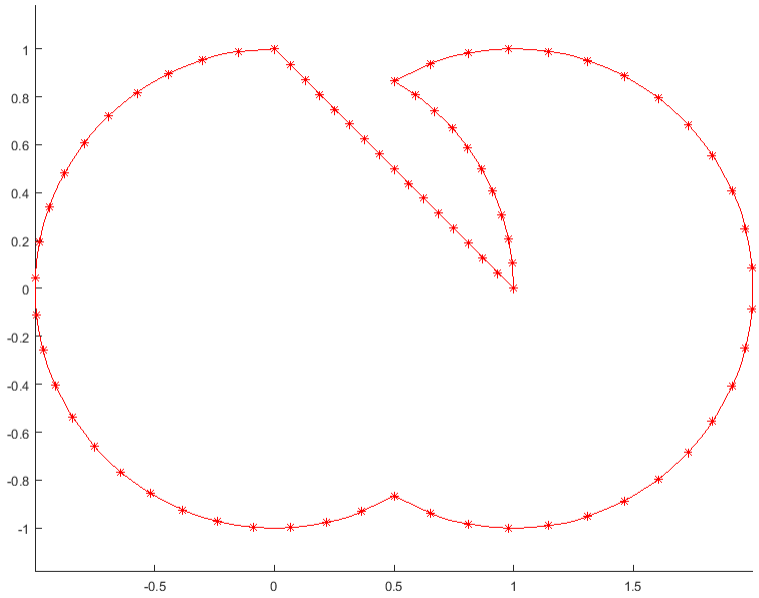

Figure 6.19: The contour curve for Example 6.2.4.

no hole in this triangulation. At the origin, we have several short line segments on which the spline function shall be constant 0 *so that the required shape can be generated. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. The result is shown in Fig. 6.22.*

Example 6.2.6. *We are given some points sampled from a slightly complicated curve with several sharp corners as shown by the red points in Fig. 6.23. A constrained triangulation* $\triangle$ *with holes is shown in Fig. 6.24. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data with* $\mathcal{A}_0'$ *composed of those red points. The result is shown in Fig. 6.25.*

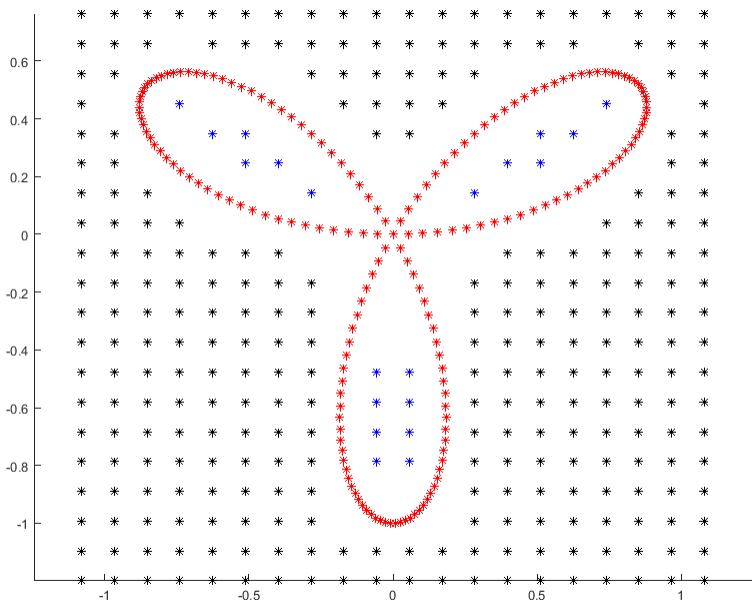

Figure 6.20: Data points for Example 6.2.5.

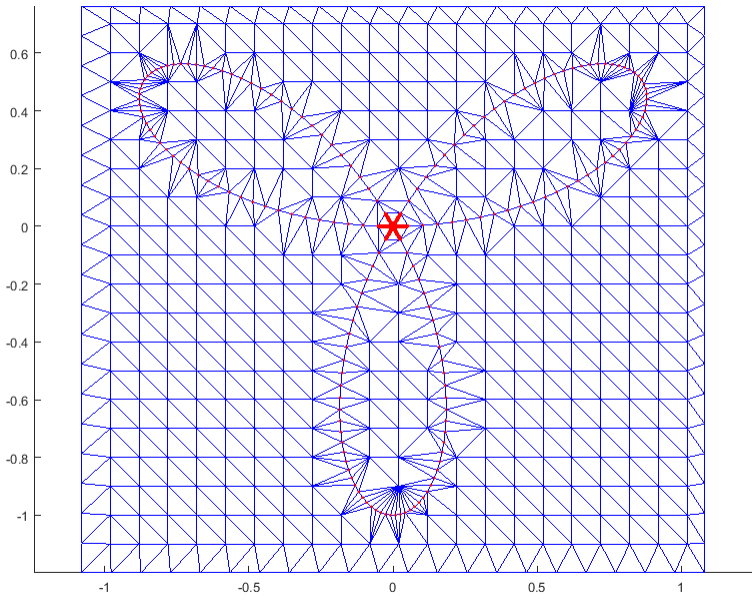

Figure 6.21: The triangulation for Example 6.2.5.

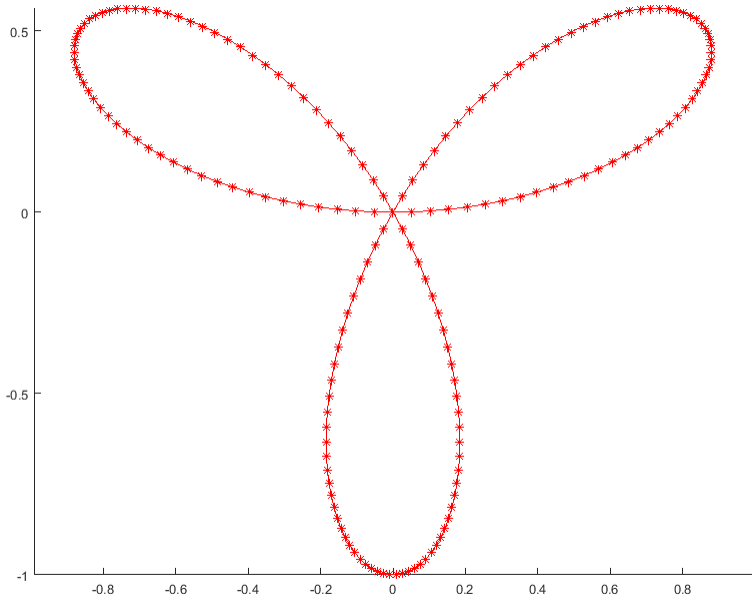

Figure 6.22: The contour curve for Example 6.2.5.

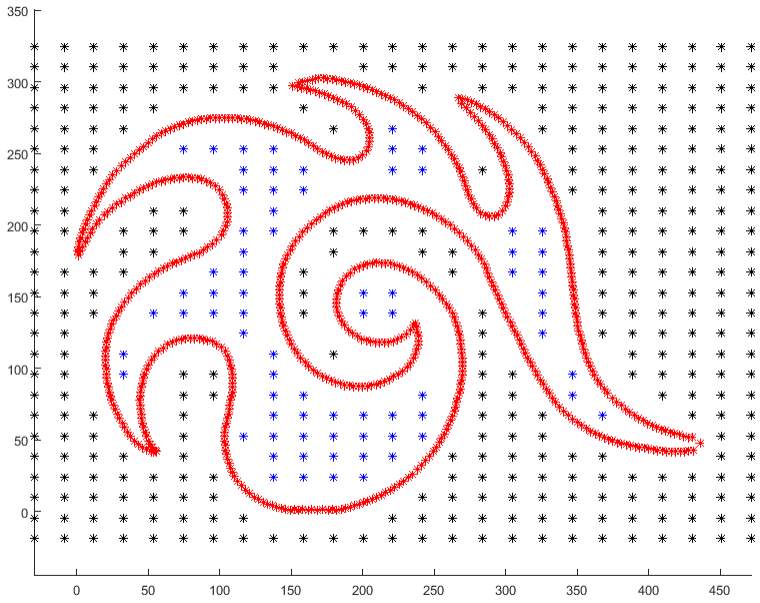

Figure 6.23: Data points for Example 6.2.6.

