IDEA League

Master of Science in Applied Geophysics

Research Thesis

Triangulation for seismic modelling with optimization techniques

Weijun Wang

August 10, 2017

Triangulation for seismic modelling with optimization techniques

Master of Science Thesis

for the degree of Master of Science in Applied Geophysics
by

Weijun Wang

August 10, 2017

IDEA LEAGUE
JOINT MASTER'S IN APPLIED GEOPHYSICS

Delft University of Technology, The Netherlands
ETH Zürich, Switzerland
RWTH Aachen, Germany

Dated: *August 10, 2017*

Supervisor(s):

Prof.dr. W.A. Mulder

Committee Members:

Prof.dr. W.A. Mulder

Prof.dr.ir. E.C. Slob

Prof.dr. J.F. Wellmann

Abstract

The finite-element method can easily handle complicated geometries because of the application of unstructured meshes. Unlike the Cartesian grid used in the finite-difference method, the unstructured mesh can follow the sharp interfaces that separate two layers of different properties. Therefore, the finite-element method can provide more accurate solutions for the simulation of seismic wave propagation. Meshes of good quality are required for the finite-element simulation. However, it is not trivial to set up an appropriate mesh. First of all, the mesh should contain elements of good shapes and sizes. In addition, the sharp interfaces should coincide with the edges of the elements instead of intersecting with them. These requirements are formulated as an optimization problem with three terms, measuring the difference between the actual and prescribed scaling field, shape quality, and the area between prescribed curves and the nearest triangle edges. The solution of the optimization problem should provide the desired mesh. The mesh generator MESH2D was applied to obtain an initial mesh. The Matlab function minFunc was used to search for the minimum of the constructed objective function. Three weights balance the three terms in the objective function. When it comes to complicated models, these weights have to be chosen carefully to produce a reasonable mesh.

Acknowledgements

First of all, I want to thank my supervisor, Prof.dr. W.A. Mulder, for his preliminary work and kind help during the whole period of my thesis. Without his supervision, I would have needed considerably more time and effort to solve the problems occurring during the preparation of this thesis. Also I appreciate a lot him agreeing on my working at home, due to the remoteness of my living place, which saved me a lot of time otherwise spent on commuting.

In addition, I express my thanks to my dear wife and unborn baby, expected to be born this September. My wife is always by my side and encourages me when I feel frustrated. As a husband and father-to-be, a sense of responsibility motivates me to try my best to overcome all kinds of obstacles.

Furthermore, I want to express my gratitude to the IDEA LEAGUE and Royal Dutch Shell for their provision of the comprehensive scholarship that covers everything during my master program. With their financial support, I had the opportunity to participate as an international student in this excellent program.

Delft University of Technology — Weijun Wang
August 10, 2017

Table of Contents

List of Figures

Chapter 1

Introduction

The finite-difference method has been the preferred option for time-domain modelling of the wave equation, with applications in acquisition optimization, development and testing of seismic processing algorithm, reverse-time migration and full waveform inversion. This is due to the fact that finite-difference method has some inherent merits, such as the relative ease of coding and low computational cost. However, this method is mainly implemented on rectangular domains with smooth velocity variations. When it comes to a domain with irregular free surface or sharp contrasts in the properties of the medium, the finite-difference method on a Cartesian grid will lose accuracy, since the grid does not fit the irregular interface precisely, which results in the staircasing effect (Zhebel et al., 2014).

Compared with the finite-difference method, the finite-element method has the advantage that complicated geometries can be handled relatively easily with the help of unstructured meshes and spatial local refinement. By using unstructured meshes, the sharp interfaces can be followed to avoid the staircasing effect and loss of accuracy. Besides, the size of the elements can be adjusted according to specific parameters of the medium in order to increase the computational efficiency.

The finite-element method requires meshes that consist of elements with the right shapes and sizes. However, generating a suitable unstructured mesh is a non-trivial task. For the sake of computational efficiency, the elements should maintain good shapes and sizes. In addition, sharp interfaces should be followed by the elements. Namely, the interfaces should coincide with the edges of the elements or be as close as possible. In order to fulfil all these requirements, we can utilize the tools of optimization by formulating our design criteria as some appropriate objective functions. Then a desired mesh can be generated after solving the optimization problem.

In the engineering world, an equilateral triangle is universally accepted as the ideal element in a 2D mesh. Conversely, elements with either too large or too small angles are considered as bad elements and are avoided by most mesh generators. But is this universal recognition really true? Are triangles with too large or too small angles always defined as being of poor quality? How to evaluate the quality of a triangle mathematically? These questions will be discussed in chapter 2 in detail.

We only focus on 2D unstructured mesh generation in this project. Many algorithms can be chosen to generate a decent mesh that meets user-defined requirements. These algorithms include advancing front methods, quadtree methods, and Delaunay methods. The way to generate a mesh differs, and so does the metric of the element quality. In chapter 3, I will review these three algorithms, including the way they operate.

In this project, we formulated the mesh generation as an optimization problem. Optimization refers to the process of locating the minimum or maximum of an objective function, which expresses our requirements in mathematical terms. Therefore, we need an algorithm that can find the minimum or maximum of the objective function. In chapter 4, I will present different search methods based on the use of the objective function values, their gradients, and their Hessians. Each method has its own scope of application and rate of convergence. According to the objective function that needs to be optimized, an appropriate method has to be chosen. Apart from that, the Optimization Toolbox provided by Matlab will also be reviewed in this chapter.

In chapter 5, I will explain step by step how the problem is formulated. This includes the construction of the terms in the objective function, the calculation of the gradients, and the way we combine the different terms in objective function. In addition, the meshes generated by the optimization will be presented for both simple and complicated models.

This thesis provides a 2D prototype for the unstructured mesh generation. The generated 2D mesh maintains a reasonable quality in both shapes and sizes. Additionally, the topography as well as the seismic horizons and faults can be followed by the mesh. This project can serve as a foundation of the further 3D research on related topics.

Part I

Theory

Chapter 2

Mesh quality control

Mesh quality is extremely important for numerical accuracy and efficiency. In this paper, mesh quality refers to the shapes and sizes of mesh elements. Later on, also the goodness of fit to prescribed external and internal boundary curves will be considered. In this chapter, I will present some notation and definitions regarding meshes used in this paper. Then the influence of element shapes and sizes on the numerical simulation will be introduced.

2-1 Notation

Notations used in this paper are listed below:

t: the triangle (2D) or tetrahedral (3D) element in a mesh. In this paper, we only consider the triangle in 2D, which is composed of vertices $v_i\,(i = 1, 2, 3)$ and edges $e_i\,(i = 1, 2, 3)$.

$l_{min}, l_{med}, l_{max}$: the minimum, median and maximum edge lengths of an element t.

di: the diameter of the incircle or insphere of t.

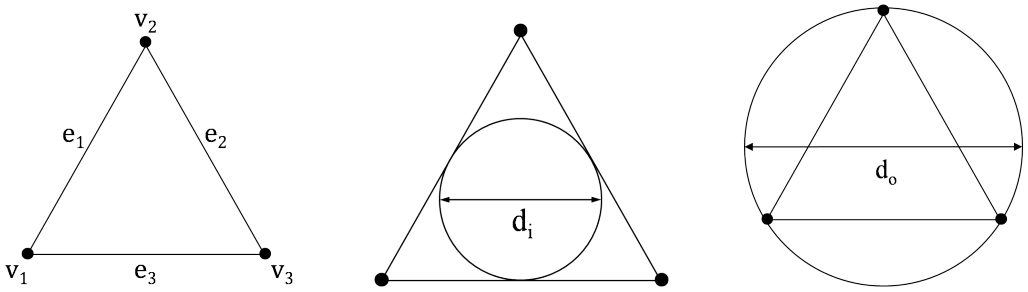
do: the diameter of the circumcircle or circumsphere of t.

a: the area of the triangle element t.

Figure 2-1 illustrates some of the above quantities.

2-2 Element shapes

In terms of mesh generation in a 2D domain, two kinds of element shapes can be used separately or simultaneously: quadrilateral and triangle. To fit the irregular topography and the interfaces of a medium, triangle-shaped elements have a better performance. The shape quality of both the quadrilateral and the triangle is vital to the finite-element simulation. Poor element shapes will lead to convergence issues as well as inaccuracy of the final results.



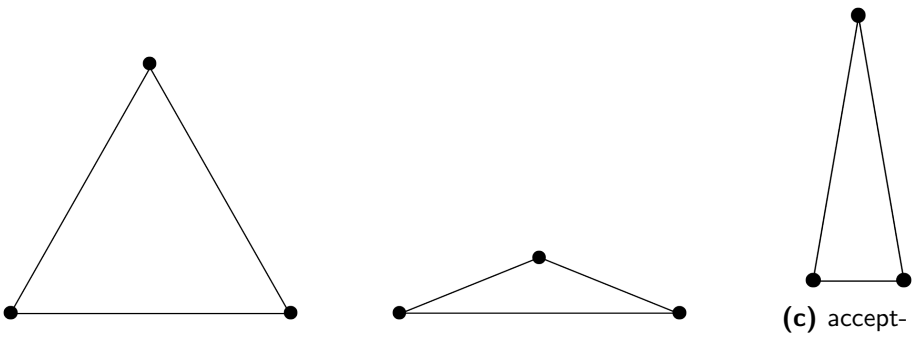
(a) triangle **(b)** incircle **(c)** circumcircle

Figure 2-1: Triangle quantities

A considerably large number of element shape measures, including Jacobian ratio, aspect ratio, corner angle, etc., have been applied to evaluate the shape quality according to different applications. In (Shewchuk, 2002), the mathematical connections of different quality measures to the conditioning of finite-element stiffness matrices and the accuracy of linear interpolation of functions and their gradients are evaluated in detail.

According to engineering experience, equilateral elements are desired for mesh generation, and elements with both too large and too small angles should be avoided. But the mathematical support behind this experience is insufficiently known by most people who are involved in mesh generation. Besides, this experience does not always hold if the focus of the application varies. In some circumstances, elements with too large or too small angles should not be considered as bad elements. Based on the research of Shewchuk (2002), element shapes have different sensitivities to the interpolation error, the gradient error, and the discretization error and the condition number of the stiffness matrix if finite-element methods are applied.

The interpolation error refers to the difference between the true function $f(p)$, which is continuously defined over a mesh, and the interpolated function $g(p)$, which is a piecewise linear approximation to $f(p)$. For triangles, this error is bounded by $c\, l_{max}^2/6$, where c is the curvature bound of f, or in other words the maximum magnitude of the directional second derivative of f. Therefore, the shapes of elements only slightly influence the interpolation error. In other words, elements with large or small angles cannot be regarded as bad elements if the primary purpose is interpolation with regards to the true function (see Figure 2-2).



(a) ideal **(b)** acceptable **(c)** acceptable

Figure 2-2: The influence of shapes on the interpolation error is small. Elements with both too large and too small angles can be accepted.

The gradient error is the difference between the gradient of $f(p)$ and the gradient of $g(p)$. For the wave equation, we are more interested in ∇f (particle velocity of the wave) compared to f (displacement of the wave). Therefore, the gradient error is more important compared with the interpolation error for certain applications. The upper bound of the gradient error can be written as $3\,c\,l_{max}\,l_{med}\,l_{min}\,/4\,A$. If a triangle element has a large angle that approaches 180°, the area a will be close to zero. But the lengths of all three edges do not change very much. As a result, the bound of the gradient error will increase considerably as the angle grows towards 180°. Nevertheless, as a tiny angle decreases towards zero, both the area a and the shortest edge l_{min} approach zero at almost the same rate. Therefore, the upper bound of the gradient error will remain nearly unchanged. Namely, elements with too large and too small angles can result in a different performance in terms of the gradient of the interpolated function. Figure 2-3 illustrates the shape qualities regarding the gradient error.