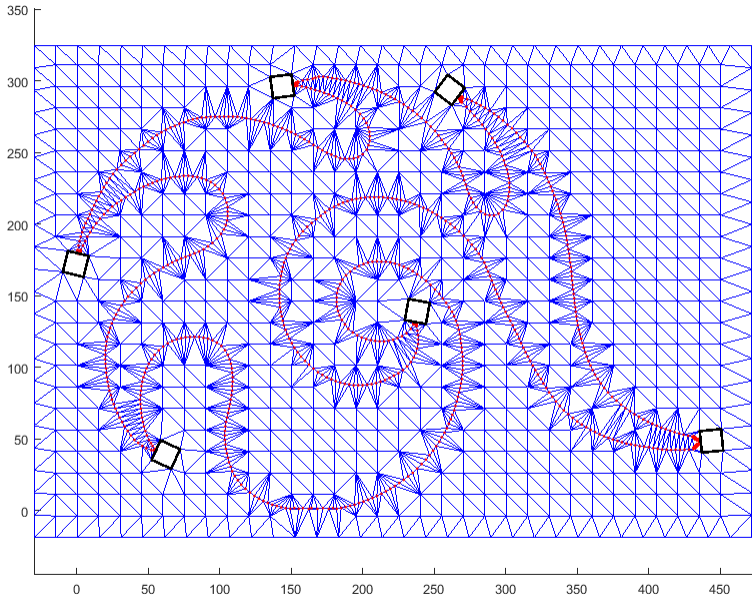

Figure 6.24: The triangulation for Example 6.2.6.

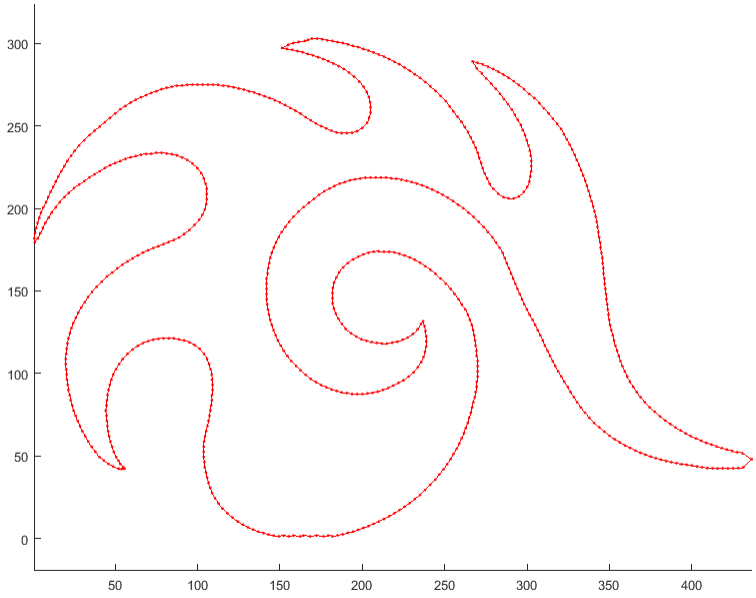

Figure 6.25: The contour curve for Example 6.2.6.

6.3 Construction of 3D Smooth Surfaces

In this section we describe the algorithm to generate smooth surfaces in the 3D setting. It is very similar to the 2D case.

6.3.1 Our Algorithm

Suppose we are given points $\mathcal{A}_0 := \{(x_i, y_i, z_i)\}_{i=1}^{N_0}$ from some unknown closed smooth surface C. We first construct a polyhedral domain $\Omega \subseteq \mathbb{R}^3$ which contains all the given points in its interior. For simplicity, we can just use a cuboid bounding box. Then we find a tetrahedralization $\triangle$ of Ω.

We wish to compute a smooth function s whose isosurface at value 0 approximates the desired surface. In order for the isosurface to stand out, we choose auxiliary points in the following sense. Pick points $\mathcal{A}_{-1} := \{(x_i, y_i, z_i)\}_{i=1}^{N_{-1}} \subset \Omega$ outside the closed surface, and assign them function value -1. Pick points $\mathcal{A}_1 := \{(x_i, y_i, z_i)\}_{i=1}^{N_1} \subset \Omega$ inside the closed surface, and assign them function value 1.

Now do the data fitting. Given $0 \leq r < d$ and a tetrahedralization $\triangle$, let

$$\mathcal{S}_d^r(\triangle) \;:=\; \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{6.16}$$

be the associated space of trivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of trivariate polynomials of degree at most d, as defined in Section 2.2. Also fix a constant $\lambda > 0$. The corresponding penalized least-squares (PLS) spline (cf. [1]) is defined to be:

$$s^* \;:=\; \arg\min_{s \in \mathcal{S}_d^r(\triangle)} \|s - (-1)\|_{\mathcal{A}_{-1}}^2 + \|s - 0\|_{\mathcal{A}_0}^2 + \|s - 1\|_{\mathcal{A}_1}^2 + \lambda E(s), \tag{6.17}$$

where

$$\|s - t\|_{\mathcal{A}}^2 \;:=\; \sum_{(x_i, y_i, z_i) \in \mathcal{A}} (s(x_i, y_i, z_i) - t)^2, \tag{6.18}$$

and

$$E(s) \;:=\; \int_\Omega \sum_{\substack{\alpha,\beta,\gamma\geq 0\\ \alpha+\beta+\gamma=2}} \left(\frac{\partial^2 s}{\partial x^\alpha \partial y^\beta \partial z^\gamma}\right)^2 dxdydz. \tag{6.19}$$

Since $s|_T$ is a polynomial of degree d on each tetrahedron $T \in \triangle$, we can use the B-form representation (2.58):

$$s|_T \;=\; \sum_{i+j+k+l=d} c^T_{ijkl} B^d_{ijkl}. \tag{6.20}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c \;:=\; [c^T_{i,j,k,l}, i+j+k+l=d, T\in\triangle], \tag{6.21}$$

where c is a column vector of dimension $m\binom{d+3}{3}$ with m denoting the number of tetrahedrons in $\triangle$. Moreover, using the B-form representation (2.58), we obtain

$$\|s-(-1)\|^2_{\mathcal{A}_{-1}} + \|s-0\|^2_{\mathcal{A}_0} + \|s-1\|^2_{\mathcal{A}_1} \;=\; \|Ac-b\|^2 \tag{6.22}$$

for some matrix A and $b = [-1,\ldots,-1,0,\ldots,0,1,\ldots,1]^\tau$. We also have

$$E(s) \;=\; c^\tau K c, \tag{6.23}$$

for some matrix K. Smoothness conditions from Theorem 2.2.12 helps us to write the constraint $s \in \mathcal{S}^r_d(\triangle)$ into a linear constraint:

$$Hc \;=\; 0 \tag{6.24}$$

for some matrix H.

Thus, the optimization problem (6.17) can be reformulated into the following form:

$$\min_c \qquad \|Ac-b\|^2 + \lambda c^\tau K c \tag{6.25}$$

$$\text{s.t.} \qquad Hc = 0. \tag{6.26}$$

We can apply the randomized domain decomposition method presented in Chapter 5 to find the solution.

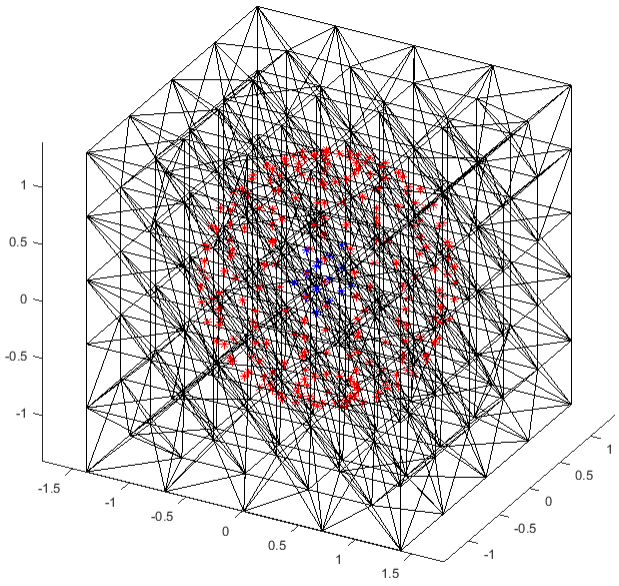

Figure 6.26: Data points and a tetrahedralization for Example 6.3.1.

6.3.2 Examples

Example 6.3.1. *We are given some points sampled from a sphere as shown by the red points in Fig. 6.26. We select some other points inside and outside the closed surface. The inside points are shown by the blue points in the figure. A tetrahedralization* $\triangle$ *is then constructed from the bounding box. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data. The result is shown in Fig. 6.27.*

Example 6.3.2. *We are given some points sampled from a torus as shown by the red points in Fig. 6.28. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data. The result is shown in Fig. 6.29.*

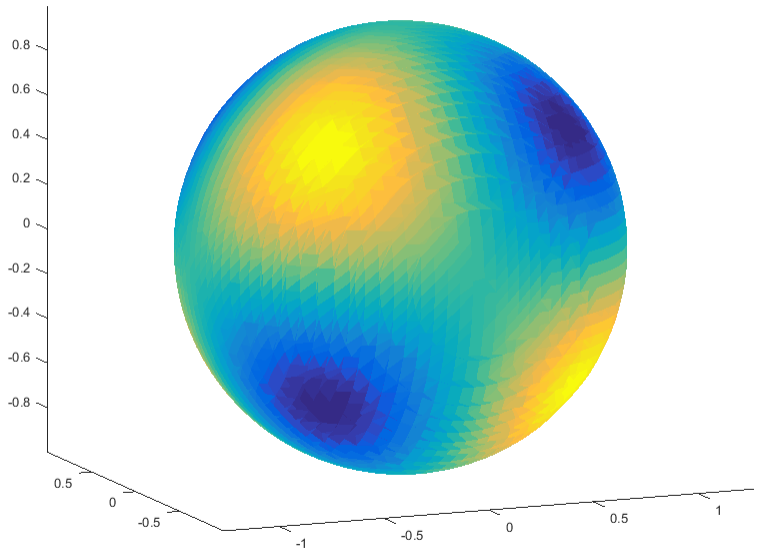

Figure 6.27: The isosurface for Example 6.3.1.

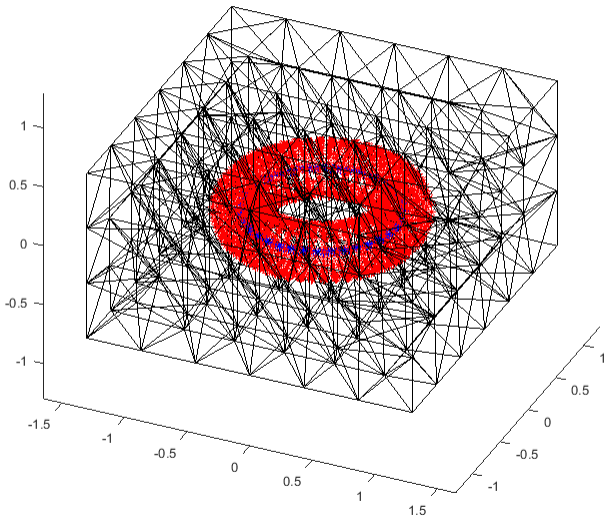

Figure 6.28: Data points and a tetrahedralization for Example 6.3.2.

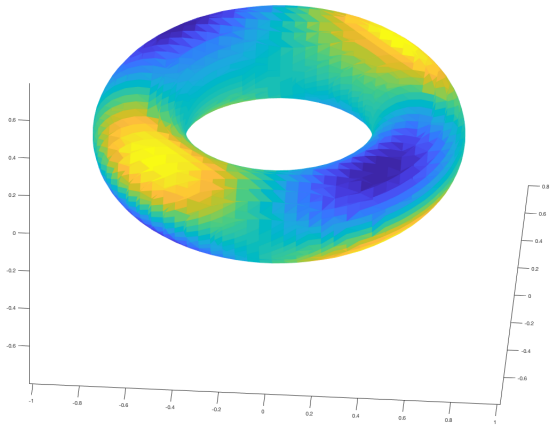

Figure 6.29: The isosurface for Example 6.3.2.

Example 6.3.3. *We are given some points sampled from a double torus as shown by the red points in Fig. 6.30. We compute the spline function* $s \in \mathcal{S}_5^1(\triangle)$ *to fit the data. The result is shown in Fig. 6.31. Fig. 6.32 shows the hollow interior of the isosurface.*

6.4 Future Work

There are lots of open problems in the surface reconstruction. For complicated 3D shapes, a customized constrained tetrahedralization might be necessary. However, as we mentioned in Chapter 3, this is not very easy. In addition, shape corner problems in 3D are much more involved than that in 2D. For example, we have not only pointy corners but also corner edges. These problems are quite challenging.

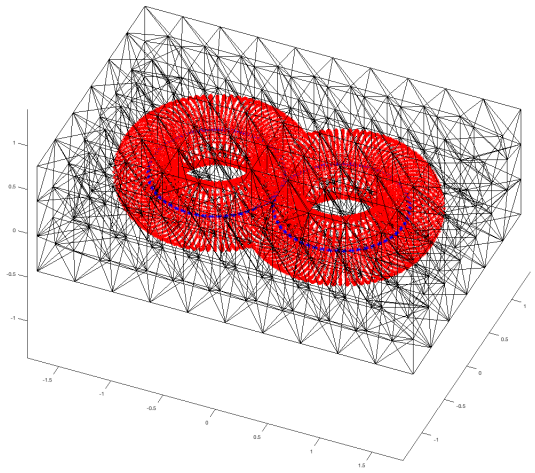

Figure 6.30: Data points and a tetrahedralization for Example 6.3.3.

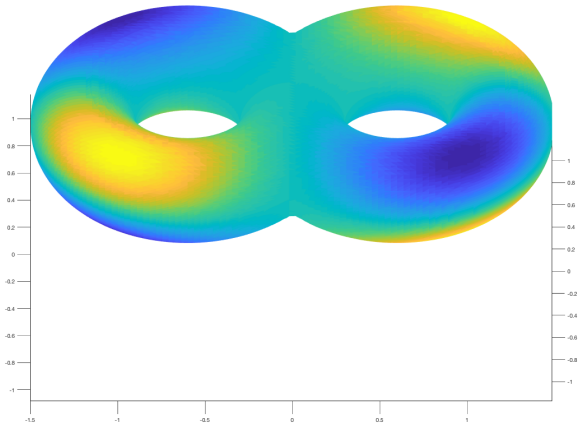

Figure 6.31: The isosurface for Example 6.3.3.