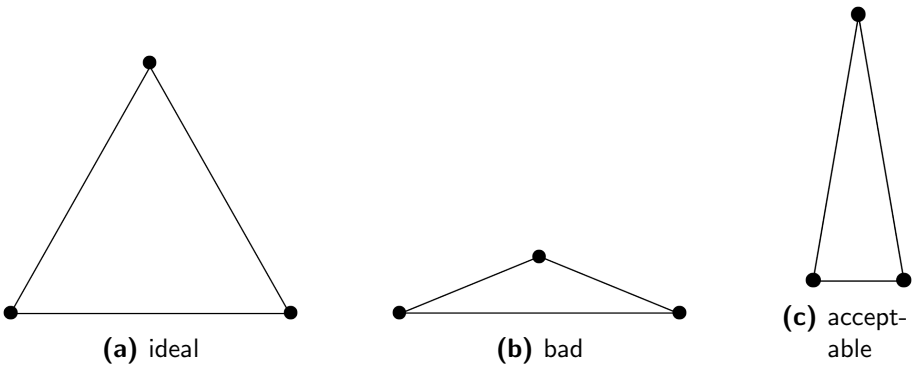


Figure 2-3: The influence of shapes on the gradient error varies. Elements with too large angles can cause large gradient errors. Very small angles hardly affect the gradient error.

In the finite-element method, the global stiffness matrix is assembled from all the element stiffness matrices, each of which is composed of the basis function of that element. The ultimate objective, which is also the difficulty of the finite element method, is to solve the constructed linear system of equations associated with the global stiffness matrix. A critical factor with regard to the stiffness matrix is the corresponding conditioning number $C = \lambda_{max}/\lambda_{min}$, where λ_{max} and λ_{min} are the largest and smallest eigenvalues of the matrix. To solve the linear system of equations, typically two groups of methods exist: iterative methods and direct methods. No matter which methods are applied, their performance is dominated by the conditioning number C of the stiffness matrix. A larger conditioning number can decrease the speed of convergence for iterative methods and increase the size of the round-off error for direct methods. According to Fried (1972), the conditioning number C of the global stiffness matrix is roughly proportional to the largest eigenvalue in terms of all the element stiffness matrices. Shewchuk (2002) shows that if one of the angles of an element approaches zero, the largest eigenvalue of that element will approximate infinity. Therefore, small angles are deleterious to the the matrix conditioning, and equilateral triangles are the preferred element shapes for the condition number of the stiffness matrix (see Figure 2-4).

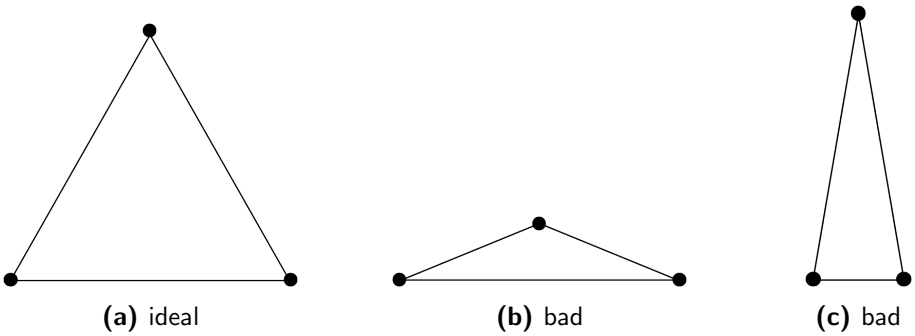


Figure 2-4: The influence of shapes on the conditioning number of the stiffness matrix. Small angles can ruin the matrix conditioning.

2-3 Element sizes

Element size is also a crucial factor for the numerical analysis. Smaller elements that require more nodes in a mesh can increase the accuracy of the numerical results, but the drawback is that the complexity of the geometry and the computation time increase correspondingly. Larger elements can simplify the geometry and save computation time, but at the expense of the numerical accuracy. An example of a rectangular beam test carried out by Gardiner (2017) showed the influence of the number of elements on the vertical deflection and solve time (c.f. Figure 2-5). Reduction of the element size leads to more elements in a mesh. As shown in the figure, a converged solution can be achieved with more elements, but at the expense of more solve time.

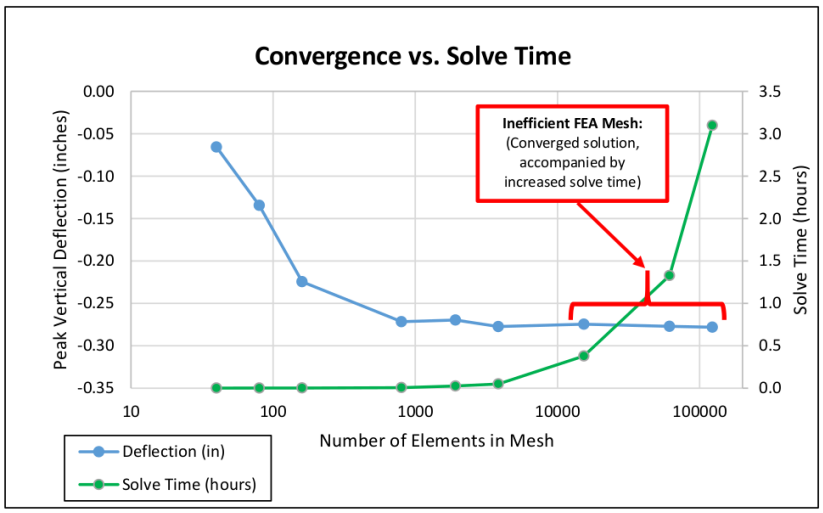


Figure 2-5: The influence of the number of elements on the vertical deflection and solve time for a rectangular beam test.

A large number of research studies (More and Bindu, 2015; Dutt, 2015; Skotny, 2017, e.g.) has been carried out to evaluate the effect of element size on the numerical results based on various models, parameters as well as analysis methods. Their primary objective is to search for a trade-off between the element size and the numerical accuracy. Besides, the accuracy is

not constantly proportional to the number of the nodes. Liu et al. (2011) found that when element size is reduced to a certain magnitude, the changes of the results will remain stable, and no more significant improvements can be expected. Therefore, an optimal element size has to be chosen in order to balance the numerical accuracy and efficiency. It is not necessary and sometimes unphysical to use a super-fine mesh. As long as the results are physical and achieve the desired accuracy, the generated mesh is good enough. One possible solution to the trade-off is to generate a coarse mesh first, and then apply a mesh refinement where finer elements are needed, such as in regions with large deformations, stresses, and instabilities, instead of applying the refinement to the entire domain. There is not a common criterion applicable to all cases. Several factors can be considered when it comes to determine the sizes of the elements: the curvature of the geometry boundary, the local feature size of the geometry, the numerical error estimate, or any user-specified size constraints (Persson, 2006).

Chapter 3

Mesh generation algorithms

In this chapter we will look at some algorithms, including the advancing front methods, the quadtree methods, as well as the Delaunay methods, for the generation of a 2D unstructured mesh. In this thesis, we will use a method based on optimization, inspired by but quite different from that of Alliez et al. (2005).

3-1 Advancing front methods

Advancing front methods are based on the generation of triangles by the propagation of a front towards the interior, starting from the boundary of a domain (Fleischmann, 2000). The initial front is constructed by discretization of the original one with the desired density of the nodes. New vertices are then added progressively in the interior of the domain to form new elements with acceptable shapes and sizes until no active front remains, namely, the entire domain is meshed. Since this mesh generator starts from the discretized boundary, the local mesh density around the boundary can be controlled directly (Ito et al., 2004). In addition, the physical features of the original boundary can be naturally preserved. Figure 3-1 (Bahar, 2001) demonstrates the general procedure of advancing front based methods.

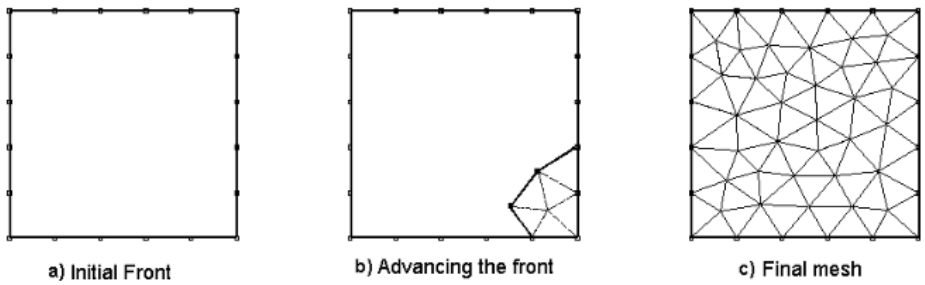


Figure 3-1: Overview of advancing front based methods.

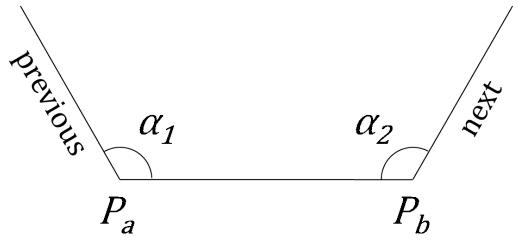


Figure 3-2: A section of the active front

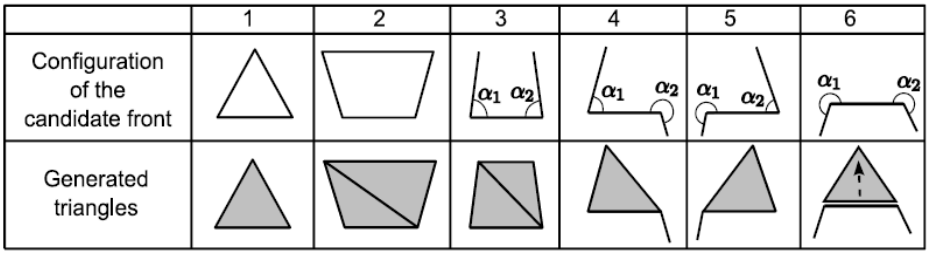


Figure 3-3: The 6 front configurations and their specific triangulation process.

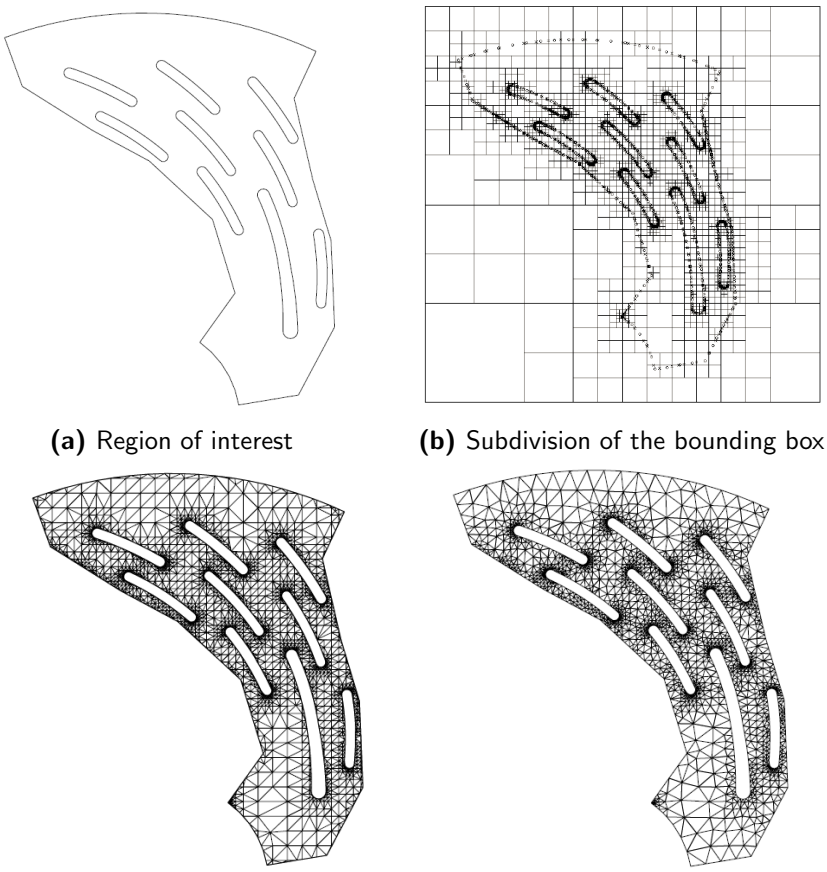
The placement of new vertices is not a trivial task. We consider a front section next to P_aP_b on an active front with two angles α_1 and α_2, which are the angles formed by the current front element P_aP_b and its previous as well as next front elements, respectively (cf. Figure 3-2). The position where the new vertex is placed depends on how big these two angles are. Accordingly, there are six situations in terms of creating a new triangle (Foucault et al., 2008). The candidate front in figure 3-3 refers to the active front that marches towards the interior of the domain.