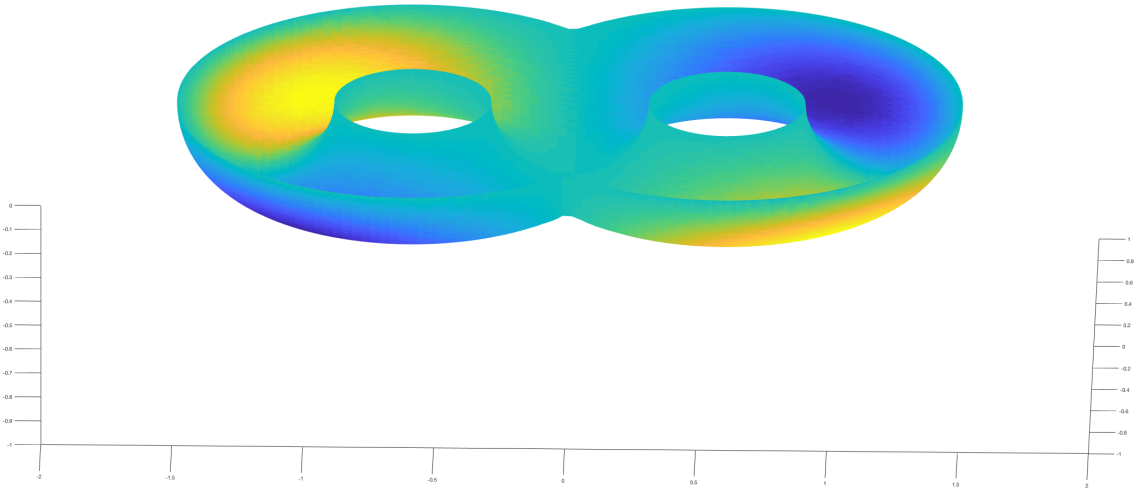

Figure 6.32: The hollow interior of the isosurface in Fig. 6.31.

Another possible future work is to analyze how many sample points are needed and how to choose the size of the triangulation/tetrahedralization so that the desired curve/surface has the minimum energy among all candidates and can be reconstructed.

Chapter 7

A Randomized Domain Decomposition Method for Spline Solutions of Poisson Equations

In this chapter, we present a randomized domain decomposition method for computing spline solutions of Poisson equations.

7.1 Introduction

We mainly introduce the two-dimensional problem. The three-dimensional case is similar.

7.1.1 Poisson Problems

Let Ω be a polygonal domain in Euclidean space $\mathbb{R}^2$ with boundary $\partial\Omega$. The problem of finding a function u satisfying the following partial differential equation with boundary condition

$$\begin{cases} -\Delta u = f & \text{in } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \tag{7.1}$$

is called the Dirichlet problem of Poisson equations, where $f \in L^2(\Omega)$ and g is continuous over $\partial\Omega$. The existence, uniqueness and stability of the solution has been studied extensively. See [14] for a detailed analysis.

The weak formulation is to find $u \in H^1(\Omega)$ such that

$$\int_\Omega \nabla u \cdot \nabla v dx dy \quad = \quad \int_\Omega f v dx dy. \tag{7.2}$$

holds for all $v \in H_0^1(\Omega)$, and $u = g$ on $\partial\Omega$.

Based on the standard calculus of variations, the weak solution is the minimizer of

$$E(u) \quad = \quad \int_\Omega \left(\frac{1}{2} \nabla u \cdot \nabla u - uf \right) dx dy \tag{7.3}$$

over the set of admissible functions (cf. [14]).

Our job is to compute approximate weak solutions with multivariate spline functions.

7.1.2 Related Works

Many PDEs are related to physical phenomena. Examples include Navier-Stokes equations in fluid mechanics, elasticity systems in solid mechanics, Schrödinger equations in quantum mechanics, etc. Solving such equations often leads to a series of linear systems. As the scale of the problems gets larger every day, more efficient and tractable algorithms are needed. Domain decomposition methods (DDMs) have thus received lots of attention. DDMs generally refer to the splitting of a PDE into coupled problems on smaller subdomains forming a partition of the original domain. Schwarz alternating method is one of the pioneers (cf. [44]). Many DDMs provide preconditioners that can be accelerated by Krylov subspace methods and operate on parallel machines (cf. [30], [2], and [24]).

In contrast, our algorithm is based on constrained optimization. We hope that it can provide some new perspective.

7.2 Two-Dimensional Algorithm

7.2.1 Spline Approximation of Weak Solutions

Given $0 \leq r < d$ and a triangulation $\triangle$ of Ω, let

$$\mathcal{S}_d^r(\triangle) \quad := \quad \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{7.4}$$

be the associated space of bivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of bivariate polynomials of degree at most d, as defined in Section 2.1.

Our goal is to find an approximate weak solution s^* such that

$$s^* := \arg \min_{\substack{s \in \mathcal{S}_d^r(\triangle) \\ s=g \text{ on } \partial\Omega}} E(s), \tag{7.5}$$

where

$$E(s) = \int_\Omega \left(\frac{1}{2} \nabla s \cdot \nabla s - sf \right) dxdy. \tag{7.6}$$

For the existence and uniqueness of the spline solution, see [1].

Since $s|_T$ is a polynomial of degree d on each triangle $T \in \triangle$, we can use the B-form representation (2.15):

$$s|_T = \sum_{i+j+k=d} c_{ijk}^T B_{ijk}^d. \tag{7.7}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c := [c_{i,j,k}^T, i+j+k=d, T \in \triangle], \tag{7.8}$$

where c is a column vector of dimension $m\binom{d+2}{2}$ with m denoting the number of triangles in $\triangle$. Moreover, using the B-form representation (2.15), we obtain

$$E(s) = \frac{1}{2} c^\tau K c - c_f^\tau M c, \tag{7.9}$$

for some matrices K and M, where c_f is the B-coefficient vector of the spline approximation of f. The boundary condition can be written as

$$Ac = b_g, \tag{7.10}$$

for some matrix A, where b_g is the vector composed of the values of g at the domain points on $\partial\Omega$. Smoothness conditions from Theorem 2.1.11 helps us to write the constraint $s \in \mathcal{S}_d^r(\triangle)$ into a linear constraint:

$$Hc = 0 \tag{7.11}$$

for some matrix H.

Thus, to find an approximate weak solution in $\mathcal{S}_d^r(\triangle)$, we need to solve the following minimization problem:

$$\min_c \quad \frac{1}{2}c^\tau Kc - c_f^\tau Mc \tag{7.12}$$

$$\text{s.t.} \quad Hc = 0, \tag{7.13}$$

$$Ac = b_g. \tag{7.14}$$

7.2.2 Our Algorithm

Instead of finding all the entries of the coefficient vector c at once, we solve a collection of smaller problems at each iteration. To state our algorithm formally, we need some notations similar to those in Chapter 5. For any subset $\delta \subset \triangle$, we set $\text{star}^0(\delta) = \delta$, and for all $k \geq 1$, recursively define

$$\text{star}^k(\delta) \quad = \quad \cup\{T \in \triangle : T \cap \text{star}^{k-1}(\delta) \neq \emptyset\}. \tag{7.15}$$

The algorithm is as follows.

Algorithm 7.2.1. *(Randomized Domain Decomposition Method for 2D Poisson Equations)*

1. *Fix a triangulation $\triangle$ of Ω. Choose $k > 0$. Apply Algorithm 5.2.1 to get an initial guess c_0 of the coefficient vector c such that c_0 satisfies the boundary condition (7.14).*

2. *Randomly select a triangle $T \in \triangle$. Find $\Omega_T^k := \text{star}^k(T)$.*

3. *Let $s_T^k \in \mathcal{S}_d^r(\triangle)|_{\Omega_T^k}$ be the restricted minimizer of (7.12) such that after updating $c_{i,j,k}^t = s_{i,j,k}^t$ for all $i+j+k = d$ and $t \in \Omega_T^k$, the resulting spline still satisfies (7.13) and (7.14).*

4. *If certain stopping criterion is met, quit; otherwise, go back to step 2.*

Let's take a look at how to find the local fit s_T^k in Step 3 above.

Write the coefficient vector c as $[c_1^\tau, c_2^\tau]^\tau$, where c_1 is the coefficient vector associated with the triangles in Ω_T^k, and c_2 is the rest. Write the smoothness matrix H in (7.13) as $[H_1, H_2]$ such that

$$Hc = Hc_1 + Hc_2. \tag{7.16}$$

Similarly, write the matrix A in (7.14) as $[A_1, A_2]$ such that

$$Ac = A_1c_1 + A_2c_2. \tag{7.17}$$

Let K_1, M_1 and c_{f1} be the submatrices of K, M and c_f related to the triangles in Ω_T^k respectively.

Now finding the local fit s_T^k is equivalent to solving the following minimization problem

$$\min_{c_1} \quad \frac{1}{2}c_1^\tau K_1 c_1 - c_{f1}^\tau M_1 c_1 \tag{7.18}$$

$$\text{s.t.} \quad H_1c_1 = -H_2c_2, \tag{7.19}$$

$$A_1c_1 = b_g - A_2c_2. \tag{7.20}$$

Note that the right-hand sides of the constraints (7.19) and (7.20) are constant since c_2 is fixed here.

We can use the same method as in Chapter 5 to solve the problem (7.18-7.20). For example, Lagrange multiplier can be used. Letting

$$\mathcal{L}(c_1, \xi, \eta) = \frac{1}{2}c_1^\tau K_1 c_1 - c_{f1}^\tau M_1 c_1 + \xi^\tau(H_1c_1 + H_2c_2) + \eta^\tau(A_1c_1 + A_2c_2 - b_g), \tag{7.21}$$

there exist ξ and η such that

$$K_1c_1 + H_1^\tau\xi + A_1^\tau\eta = M_1^\tau c_{f1}, \tag{7.22}$$

$$H_1c_1 = -H_2c_2, \tag{7.23}$$

$$A_1c_1 = b_g - A_2c_2. \tag{7.24}$$

The method of least squares or the iterative method described in [1] can be applied then.

7.2.3 Convergence Analysis

In this section, we analyze the convergence of Algorithm 7.2.1.

Since (7.18-7.20) is essentially the minimization problem (7.12-7.14) with respect to c_1 with c_2 fixed, it can be easily seen that Algorithm 7.2.1 is a randomized block coordinate descent method. Thus, we can use the results from Chapter 4 to analyze the convergence.

Theorem 7.2.2. *For the optimization problem (7.12-7.14), let*

$$g(c) := \frac{1}{2}c^\tau Kc - c_f^\tau Mc \tag{7.25}$$

be the objective function with optimal value g^*. If $\{c^{(n)}\}_{n\geq 0}$ is generated by Algorithm 7.2.1, then we have the following rate of convergence for the expected values of g:

$$\mathbb{E}[g(c^{(n)})] - g^* \;\leq\; \frac{M}{n} \tag{7.26}$$

for some constant $M > 0$.

Proof. The only difference between Algorithm 7.2.1 and Algorithm 4.3.3 is that instead of using (4.9), Algorithm 7.2.1 computes the minimum of the left-hand side of (4.6) directly by the method of least squares mentioned in the last subsection. However, this little difference would make no difference since the inequality (4.25) still holds. Thus, the proof of Theorem 4.5.3 passes through. Now this theorem follows. □

7.2.4 Numerical Examples

Example 7.2.3. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x, y) \;=\; 10e^{-x^2-y^2} \tag{7.27}$$

over a square domain. The triangulation $\triangle$ is shown in Fig. 7.1 with the red dots representing the boundary data points for the degree 5 case. There are 200 triangles. We use Algorithm 7.2.1 to find a spline function in $\mathcal{S}_d^r(\triangle)$. In the algorithm, we choose $k = 2$. The maximum errors are measured on 40401 points uniformly distributed over the domain. The results are summarized in Table 7.1.

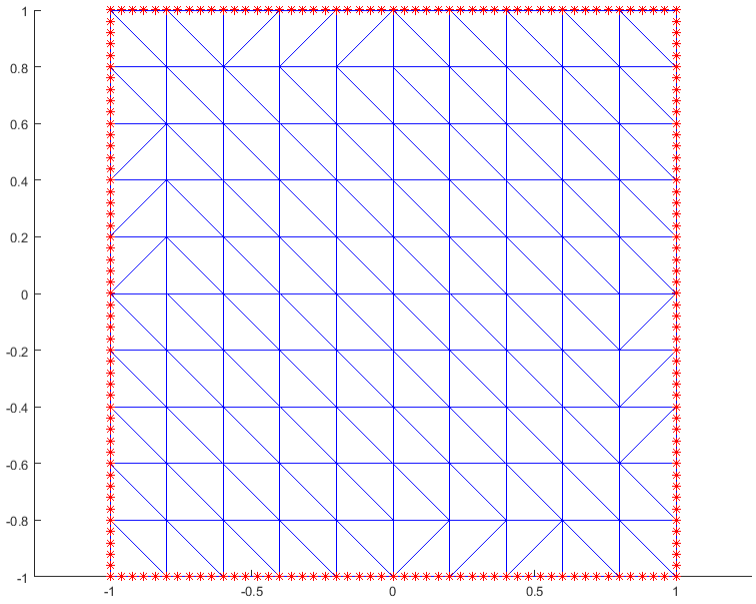

Figure 7.1: The triangulation for Example 7.2.3.

Table 7.1: Approximation errors in Example 7.2.3

	Errors	**CPU**
$\mathcal{S}_5^1$	1.25e-05	24.31s
$\mathcal{S}_6^1$	1.32e-05	33.97s
$\mathcal{S}_7^1$	1.26e-05	51.37s
$\mathcal{S}_8^1$	8.19e-06	76.05s

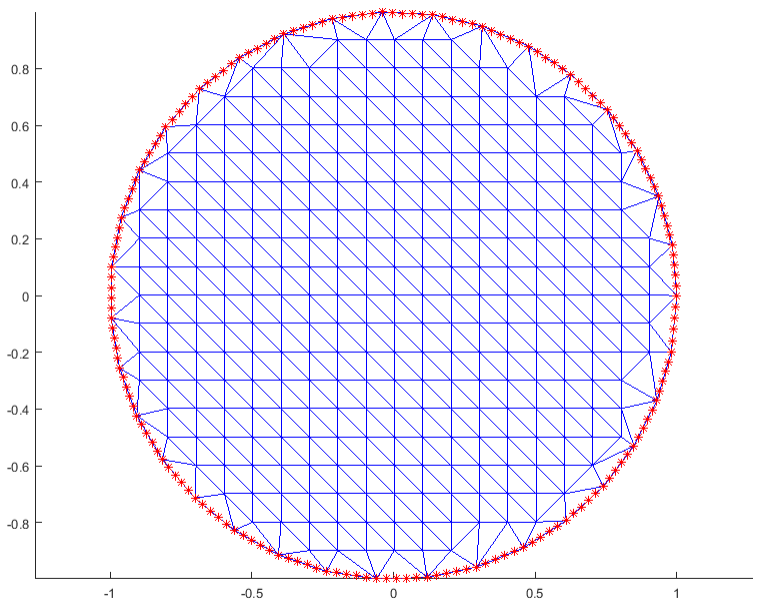

Figure 7.2: The triangulation for Example 7.2.4.