α_1 and α_2 have to be computed before one of the six configurations is applied. The new triangle will be generated using one or two neighbouring front elements if the front configuration lies agrees with one of the cases between 1 and 5 (case 6 is an exception in that a new vertex should be created instead of connecting the vertices that already exist to form a new triangle). It is realized by comparing α_1 and α_2 with two prescribed threshold angles, such as 85^o and 135^o. If both α_1 and α_2 exceed the larger threshold angle as shown in the 6th configuration, then a new vertex will be created to form the new triangle that has to satisfy the requirements of both the element shape and the element size. Besides, other front elements nearby apart from the neighbouring ones also have to be taken into account in order to avoid their cross-over (triangles overlapping each other) and to create an optimal node location. After the new triangle is generated, the active front is updated by replacing the previous front element with edges of the newly created triangle. The whole process will terminate until the active front is empty, i.e., the entire domain is covered by mesh elements.

3-2 Quadtree methods

The quadtree decomposition, which has a counterpart named octree in 3D, has been used for the purpose of mesh generation for decades since its first implementation by Yerry and Shephard (1983). They presented four reasons why the standard quadtree technique is im-

practical for finite-element analysis and proposed a modified quadtree approach suitable for finite-element mesh generation. The basic concept of quadtree mesh generation algorithm is to divide a square area into smaller squares recursively depending on the geometry of a model, followed by subdivision of the created quadrants into finite elements (triangles, quadrilaterals, or their combination). An optimization is usually applied afterwards to improve the quality of mesh elements (cf. Figure 3-4 (FREY and MARECHAL, 1998)).



(a) Region of interest

(b) Subdivision of the bounding box

(c) Mesh generation based on created quadrants

(d) Mesh optimization

Figure 3-4: The process of quadtree based mesh generation.

The quadtree based method is applied to a bounding box, usually a square, which can enclose a region of interest. This bounding box is considered as the root of the tree. Then a subdivision of the square is performed to create 4 quadrants if the square contains area outside the objective region. The new created quadrants will be tested to see whether they are inside (full), outside (empty), or partially inside (partial) the objective region. If the quadrants are partially inside the region, then subdivisions will be applied to them again. This process will be performed recursively until it reaches a stop criterion, which can be a user-defined resolution.

The quadtree representation of a given region has a clear tree structure (cf. Figure 3-5 (Yang, 1994)). Starting from the root, each quadrant is subdivided into 4 subquadrants or has no subdivision at all. The depth of the tree is denoted by a level number starting from

0 on the root. A leaf of the tree refers to a terminal quadrant that has no subdivisions. Leaves inside the region are the foundation for mesh generation. After each subdivision, four subquadrants will be created with relative positions labeled as TL (top-left), BL (bottom-left), BR (bottom-right) and TR (top-right). Consequently, each quadrant can be easily located following a specific path linked by the root.

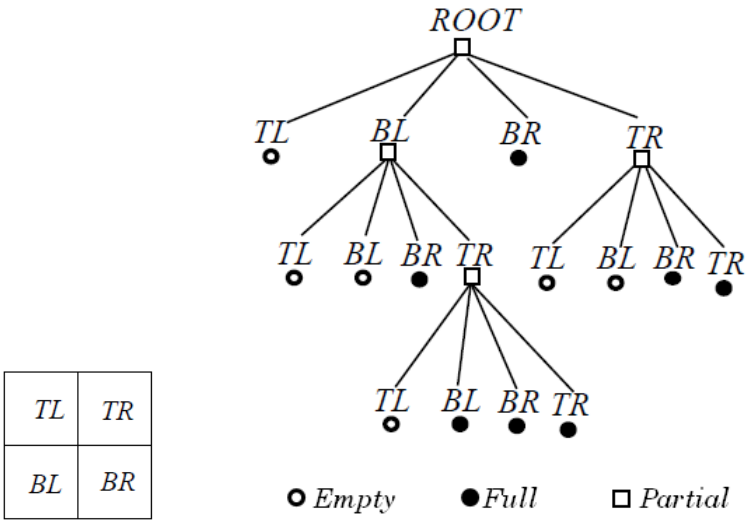


Figure 3-5: The tree structure of a quadtree representation.

Due to the recursive subdivision of the quadrants, the resulting decomposition of the tree might be quite unbalanced, which refers to a large difference of the level between neighbour quadrants. Working on an unbalanced quadtree will result in ill-shaped triangles in the subsequent triangulation stage. In order to overcome this problem, Yerry and Shephard (1983) introduced a balancing condition that is known as the $2:1$ rule: the sizes of any two adjacent quadrants should differ by at most a factor of two. This is a quite useful intermediate step to ensure a good quality of elements in the stage of mesh generation.

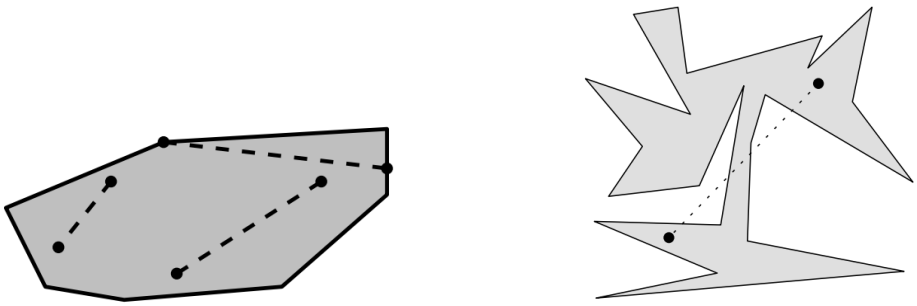
The subsequent operation following quadtree decomposition is finite-element mesh generation based on a balanced quadtree (cf. Figure 3-4c). It is realized by cutting the quadrants to approximate the geometry of a region and triangulating them in a proper way, for instance, with the Delaunay triangulation technique. Mesh optimization (cf. Figure 3-4d) serves as the final stage to improve the mesh quality and obtain a better representation of the region geometry.

3-3 Delaunay methods

The Delaunay triangulation has been used for the purpose of scientific computing for many years since its first introduction by Boris Delaunay in 1934. It has several advantages among all the triangulation methods in terms of mesh quality control, such as avoiding the presence of small angles by maximizing the minimum angle of all the triangles. Besides, there is a lot

of useful functionality that can be implemented for triangulation-based applications including a query point location and adding or removing points in the triangulation.

Before talking about the Delaunay triangulation, let me first introduce the definition of a convex polygon and a convex hull (de Berg et al., 2008). Given a point set P, it is called convex if and only if for any pair of points p_i and p_j belong to P, the connecting line segment of them is also completely contained in P. Otherwise, it is not convex. The polygon shown in Figure 3-6a is convex, while the one shown in Figure 3-6b is not, since parts of the connecting line segment of the indicated two points are outside the polygon. Accordingly, a convex hull for any subset of the plane such as a set of points, a rectangle, or a simple polygon is the smallest convex set that contains that subset, namely, the intersection of all convex sets that contain the subset. Figure 3-7 illustrates three kinds of subset of the plane and their corresponding convex hulls.



(a) A convex polygon **(b)** A non-convex polygon

Figure 3-6: Examples of polygons.

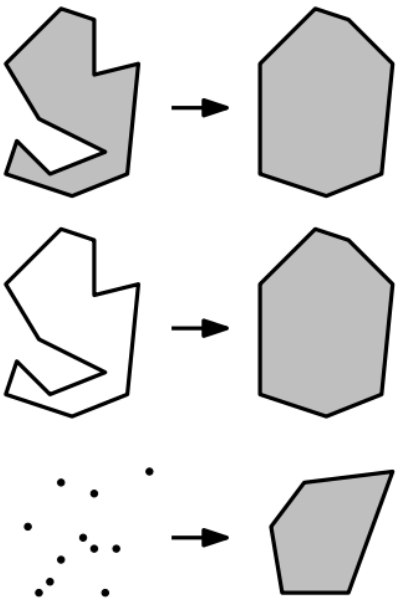
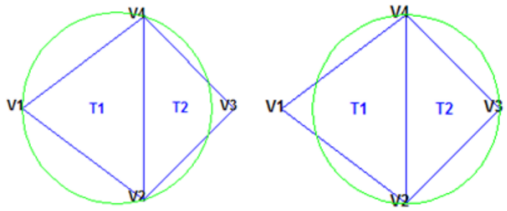
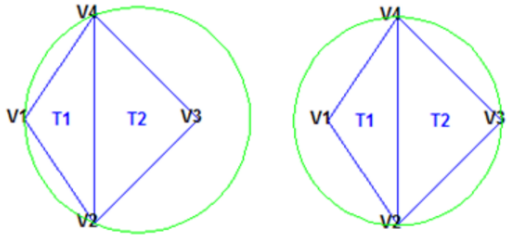


Figure 3-7: Subsets of the plane (left-hand side) and their corresponding convex hulls (right-hand side).



(a) Delaunay triangulation



(b) Non-Delaunay triangulation

Figure 3-8: A triangulation is Delaunay if and only if it satisfies the circumcircle property.

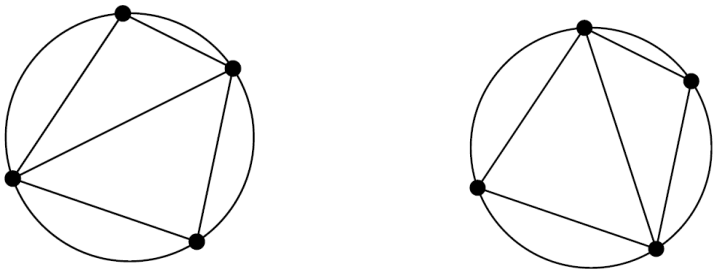


Figure 3-9: Non-uniqueness of the Delaunay triangulation when co-circular points exist.

A Delaunay triangulation of a set of points P is a partition of the convex hull into a certain number of triangles that satisfy the circumcircle property: any circumcircle of the triangles in the convex hull of the point set does not contain a point of P in its interior. To be more precise, their corresponding circumcircles are empty. The triangulation shown in Figure 3-8a is a Delaunay triangulation since the circumcircles of both triangles $T1$ and $T2$ are empty. On the contrary, the triangulation in Figure 3-8b is not Delaunay as the circumcircle of the triangle $T1$ contains a point $V3$ in its interior, and so does that of $T2$.

Every point set that has more than three points has a Delaunay triangulation if not all the points are collinear. However, the uniqueness of the Delaunay triangulation depends on the position of the points. If four or more points are located on a common circle, then there is more than one way to construct a Delaunay triangulation. Both the triangulations of the same point set shown in Figure 3-9 are Delaunay. But the chances that four or more points lie on a common circle is quite small. So we can assume a set of points is in general position where no more than three points are co-circular and the Delaunay triangulation is unique.

In order to know whether a triangulation is Delaunay or not, we have to check if all the

circumcircles are empty according to the circumcircle property. We consider a simple 2D convex hull with four vertices A, B, C, and D (cf. Figure 3-10). To detect where the vertex D lies is equivalent to evaluate the following determinant (Guibas and Stolfi, 1985):

$$det(A,B,C,D) = \begin{vmatrix} x_A & y_A & x_A^2+y_A^2 & 1 \\ x_B & y_B & x_B^2+y_B^2 & 1 \\ x_C & y_C & x_C^2+y_C^2 & 1 \\ x_D & y_D & x_D^2+y_D^2 & 1 \end{vmatrix}$$

If the vertex D is inside the circumcircle of the triangle ABC that is defined in a counter-clockwise order, this determinant should be positive. Otherwise, it should be negative. This can be used as a guideline for the construction of a Delaunay triangulation. We consider the convex hull $ABCD$ again. There are two ways to triangulate this convex hull, adding either the diagonal AC or the diagonal BD. One of them must fulfil the Delaunay condition. Then we can test one case by evaluating its determinant before deciding which configuration we should choose for our Delaunay triangulation construction. This process constitutes the foundation of the flip algorithm for the computation of a Delaunay triangulation. Starting from a random triangulation of a point set, all the edges will be tested according to the circumcircle property. For any one who violates the Delaunay condition, it will be flipped by exchanging it for the other diagonal.

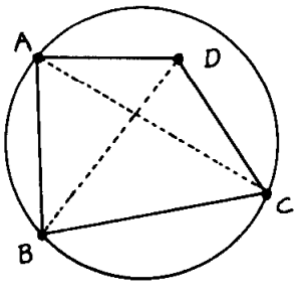


Figure 3-10: An example of a 2D convex hull with four vertices.

Although the performance of the flip algorithm is quite good, it will take a running time of $O(n^2)$ in the worst case, where n is the number of points to be triangulated. In order to improve the computational performance, many other algorithms were proposed and implemented with less running time. The divide-and-conquer algorithm for triangulation was presented by Guibas and Stolfi (1985). To begin with, it successively partitions a set of points into two halves according to their x-coordinates (y-coordinates are considered if they have the same x-coordinate), until subsets with no more than three points are left. Then each subset was processed by Delaunay triangulation and merged with its former half. Once the final two half triangulations are merged, the final Delaunay triangulation for the whole set is completed. The overall running time is expexted to be $O(n\,log\,n)$ (Leach, 1997). The incremental algorithm proposed by Green and Sibson (1977) allows new points to be added or deleted rather than working on the provided points only. Typically, two steps are involved in this algorithm: locating the triangle that contains the new site and updating the new diagram. This algorithm was improved later by Su and Drysdale (1997) so that the overall running time is decreased to $O(n)$ while maintaining its relative simplicity.

Chapter 4

Optimization techniques

Given a physical problem, we want first to identify what our objective is, and figure out how it can be expressed by a mathematical formula before we can solve it using computational tools. This mathematical expression of our objective is called the objective function, which depends on a certain number of variables or unknowns. Optimization refers to the process of finding the values of variables that can maximize or minimize our objective function with or without constraints. In order to optimize the objective function efficiently, both the objective function itself and the optimization algorithm have to be taken into consideration. The objective function should not be neither too simple, lacking some information of the problem we want to solve, nor too complex, requiring lots of computational time and storage. Besides, there are numerous optimization algorithms aimed at distinct objective functions. A proper one has to be chosen among them according to the characteristics of the objective function to be optimized.

4-1 Search methods

As mentioned above, optimization searches for the variable values by which the objective function achieves its minimum value (we only consider minimization in this paper). Generally three categories in terms of search methods exist according to their use of objective function evaluations, gradients of the objective function, and Hessians of the objective function (Kiranyaz et al., 2014).

4-1-1 Gradient based methods

To understand the minimization using gradients of the objective function, we can start with a simple example. Suppose you are on the top of a mountain with a lake on its foot. Your aim is to reach the lake under the condition that you are blindfolded. So you have no visibility to see where you are headed. Under such a circumstance, the best approach that can guide you to the lake is to follow the slope of the hill downwards.

In the mathematical point of view, gradient methods can be explained using Taylor's theorem. Consider the optimization problem

$$\underset{x \in \mathbf{R}^n}{\text{minimize}}\, f(x),$$

where $f : \mathbf{R}^n \to \mathbf{R}$ is differentiable and convex. By introducing Taylor series expansion, we can obtain the first order approximation of the objective function at the location $x_k + \Delta x_k$ if we know the value at the point x_k:

$$f(x_k + \Delta x_k) = f(x_k) + \nabla f(x_k)^T \Delta x_k. \tag{4-1}$$

We want to find a smaller value than $f(x_k)$ by moving a step Δx_k along a certain direction, namely, $f(x_k + \Delta x_k) < f(x_k)$. Then it requires that the second term on the right-hand side of (4-1) should be negative. Δx_k could be either negative or positive. Therefore, the sign of Δx_k should be opposite to that of the gradient of $f(x_k)$. If this condition is satisfied, $f(x_k)$ will descend as we move along the direction of $-\nabla f(x_k)$, and we can reach the minimum of $f(x)$ at the end.

The simplest first-order algorithm based on this theorem is the gradient descent method, using only the objective function $f(x)$ and its gradient $\nabla f(x)$. The minimum of $f(x)$ can be found following the direction of the negative gradient of $f(x)$, along which $f(x)$ decreases fastest. But how far we should move for each step along this direction still remains unknown. Actually, the step size dominates the efficiency of the entire optimization process. Starting from an initial point x_0, the gradient descent method applies the following updates for each iteration until a stop criterion is fulfilled:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{4-2}$$

where α_k is the step size, which is a positive real number. α_k could be either a constant value or adapted to the iteration automatically by applying a line search, which minimizes $f(x_k - \alpha_k \nabla f(x_k))$. The step size has a significant influence on the performance of the iteration. If it is too small, the convergence will be very slow, while if it is too large, divergence may occur, and the minimum can never be reached. The rate of convergence for the gradient descent method is linear (Nocedal and Wright, 1999):

$$\|s_{k+1} - s^*\| \leq r \|s_k - s^*\|, \tag{4-3}$$

if k is sufficiently large for a sequence of numbers s_k that converges to some limit s^*, where r is a constant between 0 and 1.

The gradients or derivatives can be computed either analytically or numerically. If the objective function has a simple expression and the derivatives are easy to calculate by hand or a computer algebra system, then users can provide the code to compute them. A derivative check with the numerical approximation can be applied to ensure the correctness of the results. However, if the function is quite complicated, calculating the derivatives analytically could be infeasible. In this case, the finite difference approximation of the derivatives or other differentiation methods (Nocedal and Wright, 1999) can provide the possibility to compute them automatically.

4-1-2 Hessian based methods

Methods involving only the first partial derivatives are quite simple in terms of programming and requires less computation, but the rate of convergence is relatively slow. If the second derivative or Hessian is included, the rate of convergence will be improved significantly. Suppose $f(x)$ is twice differentiable and convex, the second order Taylor series expansion of $f(x_k + \Delta x_k)$ is:

$$f(x_k + \Delta x_k) = f(x_k) + \nabla f(x_k)^T \Delta x_k + \frac{1}{2} \Delta x_k^T \nabla^2 f(x_k) \Delta x_k := m(\Delta x_k). \tag{4-4}$$

We are trying to looking for a direction for Δx_k that can minimize $m(\Delta x_k)$. As we know, the gradient is zero at the minimum of a function. According to this, we set the derivative of $m(\Delta x_k)$ zero, and can obtain:

$$\Delta x_k = - \left[\nabla^2 f(x_k)\right]^{-1} \nabla f(x_k). \tag{4-5}$$

The term on the right-hand side of 4-5 is called the Newton direction. By taking the step size α_k into account we can compute the successive points through the following iteration:

$$x_{k+1} = x_k - \alpha_k \left[\nabla^2 f(x_k)\right]^{-1} f(x_k). \tag{4-6}$$

Different from the gradient descent method, the Newton's method usually uses a unit step size ($\alpha = 1$). Only when the obtained minimum of f is not satisfactory will α be adjusted by a line search. The rate of convergence for the Newton's method is very fast or quadratic if we are sufficiently close to the minimum and the functional is convex and sufficiently smooth. If a sequence of numbers s_k converges to some limit s^* at a quadratic speed, these numbers should satisfy (Nocedal and Wright, 1999):

$$\|s_{k+1} - s^*\| \leq M \|s_k - s^*\|^2, \tag{4-7}$$

for all k that is sufficiently large, where M is a positive constant. Compared to the gradient descent method with a linear rate of convergence, the rate of convergence of Newton's method is improved significantly.

Although the second-order optimization converges fast to the minimum of f, there are some limitations. The major one is that this method requires a convex and twice differentiable function. This is to make sure that a positive semi-definite Hessian, which satisfies $\nabla^2 f(x)$, is available. Otherwise, the Newton direction will not exist, or is not a descent direction any more. Being convex is to ensure that a global minimum can be reached rather than being stuck in a local minimum. Another limitation is that the Hessian matrix has to be computed at each iteration, which could be computationally expensive. Therefore, the pure Newton method is not suitable for applications in higher dimensions.

In order to combine the computational efficiency of the first-order methods and the fast rate of convergence of the second-order methods, a class of quasi-Newton methods, such as the BFGS method, the DFP method, and the SR1 method, are designed based on the regular Newton method. Instead of computing the Hessian matrix explicitly, the quasi-Newton methods construct an approximation to the Hessian matrix using the gradients from the previous

iterations. Different quasi-Newton methods have different ways to update this approximation for each iteration with the following formula:

$$H_{k+1} = H_k + U_k, \tag{4-8}$$

where U_k is the term to update H_{k+1}, and differs in various quasi-Newton methods. But all the approximated matrices should preserve some essential features of the Hessian matrix, such as being symmetric and positive-definite. Besides, the quasi-Newton condition should be satisfied:

$$H_{k+1}\,\Delta x_k = y_k, \tag{4-9}$$

where H_{k+1} is the updated estimate of the Hessian matrix, $\Delta x_k := x_{k+1} - x_k$ and $y_k := \nabla f(x_{k+1}) - \nabla f(x_k)$. Since the quasi-Newton methods preserve the structure of the Newton method and avoid the computation of the second derivatives, it is preferred in the majority of the optimization problems. Typically, the quasi-Newton methods converge at a superlinear rate, meaning:

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0. \tag{4-10}$$

4-1-3 Objective function value based methods

Optimizers using either the gradient only or both the gradient and the Hessian are efficient in finding the local minimum of convex objective functions. But if the gradients do not exist, such as in the case of discontinuous functions, or is so expensive that their evaluation is impractical, the gradient-based methods will be inefficient or useless at all. Then, derivative-free methods that only evaluate the values of the objective function itself can be applied to deal with these problems. There is a long history of the development for the gradient-free algorithms. Various methods have been designed and improved (Rios and Sahinidis, 2013), such as the direct local method, Nelder-Mead Simplex method, and the DIvided RECTangles (DIRECT) Method, which is a global optimizer.

The Nelder-Mead Simplex method is a classical direct search method, and is among the first generation of the derivative-free methods in 1960s. This algorithm works with a simplex, which is formed by $n+1$ points in n-dimensional space (Hicken and Alonso, 2012). Specifically, in 2-dimensional space, the simplex is a simple triangle. The first step is to generate a simplex starting from an initial point by adding n new extra points. Then the values of the objective function at each vertex of this simplex will be evaluated. The vertex that yields the worst result (with the highest value) will be replaced by a new point, which can be obtained through a series of operations: reflection, expansion, outside contraction, inside contraction and shrinking. The algorithm will be terminated if the size of the simplex is small enough, or the objective function values at the vertices are close enough to the minimum.

To avoid being trapped in a local minimum, a global optimizer has to be implemented using a systematic search of the design space. Such optimizers involve partitioning the search space into subsets (Rios and Sahinidis, 2013), such as the DIRECT method and the multilevel coordinate search (MCS) method. Although these algorithms can converge to the minimum globally, this convergence is based on a large and exhaustive search over the entire domain.

4-2 Minimization solvers in Matlab

In Matlab, the Optimization Toolbox provides solvers for various kinds of optimization problems, such as minimization, for multi-objective problems, objective functions that are a sum of squares, and so on. Since the objective function in our problem is formulated as a scalar that should be minimized, we focus on the minimization solver, or minimizer of this toolbox.

`fminsearch` (MathWorks, 2017a) is a non-linear minimization solver designed for the unconstrained objective functions using the derivative-free methods—the Nelder-Mead simplex algorithm. The objective function $f(x)$ returns a scalar, while the variables x can be a vector or matrix. The full syntax of `fminsearch` is:

```
[x,fval,exitflag,output] = fminsearch(fun,x0,options)
```
(4-11)

where the input and output arguments are defined as below:

`fun`: the name or handle of the objective function to be minimized, a real scalar
`x0`: the initial point specified as a real vector or real array
`options`: a structure of the optimization options, which can define whether to display during the process or not and what kinds of information to be displayed, the maximum number of function evaluations and iterations allowed, the termination tolerance on f and x, and so on
`x`: the solution, which can be a real vector or real array with the same size as that of `x0`
`fval`: the value of the objective function at the solution
`exitflag`: the reason why the solver was terminated
`output`: a structure indicating the information of the optimization process, including the number of iterations and function evaluations, algorithm used, etc.

Since `fminsearch` uses the simplex algorithm, which is a derivative-free method to locate the minimum, it can be applied to discontinuous functions. But only the local minimum will be found, unless the function is strictly convex. Besides, `fminsearch` only accepts real numbers of f and x. Complex values have to be split into real and imaginary parts before they can be processed by `fminsearch`.