Example 7.2.4. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x,y) \quad = \quad \sin(\pi(2x+y)) - y^2 + x + 2y \tag{7.28}$$

over a convex polygon. The triangulation is shown in Fig. 7.2. There are 607 *triangles. We choose* $k = 4$ *in the algorithm. The maximum errors are measured on* 31234 *points uniformly distributed over the domain. The results are summarized in Table 7.2.*

Example 7.2.5. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x,y) \quad = \quad (x+y)\sin(\pi(2x+y)) + \frac{1}{x^2+y^2+1} + (x+y)e^{x^2-y} \tag{7.29}$$

over a non-convex polygonal domain with two holes. The triangulation is shown in Fig. 7.3. There are 1169 *triangles. We choose* $k = 4$ *in the algorithm. The maximum errors are measured on* 59945 *points uniformly distributed over the domain. The results are summarized in Table 7.3.*

Table 7.2: Approximation errors in Example 7.2.4

	Errors	**CPU**
$\mathcal{S}_5^1$	6.62e-05	76.11s
$\mathcal{S}_6^1$	1.73e-05	118.16s
$\mathcal{S}_7^1$	1.64e-05	231.31s
$\mathcal{S}_8^1$	9.72e-06	415.52s

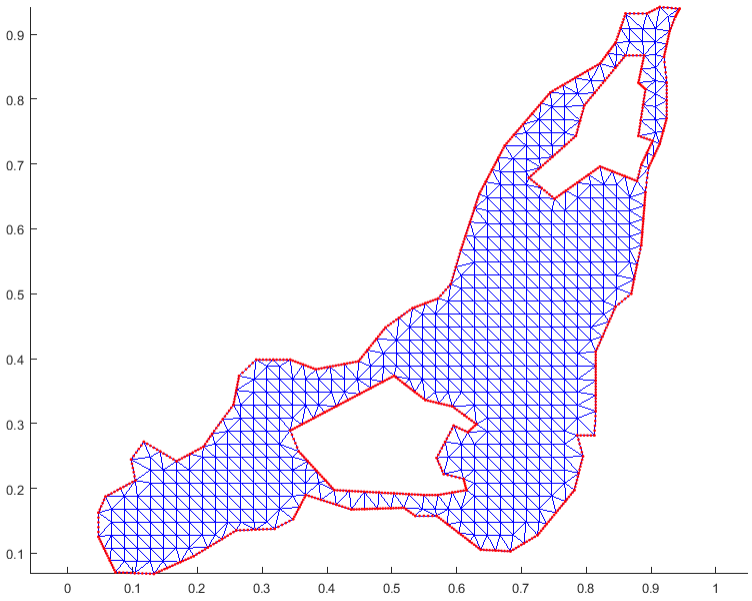

Figure 7.3: The triangulation for Example 7.2.5.

Table 7.3: Approximation errors in Example 7.2.5

	Errors	**CPU**
$\mathcal{S}_5^1$	1.58e-05	164.01s
$\mathcal{S}_6^1$	2.01e-05	277.22s
$\mathcal{S}_7^1$	1.57e-05	539.41s
$\mathcal{S}_8^1$	1.08e-05	680.11s

7.3 Three-Dimensional Algorithm

The three-dimensional randomized domain decomposition method is essentially the same as the two-dimensional one. We briefly present the algorithm and illustrate its effectiveness by several examples.

7.3.1 Spline Approximation of Weak Solutions

Let $\Omega \subseteq \mathbb{R}^3$ be a polyhedral domain. Given $0 \leq r < d$ and a tetrahedralization $\triangle$ of Ω, let

$$\mathcal{S}_d^r(\triangle) \;:=\; \{s \in C^r(\Omega) : s|_T \in \mathcal{P}_d, \text{for all } T \in \triangle\} \tag{7.30}$$

be the associated space of trivariate splines of degree d and smoothness r, where $\mathcal{P}_d$ is the space of trivariate polynomials of degree at most d, as defined in Section 2.2.

Our goal is to find an approximate weak solution s^* such that

$$s^* \;:=\; \arg \min_{\substack{s \in \mathcal{S}_d^r(\triangle) \\ s=g \text{ on } \partial\Omega}} E(s), \tag{7.31}$$

where

$$E(s) \;=\; \int_\Omega \left(\frac{1}{2} \nabla s \cdot \nabla s - sf \right) dxdydz. \tag{7.32}$$

Since $s|_T$ is a polynomial of degree d on each tetrahedron $T \in \triangle$, we can use the B-form representation (2.58):

$$s|_T \;=\; \sum_{i+j+k+l=d} c_{ijkl}^T B_{ijkl}^d. \tag{7.33}$$

Hence, finding the spline solution is equivalent to finding the B-coefficient vector

$$c \;:=\; [c_{i,j,k,l}^T, i+j+k+l=d, T \in \triangle], \tag{7.34}$$

where c is a column vector of dimension $m\binom{d+3}{3}$ with m denoting the number of tetrahedrons in $\triangle$. Moreover, using the B-form representation (2.58), we obtain

$$E(s) \;=\; \frac{1}{2} c^\tau K c - c_f^\tau M c, \tag{7.35}$$

for some matrices K and M, where c_f is the B-coefficient vector of the spline approximation of f. The boundary condition can be written as

$$Ac = b_g, \tag{7.36}$$

for some matrix A, where b_g is the vector composed of the values of g at the domain points on $\partial\Omega$. Smoothness conditions from Theorem 2.2.12 helps us to write the constraint $s \in \mathcal{S}_d^r(\triangle)$ into a linear constraint:

$$Hc = 0 \tag{7.37}$$

for some matrix H.

Thus, to find an approximate weak solution in $\mathcal{S}_d^r(\triangle)$, we need to solve the following minimization problem:

$$\min_c \quad \frac{1}{2}c^\tau Kc - c_f^\tau Mc \tag{7.38}$$

$$\text{s.t.} \quad Hc = 0, \tag{7.39}$$

$$Ac = b_g. \tag{7.40}$$

7.3.2 Our Algorithm

As in the two-dimensional algorithm, instead of finding all the entries of the coefficient vector c at once, we solve a collection of smaller problems at each iteration.

For any subset $\delta \subset \triangle$, we set $\text{star}^0(\delta) = \delta$, and for all $k \geq 1$, recursively define

$$\text{star}^k(\delta) = \cup\{T \in \triangle : T \cap \text{star}^{k-1}(\delta) \neq \emptyset\}. \tag{7.41}$$

The algorithm is as follows.

Algorithm 7.3.1. *(Randomized Domain Decomposition Method for 3D Poisson Equations)*

1. *Fix a tetrahedralization $\triangle$ of Ω. Choose $k > 0$. Apply Algorithm 5.3.1 to get an initial guess c_0 of the coefficient vector c such that c_0 satisfies the boundary condition (7.40).*

2. *Randomly select a tetrahedron* $T \in \triangle$. *Find* $\Omega_T^k := \text{star}^k(T)$.

3. *Let* $s_T^k \in \mathcal{S}_d^r(\triangle)|_{\Omega_T^k}$ *be the restricted minimizer of (7.38) such that after updating* $c_{i,j,k,l}^t = s_{i,j,k,l}^t$ *for all* $i+j+k+l = d$ *and* $t \in \Omega_T^k$, *the resulting spline still satisfies (7.39) and (7.40).*

4. *If certain stopping criterion is met, quit; otherwise, go back to step 2.*

The local fit s_T^k in Step 3 above can be computed in the same way as in the two-dimensional case.

Write the coefficient vector c as $[c_1^\tau, c_2^\tau]^\tau$, where c_1 is the coefficient vector associated with the tetrahedrons in Ω_T^k, and c_2 is the rest. Write the smoothness matrix H in (7.39) as $[H_1, H_2]$ such that

$$Hc \;=\; Hc_1 + Hc_2. \tag{7.42}$$

Similarly, write the matrix A in (7.40) as $[A_1, A_2]$ such that

$$Ac \;=\; A_1c_1 + A_2c_2. \tag{7.43}$$

Let K_1, M_1 and c_{f1} be the submatrices of K, M and c_f related to the tetrahedrons in Ω_T^k respectively.

Now finding the local fit s_T^k is equivalent to solving the following minimization problem

$$\min_{c_1} \quad \frac{1}{2}c_1^\tau K_1 c_1 - c_{f1}^\tau M_1 c_1 \tag{7.44}$$

$$\text{s.t.} \quad H_1c_1 = -H_2c_2, \tag{7.45}$$

$$A_1c_1 = b_g - A_2c_2. \tag{7.46}$$

7.3.3 Convergence Analysis

The convergence of Algorithm 7.3.1 is essentially the same as that of Algorithm 7.2.1. So we give the following theorem without proof.

Table 7.4: Approximation errors in Example 7.3.3

	Errors	**CPU**
$\mathcal{S}_5^1$	7.93e-05	813.96s
$\mathcal{S}_6^1$	9.07e-07	1440.37s

Theorem 7.3.2. *For the optimization problem (7.38- 7.40), let*

$$g(c) := \frac{1}{2}c^\tau Kc - c_f^\tau Mc \tag{7.47}$$

be the objective function with optimal value g^*. If $\{c^{(n)}\}_{n\geq 0}$ is generated by Algorithm 7.3.1, then we have the following rate of convergence for the expected values of g:

$$\mathbb{E}[g(c^{(n)})] - g^* \;\leq\; \frac{M}{n} \tag{7.48}$$

for some constant $M > 0$.

7.3.4 Numerical Examples

Example 7.3.3. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x, y, z) \;=\; 5e^{-(x-0.5)^2-(y-0.5)^2-(z-0.5)^2} \tag{7.49}$$

over a cube. The tetrahedralization $\triangle$ is shown in Fig. 7.4. There are 384 tetrahedrons. We use Algorithm 7.3.1 to find a spline function in $\mathcal{S}_d^r(\triangle)$. In the algorithm, we choose $k = 2$. The maximum errors are measured on 125000 points uniformly distributed over the domain. The results are summarized in Table 7.4.

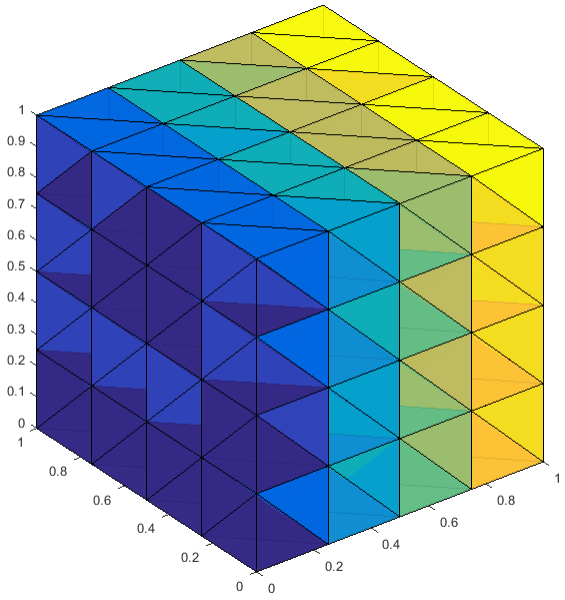

Figure 7.4: The tetrahedralization for Example 7.3.3.

Example 7.3.4. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x, y, z) \quad = \quad \sin(\pi(x + y + 2z)) + x^2 - y + z \tag{7.50}$$

over a non-convex domain. The tetrahedralization $\triangle$ *is shown in Fig. 7.5. There are* 312 *tetrahedrons. We choose* $k = 2$ *in the algorithm. The maximum errors are measured on* 95000 *points uniformly distributed over the domain. The results are summarized in Table 7.5.*

Example 7.3.5. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x, y, z) \quad = \quad \sin(2\pi(x + y + z)) \tag{7.51}$$

over the same non-convex domain as in Example 7.3.4. But the tetrahedralization here is more refined with 619 *tetrahedrons as shown in Fig. 7.6. We choose* $k = 2$ *in the algorithm.*

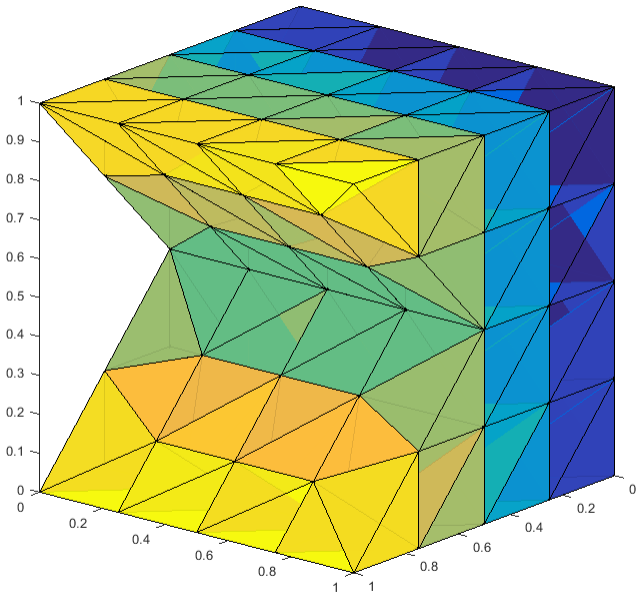

Figure 7.5: The tetrahedralization for Example 7.3.4.

Table 7.5: Approximation errors in Example 7.3.4

	Errors	**CPU**
$\mathcal{S}_5^1$	9.58e-04	630.29s
$\mathcal{S}_6^1$	4.92e-05	611.98s

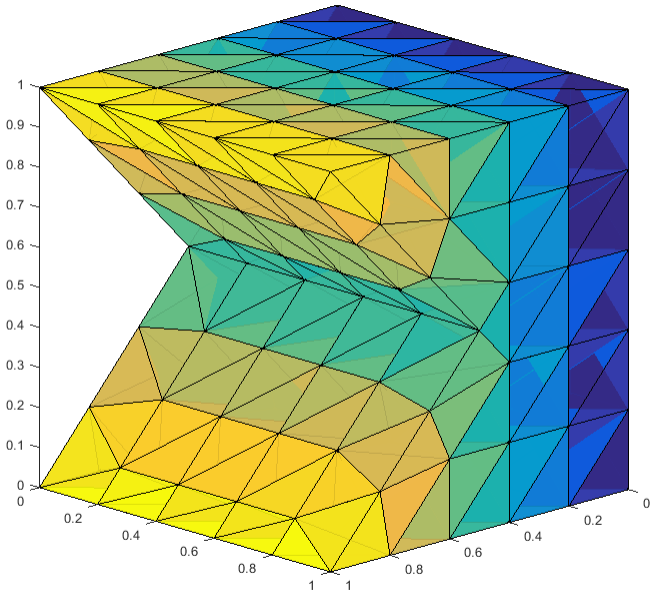

Figure 7.6: The tetrahedralization for Example 7.3.5.

Table 7.6: Approximation errors in Example 7.3.5

	Errors	**CPU**
$\mathcal{S}_5^1$	2.30e-03	2344.87s
$\mathcal{S}_6^1$	2.44e-04	2590.52s

The maximum errors are measured on 95000 *points uniformly distributed over the domain. The results are summarized in Table 7.6.*

Example 7.3.6. *Consider the Dirichlet problem of Poisson equation with exact solution*

$$u(x,y,z) \;=\; (x+y+z)\sin(\pi(2x+y-z)) + \frac{2}{x^2+y^2+z^2+1} + (x-z)e^{x^2-y} \quad (7.52)$$

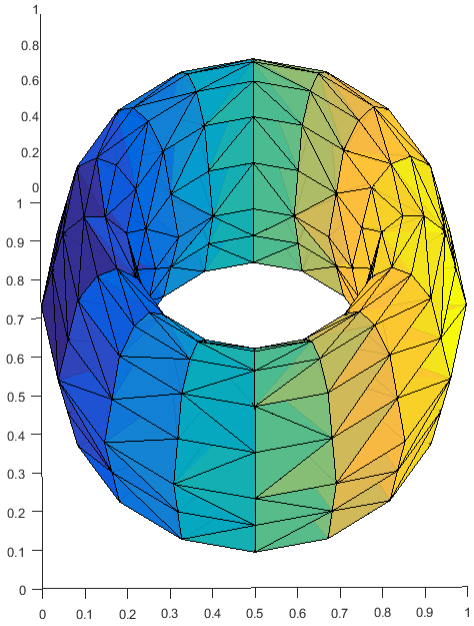

Figure 7.7: The tetrahedralization for Example 7.3.6.