If the objective function is differentiable, `fminunc` minimizer (MathWorks, 2017b) can be applied with the first or second order derivative-based methods. And it is more efficient than `fminsearch`, especially when the dimension of the problem is greater than two. The syntax to call `fminunc` is more or less the same as that to be used for `fminsearch`, except the information of the gradient and the Hessian:

```
[x,fval,exitflag,output,grad,hessian] = fminunc(fun,x0,options)
```
(4-12)

where the arguments `options`, `grad`, and `hessian` are defined:

`options`: a structure of the optimization options. In addition to the options that can be defined in `fminsearch`, it can also define what kinds of optimization algorithms are used, whether the gradients are provided by users or approximated in a finite-difference way, and so on
`grad`: a real vector of the gradient at the solution
`hessian`: a real matrix of the approximated Hessian at the solution.

Two algorithms can be chosen for `fminunc`: trust-region and quasi-Newton algorithms. If the trust-region algorithm is performed, the users have to provide the gradient in the objective function. The option `CheckGradients` can be used to compare the user-supplied derivatives to the finite-difference approximations in order to confirm the correctness of the gradients. If the gradients can be provided by users, the performance of `fminunc` will be improved significantly. Similarly, the user-supplied Hessian for the trust-region algorithm can speed up the convergence rate of `fminunc`. Typically, the Hessian is approximated using the finite differences.

minFunc is a Matlab minimizer designed for unconstrained optimization problems with a real-valued differentiable function by Schmidt in 2005. The latest version was released in 2013. The way to call minFunc is almost the same as that of calling `fminunc`. It involves a lot of algorithms, including both the first-derivative and second-derivative methods, to compute the descent direction. The quasi-Newton method with the limited-memory BFGS updating is chosen for the default setting. The step size is determined by three line search strategies: the Armijo line search, the Wolfe line search, and the line search from the Matlab Optimization Toolbox. Compared to `fminunc`, minFunc requires less function evaluations to converge, and can process a larger number of variables. Because of those advantages and its compatibility with Matlab Optimization Toolbox, minFunc is used to optimize our objective function in this project.

Part II

Application

Chapter 5

Formulation of the problem

As mentioned in the beginning, meshes that fit the topography and interfaces of the subsurface impedance model are required for the numerical analysis of the seismic wave propagation, especially for the finite-element analysis. In this project, the generation of a good mesh that can meet our requirements is formulated as an optimization problem. Given a velocity model of the subsurface medium, we pursue a qualified mesh that can be generated by our program. In this chapter, I will present the way to solve our optimization problem step by step using Matlab.

5-1 Generation of the initial mesh

To start with, we synthesized a simple 2D P-wave velocity model with a relatively smooth topography, which is considered as the curve we are trying to match the mesh with. We want our element sizes to be proportional to the synthetic velocity field. In order to realize that, the velocity field is scaled by a reference number $scaleref$ such that the scaled values can be applied to control the element sizes. The reference number $scaleref$ is chosen according to the rule that *at least three elements can be accommodated per wavelength*. Then, the velocity field can be scaled by this reference number and the P-wave velocity in the sea water. The scaled velocity field is denoted as scl, defined as:

$$scl = scaleref\ vp_{ij} \,/\, 1500, \qquad i = 1 \dots n,\ j = 1 \dots m, \tag{5-1}$$

where vp is the synthetic P-wave velocity in the xz-space. Figure 5-1 illustrates the scaled velocity field including a closed curve. The top of the curve represents the topography of the geological body, while the other three boundaries are created for the closure of the curve to make sense geologically.

Given the velocity field, we will generate our initial mesh to represent this domain. During the process of the optimization, the mesh can change in both shape and size. In order to ensure that the closed curve is always inside the mesh, we add extra points to the exterior

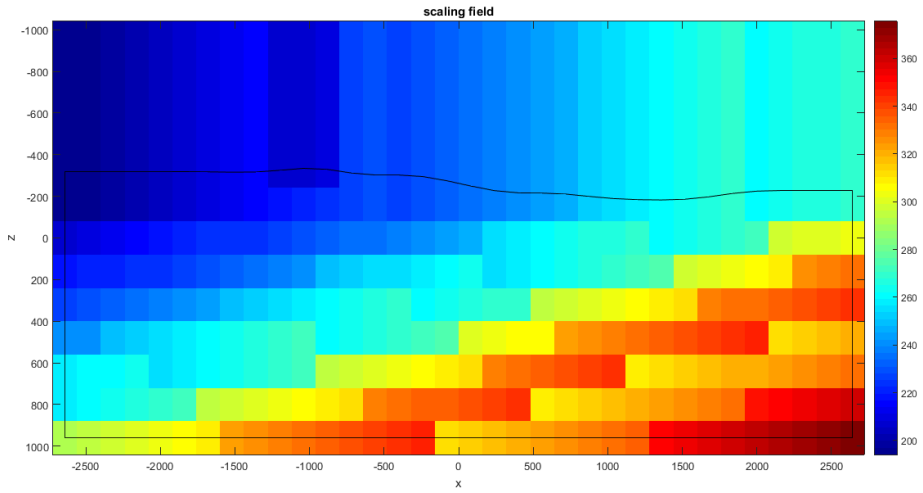


Figure 5-1: The synthetic velocity field scaled by a reference number, together with the closed curve.

of this domain to extend the velocity field by the extrapolation. When the optimization is finished, we can remove those extra elements. The algorithm we used to generate the mesh is MESH2D, which is designed by Engwirda in 2006 for the first release. It starts with a quadtree decomposition of a domain before applying the triangulation. This algorithm preserves the features of the Delaunay Triangulation, which avoids the presence of elements with small angles. Besides, the element sizes can be specified by users. We used the scaled velocity values *scl* as the constraints of the element sizes. Figure 5-2 shows the generated mesh, the initial mesh, obtained by MESH2D for the extended domain.

5-2 Objective functions and corresponding gradients

Our goals are: (1) to maintain the elements in good shapes, (2) to enable the elements to reflect the given velocity information, (3) and to let the mesh fit the topography and the interfaces of the medium. In order to achieve these goals, we created an objective model f, which can represent those requirements mathematically. Then we can deal with it through the optimization. For our optimization problem, three objective functions (f_1, f_2, f_3) are combined together in terms of element sizes, element shapes, as well as the areas between the curves and the edges. In order to maintain a trade-off between those three objective functions, the weights are needed for each term to adjust their contributions.

5-2-1 Element sizes

The first objective function f_1 includes the information of di to control the element sizes, which should be proportional to the scaled velocity field. The purpose is to avoid a dense distribution of the elements near some features, such as the vicinity of an interface between two layers. We consider a triangle element t with three vertices $v_i(x_i, z_i)$, $i = 1, 2, 3$. The area

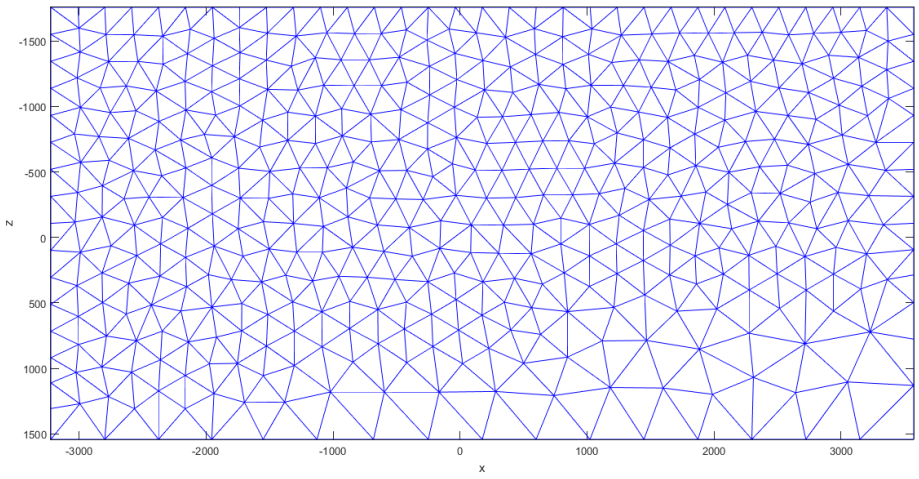


Figure 5-2: The initial mesh of the extended domain.

s_t of the triangle can be obtained by the following determinant:

$$s_t = \pm \frac{1}{2} \begin{vmatrix} x_1 & z_1 & 1 \\ x_2 & z_2 & 1 \\ x_3 & z_3 & 1 \end{vmatrix}, \tag{5-2}$$

where the plus/minus is meant to guarantee a positive value of the area. The diameter di of the inscribed circle of the triangle is obtained by:

$$di = \frac{4\, s_t}{h_{21} + h_{32} + h_{13}} \tag{5-3}$$

where h_{ij} (i, j denote the index of the vertex) represent the lengths of three edges, which can be calculated by:

$$h_{21} = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}\,; \tag{5-4a}$$

$$h_{32} = \sqrt{(x_3 - x_2)^2 + (z_3 - z_2)^2}\,; \tag{5-4b}$$

$$h_{13} = \sqrt{(x_1 - x_3)^2 + (z_1 - z_3)^2}\,. \tag{5-4c}$$

We want the element sizes to be proportional to the corresponding scaled velocity values. Namely, di should be equal to scl for the same triangle. To represent the triangle better, we take the value of scl at the centroid of t. Supposing we have n_t triangles in the mesh, the first objective model f_1 is constructed in the following way (l_2-norm for a smooth function):

$$f_1 = \sqrt{\frac{1}{n_t} \sum_{i=1}^{n_t} |di_i / scl_i - 1|^2}\,. \tag{5-5}$$

The derivatives of f_1 with respect to x_i and z_i can be obtained by differentiation by parts. To simplify the problem, we assume scl is constant within the triangle, and has no relation

with the changes of the vertices. Let gi denote the derivative of di: $gi = (\partial di/\partial x_i, \partial di/\partial z_i)$, $i = 1, 2, 3$. Then the derivatives of f_1, g_1, can be written as:

$$g_1 = \frac{\sum_{i=1}^{n_t} gi_i \, scl_i \, |\frac{di_i}{scl_i} - 1|}{\sqrt{n_t \sum_{i=1}^{n_t} |\frac{di_i}{scl_i} - 1|^2}} \,. \tag{5-6}$$

g_1 has a dimension of $n_p \times 2$, where n_p is the number of the points in the mesh. In order to be used by minFunc, g_1 has to be rearranged as a column vector.

5-2-2 Element shapes

We have already mentioned that the ideal element shape in 2D is an equilateral triangle. One important feature of it is that the ratio of its incircle and circumcircle diameters is 0.5, which we can use as a constraint of the shape to enable the elements to approach an equilateral triangle. The diameter do of the circumcircle of t can be obtained by the following formula:

$$do = \frac{h_{21} \, h_{32} \, h_{13}}{2 \, s_t} \,. \tag{5-7}$$

Based on the feature of an equilateral triangle, the objective function f_2 for the element shapes is designed by the combination of the diameters of both the incircle and the circumcircle of t:

$$f_2 = \sqrt{\frac{1}{n_t} \sum_{i=1}^{n_t} |1 - \frac{2 \, di_i}{do_i}|^2} \,. \tag{5-8}$$

The purpose of using the l_2-form is to ensure a smooth and differentiable function for the subsequent optimization.

Similarly, the derivatives of f_2 with respect to x_i and z_i can also be obtained by differentiation by parts. Let go denote the the derivative of do: $go = (\partial do/\partial x_i, \partial do/\partial z_i)$, $i = 1, 2, 3$. The derivatives, g_2, of f_2 can be calculated by:

$$g_2 = \frac{\sum_{i=1}^{n_t} gg_i \, |1 - \frac{2 \, di_i}{do_i}|}{\sqrt{n_t \sum_{i=1}^{n_t} |1 - \frac{2 \, di_i}{do_i}|^2}} \,, \tag{5-9}$$

where gg represent the derivatives of the formula $1 - 2 \, d_i/d_o$:

$$gg = 2 \, \frac{di \, go - do \, gi}{do^2} \,. \tag{5-10}$$

The dimension of g_2 is $n_p \times 2$, where n_p is the number of the points in the mesh. g_2 also has to be rearranged to a column vector such that the algorithm minFunc can use it.

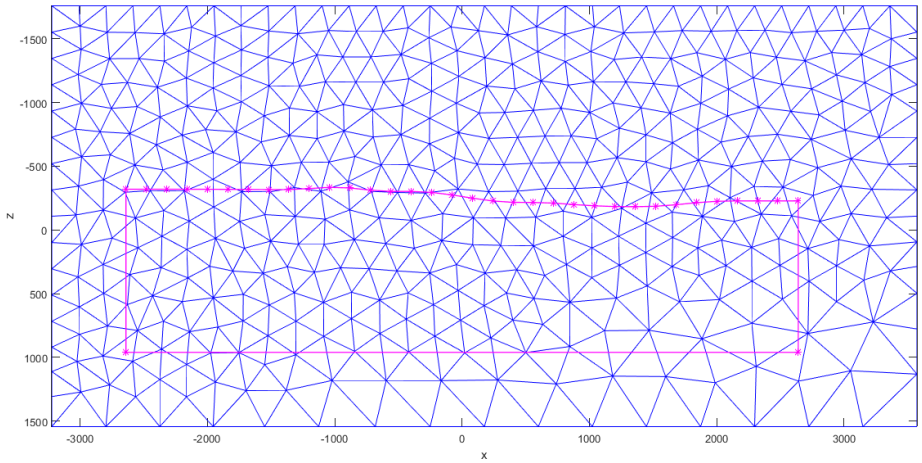


Figure 5-3: Overview of the curve and the mesh.

5-2-3 Areas between curves and edges

The last but not least aim is to enable the elements to follow the curves, including the topography and interfaces in a domain, such that the curves coincide with the edges of the elements. To express it in mathematical terms, we can try to minimize the areas between the curves and the edges. Therefore, we have to locate the points that can define a polygon between the curves and the edges. Afterwards, the areas and the corresponding gradients will be derived for the optimization.

Consider the model, including the triangulation TR and the curve, we constructed in Figure 5-3. First, we have to extract the triangles that contain the points pc on the curve with the help of the Matlab function tsearchn. We denote these triangles tc. We notice that the triangles tc do not cover all the triangles that the closed curve crosses. So we have to start with tc to locate all the triangles $tall$ that contain a section of the curve.

Suppose there are some triangles (one or more) between two points, $xlast$ and $xcurr$, on the curve (cf. Figure 5-4). The triangles enclosing these two points are denoted as $tlast$ and $tcurr$, and we can find their indices with tsearchn. We want to locate the other three triangles between the two points. We only consider $tlast$ and the two points for the first step. The intersection point $xedge$ of the edge and the connecting line between $xlast$ and $xcurr$ can be obtained by solving the equation:

$$(1 - mu)\, xv_1 + mu\, xv_2 = (1 - la)\, xlast + la\, xcurr\,, \tag{5-11}$$

where xv_1 and xv_2 are the vertices of $tlast$ in a clockwise order; mu and la are two partition coefficients of the edge and the connecting line of the points, respectively. With the partition coefficients, we can get $xedge(x, z)$ via either the left-hand or the right-hand side of Equation 5-11. Then we replace $xlast$ by a new point, which is shifted from $xedge$ towards $xcurr$ by adding a relatively small number to $xedge$:

$$xlast = xedge + (xedge - xlast)\, 10^{-12}\,. \tag{5-12}$$

This is to ensure the new $xlast$ to fall in the next triangle, which is considered to be the new $tlast$. The procedures above iterate until the stopping criterion, $tlast = tcurr$, is reached.

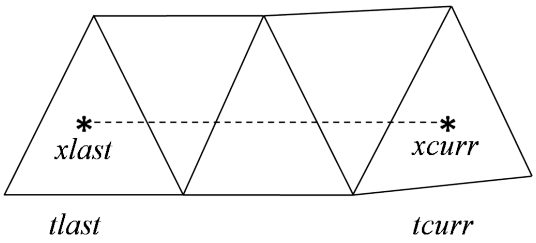


Figure 5-4: Locate all the triangles between two points on the curve.

We have already located all the triangles *tall* that the curve crosses. The next step is to construct a polygon to represent the gap between the curve and the edges of *tall* before we obtain our third objective function and its gradients. Figure 5-5 illustrates five cases when a curve intersects with a triangle. In order to construct a proper polygon, we chose the vertex that is closest to the intersection point *xedge*. So for the cases 1 to 4, the polygon consists of the points on the curve (pc_1, pc_2, pc_3), and the vertices close to pc_1 and pc_3. In case 3, pc_1 and pc_3 share a common point as the closest vertex. Case 0 is an exception, which could occur at the corner of the curve. In this case, the polygon includes only one vertex that is closest to pc_2 besides pc_1 and pc_2. Note that pc_1 acts both the first and the last point of the polygon to keep it being closed. For example, the polygon in case 1 of Figure 5-5 is $pc_1\, pc_2\, pc_3\, v_3\, v_1\, pc_1$, consisting of 6 points with pc_1 counted twice for closure. The absolute value of the polygon area *pa* can be calculated by:

$$pa = \pm\frac{1}{2}\sum_{i=1}^{n}(x_i\, z_{i+1} - x_{i+1}\, z_i) = \pm\frac{1}{2}\sum_{i=1}^{n}(x_i - x_{i+1})(z_{i+1} + z_i)\,, \tag{5-13}$$

where n is the number of points constituting the polygon, excluding the one repeated for closure (5 for case 1 in Figure 5-5). Note that point number $n+1$ needs to be set to point 1 and point 0 to n. Accordingly, the derivatives of *pa* with respect to x and z for each point are:

$$gp_i(x,z) = \pm\frac{1}{2}\,(z_{i+1} - z_{i-1},\, x_{i-1} - x_{i+1}) \qquad i = 1, 2, \ldots, n\,, \tag{5-14}$$

To generate the third objective function, we applied the square of the area instead of using the area directly. The reason to do this will be discussed later. The expression of the third objective function is give below:

$$f_3(x,z) = \frac{1}{2}\sum_{j=1}^{n} pa_j(x,z)^2\,, \tag{5-15}$$

where n is the length of *tall*. Then the derivatives of f_3 with respect to x and z follow simply from the chain rule.

$$\frac{\partial pa}{\partial x_i} = \pm\frac{1}{2}\,(z_{i+1} - z_{i-1}), \quad \frac{\partial pa}{\partial z_i} = \pm\frac{1}{2}\,(x_{i-1} - x_{i+1}), \tag{5-16}$$

we obtain

$$\frac{\partial f_3}{\partial x_k} = \sum_{j=1}^{n} pa_j\frac{\partial pa_j}{\partial x_k}, \quad \frac{\partial f_3}{\partial z_k} = \sum_{j=1}^{n} pa_j\frac{\partial pa_j}{\partial z_k}. \tag{5-17}$$

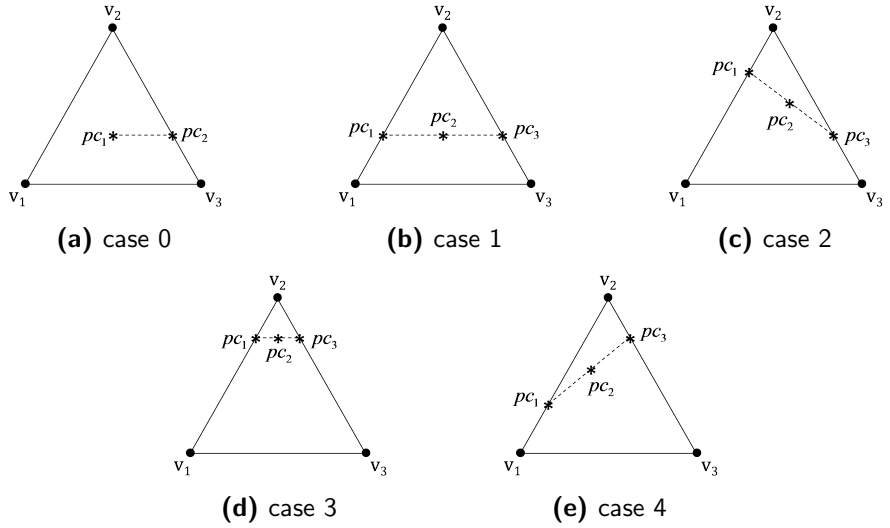


Figure 5-5: Different ways for a curve to intersect with a triangle.

Here, most of the contributions are zero. For the derivatives, we are only interested in those vertices that are the variables in our problem. We take case 1 in Figure 5-5 as an example to analyse the derivatives. The polygon used for the area minimization is $pc_1pc_2pc_3v_3v_1pc_1$. So the area of this polygon is a function of those five points. Apparently, only the vertices v_1 and v_3 contribute to the area. But the vertex v_2 is also related to the area due to its contribution to the location of the intersection points pc_1 and pc_3. If we take the derivatives at all these three vertices into consideration, things would become quite complicated. Therefore, for simplicity, we ignored the influence of the vertex v_2, and only used the derivatives of the vertices v_1 and v_3 which constitute the polygon. Other cases are processed in such a way that the impact of the vertices, which are outside the polygon, on the area is neglected. So the remaining task is to extract the derivatives of the vertices involved in the polygon, followed by assigning the correct indices to them.

Now let us consider the consequence of this approximation to the derivatives if we use the area itself as the third objective function. Due to the approximation made in the gradient computation, the derivatives may not be zero when the area is zero, which is undesirable. Instead, we want a zero derivative for a zero area. Otherwise, the vertices will keep moving when the area reaches the minimum. This problem can be solved by introducing the squares of the areas. The resulting derivative g_3 is a multiplication of the area and its derivatives. Then, g_3 vanishes when the area approaches zero.

The final step of constructing our objective function is to add those three separate objective functions using three weights. The total objective function and its gradient become

$$f = w_1 f_1 + w_2 f_2 + w_3 f_3 , \tag{5-18a}$$

$$g = w_1 g_1 + w_2 g_2 + w_3 g_3 , \tag{5-18b}$$

where w_1, w_2, and w_3 are the corresponding weights, which can be adjusted to balance the contributions of each separate objective functions and their gradient. Note that f is a scalar, while g is a column vector with a length of twice the number of points in the mesh.

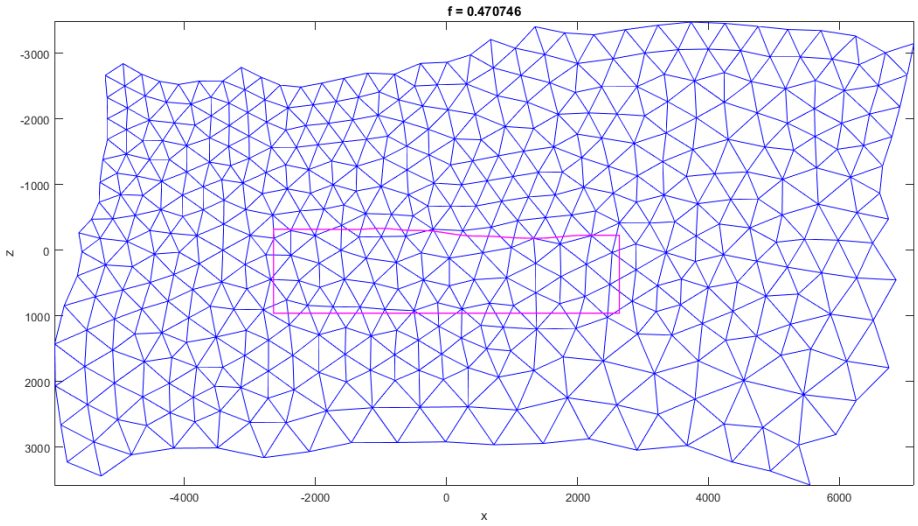


Figure 5-6: Optimization of f_1 and f_2 without bounds.

5-3 Optimization using minFunc

Having constructed the objective function and the gradients, we will subsequently carry out the optimization using minFunc (version 2012). Based on the initial points in the mesh shown in Figure 5-3, the mesh will adjust itself such that the objective function can reach its minimum. In order to increase the performance and the efficiency, we split the whole process of optimization into two steps. The objective functions f_1 and f_2 are first optimized prior to the full optimization. This can be realized by modifying the weights.