Table 7.7: Approximation errors in Example 7.3.6

	Errors	**CPU**
$\mathcal{S}_5^1$	8.20e-03	745.77s
$\mathcal{S}_6^1$	3.93e-04	763.38s

over a torus-shape domain. The tetrahedralization $\triangle$ *is shown in Fig. 7.7. There are* 588 *tetrahedrons. We choose* $k = 2$ *in the algorithm. The maximum errors are measured on* 49296 *points uniformly distributed over the domain. The results are summarized in Table 7.7.*

7.4 Remarks and Future Work

Since the goal is to have $-\Delta u = f$ in Ω, we can mix in some empirical ways to choose blocks. For example, for every odd iteration, we randomly pick a triangle/tetrahedron T and update the spline on $\text{star}^k(T)$, while for every even iteration, we pick T on which $-\Delta u = f$ is the least satisfied. This alternating method will not undermine the convergence since it is still a descent method.

One possible future work would be to extend this approach to other types of PDEs. Another direction is to use parallel or distributed computing to accelerate the computation. Extra care is needed due to the coupled constraints and overlapping blocks.

Bibliography

[1] G. Awanou, M.-J. Lai, and P. Wenston. The multivariate spline method for scattered data fitting and numerical solutions of partial differential equations. *Wavelets and splines: Athens*, pages 24–74, 2005.

[2] S. Badia and F. Verdugo. Robust and scalable domain decomposition solvers for unfitted finite element methods. *Journal of Computational and Applied Mathematics*, 344:740–759, 2018.

[3] A. Beck. *Introduction to nonlinear optimization: Theory, algorithms, and applications with MATLAB*, volume 19. Siam, 2014.

[4] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 1:23–90, 1992.

[5] D. P. Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[6] J.-D. Boissonnat, D. Cohen-Steiner, B. Mourrain, G. Rote, and G. Vegter. Meshing of surfaces. In *Effective Computational Geometry for Curves and Surfaces*, pages 181–229. Springer, 2006.

[7] S. C. Brenner and L. R. Scott. The mathematical theory of finite element methods. 2002. *Texts in applied mathematics*, 2008.

[8] F. Brezzi and M. Fortin. *Mixed and hybrid finite element methods*, volume 15. Springer Science & Business Media, 2012.

[9] P. G. Ciarlet. *The finite element method for elliptic problems*, volume 40. Siam, 2002.

[10] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.

[11] W. Deng, M.-J. Lai, Z. Peng, and W. Yin. Parallel multi-block admm with o (1/k) convergence. *Journal of Scientific Computing*, 71(2):712–736, 2017.

[12] R. A. Dwyer. A faster divide-and-conquer algorithm for constructing delaunay triangulations. *Algorithmica*, 2(1-4):137–151, 1987.

[13] N. Dyn, D. Levine, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM transactions on Graphics (TOG)*, 9(2):160–169, 1990.

[14] L. C. Evans. Partial differential equations, 2 nd. *Graduate Studies in Mathematics*, 19, 2010.

[15] G. E. Farin. *NURB curves and surfaces: from projective geometry to practical use*. AK Peters, Ltd., 1995.

[16] D. A. Field. Qualitative measures for initial meshes. *International Journal for Numerical Methods in Engineering*, 47(4):887–906, 2000.

[17] C. Frederick, Y. Wong, and F. Edge. Two-dimensional automatic mesh generation for structural analysis. *International Journal for Numerical Methods in Engineering*, 2(1):133–144, 1970.

[18] I. Gutman and W. Xiao. Generalized inverse of the laplacian matrix and some applications. *Bulletin (Académie serbe des sciences et des arts. Classe des sciences mathématiques et naturelles. Sciences mathématiques)*, pages 15–23, 2004.

[19] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144, 2001.

[20] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[21] C.-J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1064–1072. ACM, 2011.

[22] C.-J. Hsieh, I. S. Dhillon, P. K. Ravikumar, and M. A. Sustik. Sparse inverse covariance matrix estimation using quadratic approximation. In *Advances in neural information processing systems*, pages 2330–2338, 2011.

[23] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[24] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'Homme. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. *Scientific Programming*, 22(2):157–171, 2014.

[25] M.-J. Lai. Multivariate splines for data fitting and approximation. *Approximation Theory XII: San Antonio*, pages 210–228, 2007.

[26] M.-J. Lai and C. Mersmann. Adaptive triangulation methods for bivariate spline solutions of the poisson equation. 2016.

[27] M.-J. Lai and L. L. Schumaker. *Spline functions on triangulations*. Number 110. Cambridge University Press, 2007.

[28] M.-J. Lai and L. L. Schumaker. A domain decomposition method for computing bivariate spline fits of scattered data. *SIAM Journal on Numerical Analysis*, 47(2):911–928, 2009.

[29] C. L. Lawson. Software for c1 surface interpolation. In *Mathematical software*, pages 161–194. Elsevier, 1977.

[30] R. Li and Y. Saad. Low-rank correction methods for algebraic domain decomposition preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 38(3):807–828, 2017.

[31] J. Linhart. A quick point-in-polyhedron test. *Computers & graphics*, 14(3-4):445–447, 1990.

[32] D. L. Marcum and N. P. Weatherill. Unstructured grid generation using iterative point insertion and local reconnection. *AIAA journal*, 33(9):1619–1625, 1995.

[33] C. A. Micchelli. *Mathematical aspects of geometric modeling*. SIAM, 1995.

[34] I. Necoara, Y. Nesterov, and F. Glineur. A random coordinate descent method on large optimization problems with linear constraints. *University Politehnica Bucharest, Tech. Rep*, 2011.

[35] J. Peraire, J. Peiró, and K. Morgan. Advancing front grid generation. *Handbook of grid generation*, pages 17–1, 1999.

[36] C. Poignard, T. Pereira, and J. P. Pade. Spectra of laplacian matrices of weighted graphs: structural genericity properties. *SIAM Journal on Applied Mathematics*, 78(1):372–394, 2018.

[37] S. Reddi, A. Hefny, C. Downey, A. Dubey, and S. Sra. Large-scale randomized-coordinate descent methods with non-separable linear constraints. *arXiv preprint arXiv:1409.2617*, 2014.

[38] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *The Journal of Machine Learning Research*, 17(1):2657–2681, 2016.

[39] R. Schneiders. Refining quadrilateral and hexahedral element meshes. *transition*, 2:1, 1996.

[40] L. L. Schumaker. Computing bivariate splines in scattered data fitting and the finite-element method. *Numerical Algorithms*, 48(1-3):237–260, 2008.

[41] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14(Feb):567–599, 2013.

[42] J. R. Shewchuk. Lecture notes on delaunay mesh generation. 1999.

[43] J. Thompson and N. WEATHERILL. Aspects of numerical grid generation-current science and art. In *11th Applied Aerodynamics Conference*, page 3539, 1993.

[44] A. Toselli and O. Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2006.

[45] M. von Golitschek, M.-J. Lai, and L. L. Schumaker. Error bounds for minimal energy bivariate polynomial splines. *Numerische Mathematik*, 93(2):315–331, 2002.

[46] M. Von Golitschek and L. L. Schumaker. Bounds on projections onto bivariate polynomial spline spaces with stable local bases. *Constructive approximation*, 18(2):241–254, 2002.

[47] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.

[48] A. Ženíšek. Polynomial approximation on tetrahedrons in the finite element method. *Journal of Approximation Theory*, 7(4):334–351, 1973.