By setting the third weight w_3 to zero, f_3 will be neglected and only f_1 and f_2 are optimized. During the optimization, the whole mesh will deform (cf. Figure 5-6) to find the minimum objective function. This leads to a smaller density of elements inside the curve as well as a poorly shaped mesh. To avoid this situation, bounds are applied to the boundaries by constructing a mask, which has a value of one for the boundary points. Then, the gradients are multiplied by one minus this mask. After this procedure, the boundary points will remain fixed during the optimization (cf. Figure 5-7). The price paid is that the resulting objective function has a larger value than before.

The next stage of the optimization is to take f_3 into consideration, starting with the mesh generated in the first step. Since the order of magnitude of f_3 is much larger than that of the other terms, the weight w_3 has to be modified beforehand such that f_3 can be reduced to a size comparable with f_1 and f_2. One solution is to extract $ordf3$, which is the order of magnitude of f_3, followed by dividing w_3 by $ordf3$. $ordf3$ can be extracted in the following way:

$$ordf3 = floor(log(abs(f_3))/log(10))\,, \tag{5-19}$$

where $floor$, log, and abs follow Matlab syntax. Then, the modified weight w_{3n} is obtained by:

$$w_{3n} = w_3\, 10^{-ordf3}\,. \tag{5-20}$$

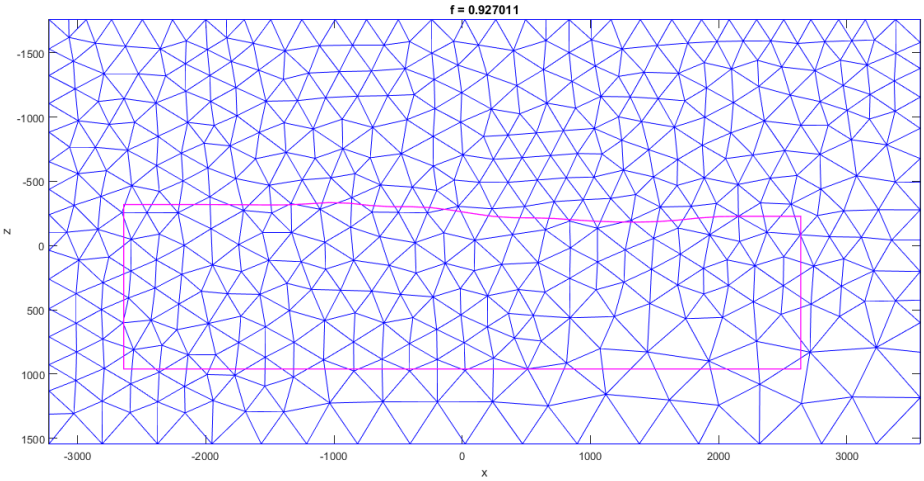


Figure 5-7: Optimization of f_1 and f_2 with bounds.

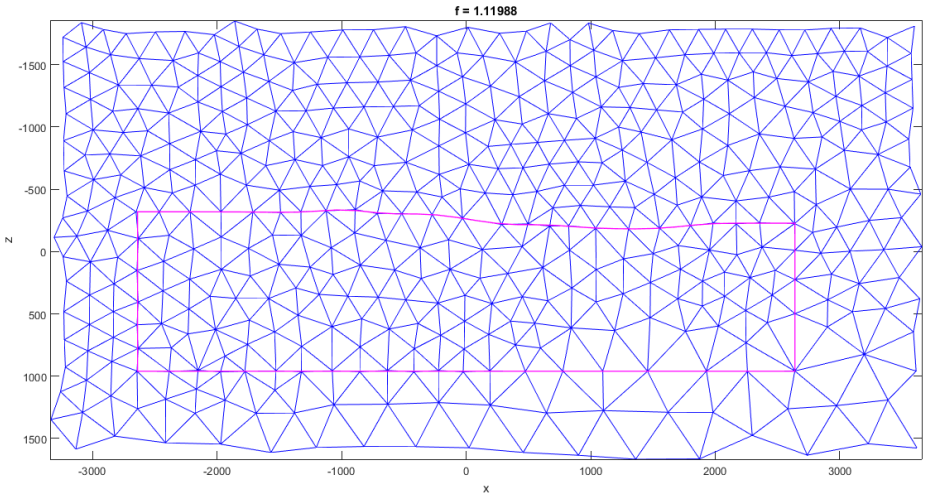


Figure 5-8: Optimization of f_1, f_2 and f_3 simultaneously without bounds.

If this modification is not enough, *ordf*3 can be adjusted manually by adding or subtracting a certain number. In order to give enough freedom to the optimization, boundary bounds are not applied during this stage. Figure 5-8 illustrates the result after the optimization. Although the mesh deforms a little bit, the shape and size relative to the curve are still acceptable. We can observe that the mesh fits the curve quite well, with reasonable element shapes and sizes. Finally, we trimmed the mesh by removing the triangles outside the boundaries (the closed curve) to form our final mesh (cf. Figure 5-9).

5-4 More complicated cases

In order to see the performance in more complicated cases, we synthesized another model named model-2 (cf. Figure 5-10) with more curves to be optimized. Because of the smaller distances between the curves, we decrease the scale to obtain a denser mesh. Figure 5-11

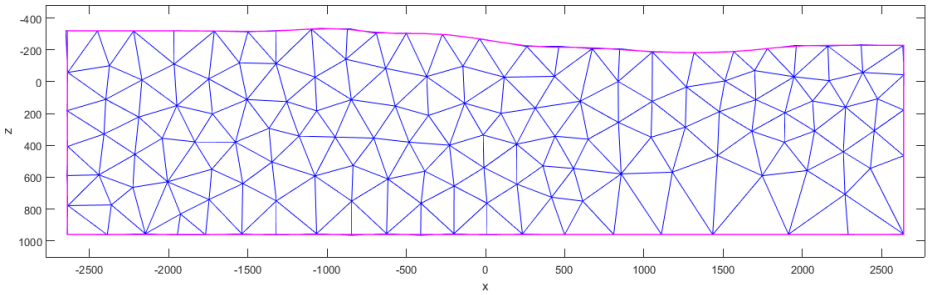


Figure 5-9: Final mesh after removing the elements outside the boundaries.

displays the initial mesh generated according to the velocity field, together with the curves. The mesh after the optimization is shown in Figure 5-12 using the weights of 1, 15, and 1.1. We can observe that the curves are followed quite well, but the quality of the mesh is not very good. Figure 5-13 displays another version of the mesh generated by changing the third weight to 1. The quality of the mesh is improved a lot, while the curves are not completely followed by the mesh. Therefore, in order to obtain a satisfied mesh, the weights have to be selected properly.

Figure 5-14 illustrates the mesh generated according to the first velocity model with a zigzag curve. The curve has a lot of sharp corners that are much smaller than the size of the elements in the mesh. We see the mesh does not follow the trend of the curve well, because the mesh tries to fit every small section of the curve. In the physical world, if the dimension of some features, which we are not interested in, on the earth surface is much smaller than that of the seismic wavelength, it will cause problems to simulate the propagation of the seismic waves. For the same model without these sharp corners, the curve is followed very well (cf. Figure 5-15). Therefore, the curves can be smoothed in advance using the Matlab function `smooth` if we want to remove the sharp corners. More examples can be found in the Appendix.

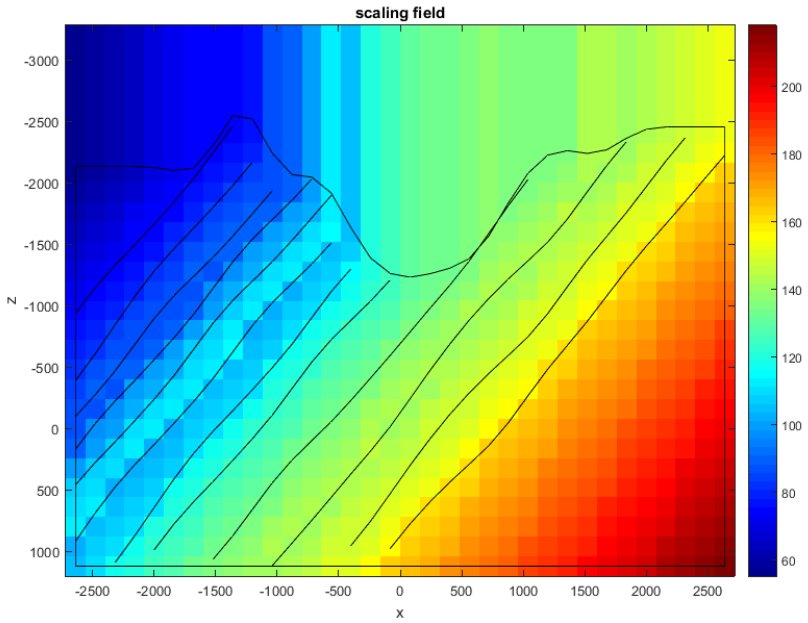


Figure 5-10: The synthetic velocity field and a set of curves of model-2.

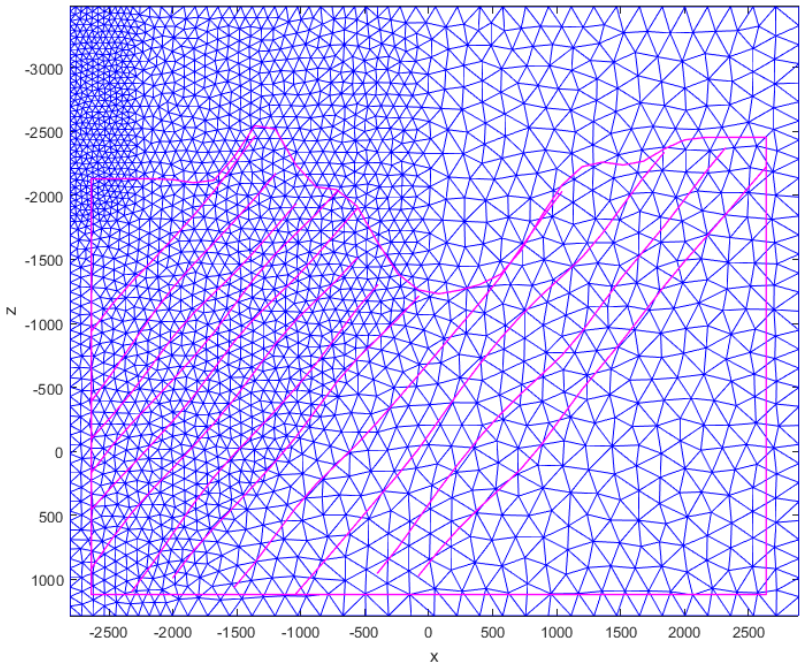


Figure 5-11: The initial mesh and curves for model-2.

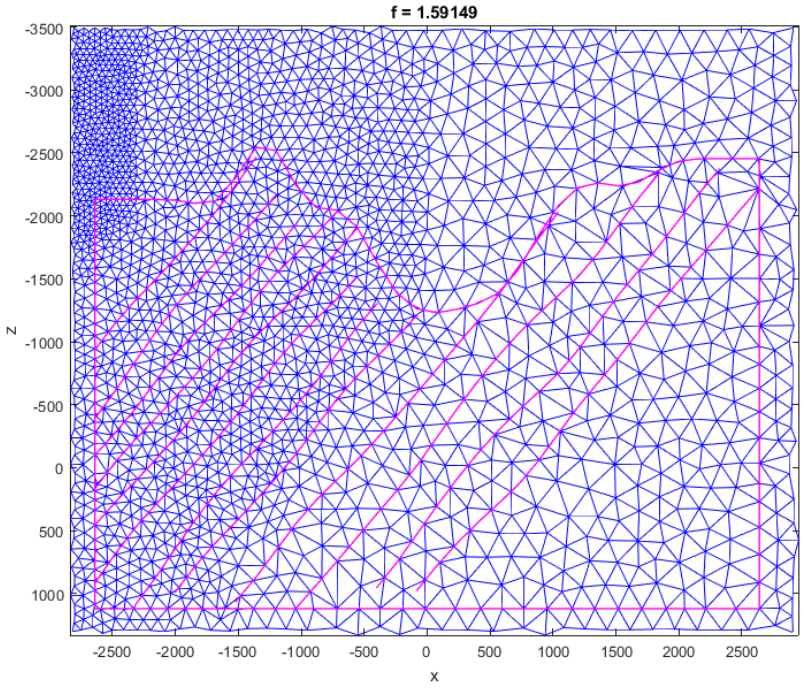


Figure 5-12: The mesh after optimization using weights of 1, 15, and 1.1 for model-2.

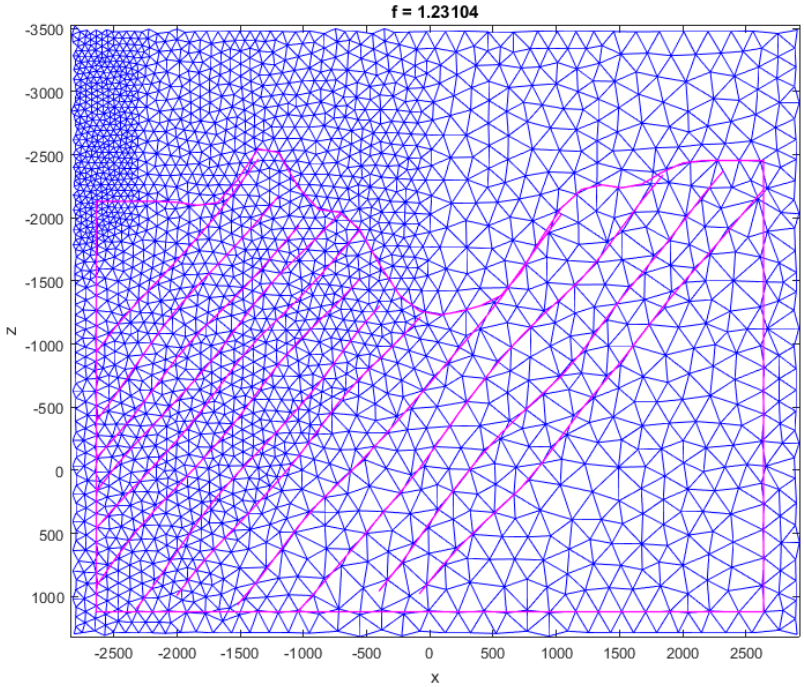


Figure 5-13: The mesh after optimization using weights of 1, 15, and 1 for model-2.

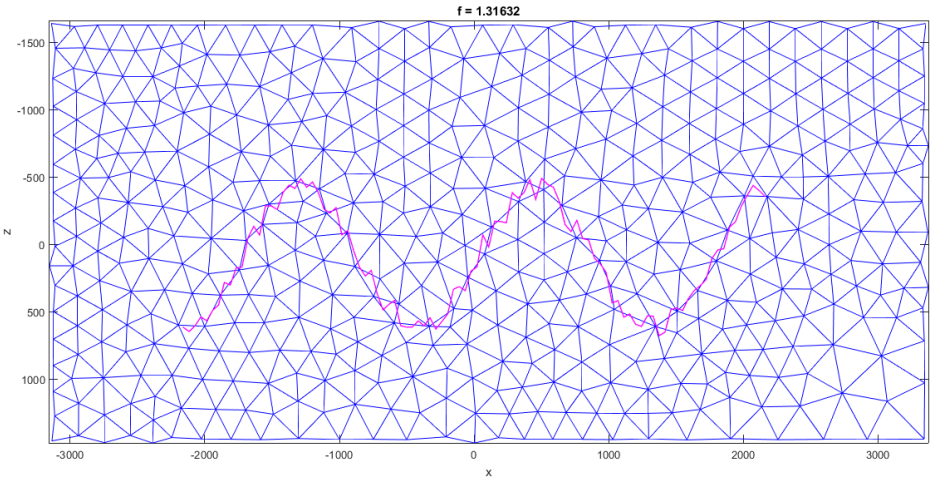


Figure 5-14: The mesh generated for the model with a zigzag curve.

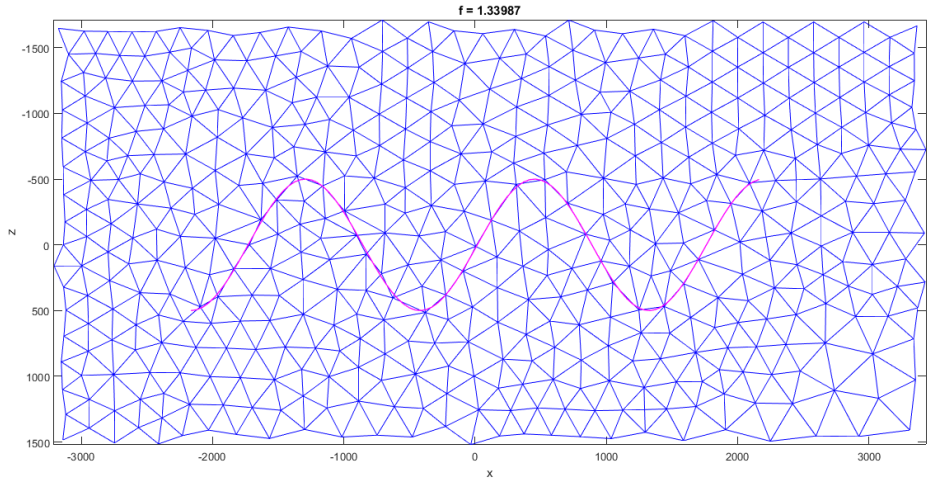


Figure 5-15: The mesh generated for the model with a smooth curve.

Chapter 6

Discussions and conclusions

For the generation of the initial mesh, we applied the generator MESH2D. We found the generated initial mesh represents the velocity field quite well due to the input of the sizing constraints. In addition to MESH2D, we also tried the Matlab function `delaunayTriangulation`. But the results are not satisfactory, since during the generation of the initial mesh, the information of the scaling field is not included. This poses a bigger challenge to the subsequent optimization. Besides, the mesh will deform substantially during the optimization of f_1 and f_2 if the boundaries are not fixed, although f_1 and f_2 can be well optimized in this case. If the boundaries are set to be fixed, the optimization of f_1 will fail. Namely, the sizes of the elements are not proportional to the velocity field. Therefore, MESH2D provides a better initial mesh, which makes the whole optimization more efficient.

For g_3, we only considered the vertices that constitute the polygon. Actually, the other vertex or vertices of the triangle, which the curve crosses, also has or have contributions to the gradients of the polygon area. This is due to the fact that the edge points are located using the vertices both in and outside the polygon. For simplicity, we ignored the influences of the vertex or vertices outside the polygon on the g_3. This process leads to an approximation to the gradients of f_3. So during the optimization of f_3, only one or two vertices of a triangle can move. This reduces the freedom, and the triangles will degenerate. But if the weights can be chosen properly, the optimization of f_2 can compensate for this effect to some degree.

If the the wavelength of a seismic wave is much larger than the variations of the topography and interfaces, these small variations cannot be recognized in the seismic data. Therefore, we expect our mesh to ignore these details of the topography and interfaces during the optimization of f_3. Otherwise, the mesh will attempt to fit every detail of the curve, which is impossible due to the relatively large element size. Therefore, a smoothing process is applied before the optimization is carried out. The smoothing can be applied to the whole curve or just a section of it where the smoothing is required, such as in areas with large wave speeds.

The sizes of the elements are proportional to the local velocity in the mesh. The exact size of an element is controlled by a reference number, $scaleref$. We choose it according to the rule that at least three elements can be accommodated per wavelength, which should

provide sufficient modelling accuracy. We have seen that both the rate of convergence and the computing time depend on the element sizes. Besides, the performance of the curve fitting for the generated mesh also relies on the element sizes, especially when multiple curves exist and the gap between those curves is small compared to the element size. Therefore, the reference number $scaleref$ has to be selected appropriately to balance between the desired numerical accuracy and the running time as well as the curve fitting.

Bibliography

Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M. (2005). Variational Tetrahedral Meshing. *ACM Transactions on Graphics*, pages 617–625.

Bahar, C. (2001). Advancing Front Method. http://www.ae.metu.edu.tr/~cengiz/Eng/index.html.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag.

Dutt, A. (2015). Effect of Mesh Size on Finite Element Analysis of Beam. *SSRG International Journal of Mechanical Engineering*, pages 8–10.

Engwirda, D. (2017). MESH2D - Delaunay-based unstructured mesh-generation. https://nl.mathworks.com/matlabcentral/fileexchange/25555-mesh2d-delaunay-based-unstructured-mesh-generation.

Fleischmann, P. (2000). Advancing Front Methods. http://www.iue.tuwien.ac.at/phd/fleischmann/node39.html.

Foucault, G., Cuillière J-C., François, V., Léon J-C., and Maranzana, R. (2008). An Extension of the Advancing Front Method to Composite Geometry. In *Proceedings of the 16th International Meshing Roundtable*, pages 287–314. Springer, Berlin, Heidelberg.

FREY, P. J. and MARECHAL, L. (1998). Fast Adaptive Quadtree Mesh Generation. In *Proceedings of the 7th International Meshing Roundtable*, pages 211–224. Sandia National Laboratories.

Fried, I. (1972). Condition of finite element matrices generated from nonuniform meshes. *AIAA Journal*, pages 219–221.

Gardiner, J. (2017). Finite Element Analysis Convergence and Mesh Independence. https://www.xceed-eng.com/finite-element-analysis-convergence-and-mesh-independence/.

Green, P. J. and Sibson, R. (1977). Computing Dirichlet Tessellations in the Plane. *The Computer Journal*, pages 168–173.

Guibas, L. and Stolfi, J. (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transactions on Graphics*, pages 74–123.

Hicken, J. E. and Alonso, J. J. (2012). Chapter 6: Gradient-Free Optimization. http://adl.stanford.edu/aa222/Lecture_Notes_files/chapter6_gradfree.pdf.

Ito, Y., Shih, A. M., and Soni, B. K. (2004). Reliable Isotropic Tetrahedral Mesh Generation Based on an Advancing Front Method. In *Proceedings, 13th International Meshing Roundtable*, pages 95–106. Sandia National Laboratories.

Kiranyaz, S., Ince, T., and Gabbouj, M. (2014). *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*. Springer Berlin Heidelberg.

Leach, G. (1997). Improving Worst-Case Optimal Delaunay Triangulation Algorithms. http://goanna.cs.rmit.edu.au/~gl/research/comp_geom/delaunay/paper_short.pdf.

Liu, W., Geni, M., and Yu, L. (2011). Effect of Mesh Size of Finite Element Analysis in Modal Analysis for Periodic Symmetric Struts Support. *Key Engineering Materials*, pages 1008–1012.

MathWorks (2017a). fminsearch. https://nl.mathworks.com/help/optim/ug/fminsearch.html.

MathWorks (2017b). fminunc. https://nl.mathworks.com/help/optim/ug/fminunc.html.

More, S. T. and Bindu, R. S. (2015). Effect of Mesh Size on Finite Element Analysis of Plate Structure. *International Journal of Engineering Science and Innovative Technology*, pages 181–185.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer-Verlag.

Persson, P.-O. (2006). Unstructured Mesh Generation. https://persson.berkeley.edu/pub/persson06unstructured.pdf.

Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, pages 1247–1293.

Schmidt, M. (2005). minFunc: unconstrained differentiable multivariate optimization in Matlab. http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html.

Shewchuk, J. R. (2002). What is a Good Linear Element? Interpolation, Conditioning, and Quality Measures. *Proc. of 11th International Meshing Roundtable*, pages 115–126.

Skotny, Ł. (2017). Correct mesh size - a quick guide. https://enterfea.com/correct-mesh-size-quick-guide/.

Su, P. and Drysdale, R. L. S. (1997). A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry*, pages 361–385.

Yang, D. D. X. (1994). *MESH GENERATION AND INFORMATION MODEL FOR DEVICE SIMULATION*. PhD thesis, STANFORD UNIVERSITY, http://www-tcad.stanford.edu/tcad/pubs/theses/danyang/.

Yerry, M. A. and Shephard, M. S. (1983). A Modified Quadtree Approach To Finite Element Mesh Generation. *IEEE Computer Graphics and Applications*, pages 39–46.

Zhebel, E., Minisini, S., Kononov, A., and Mulder, W. A. (2014). A comparison of continuous mass-lumped finite elements with finite differences for 3-D wave propagation. *Geophysical Prospecting*, 62:1111–1125.

Appendix A

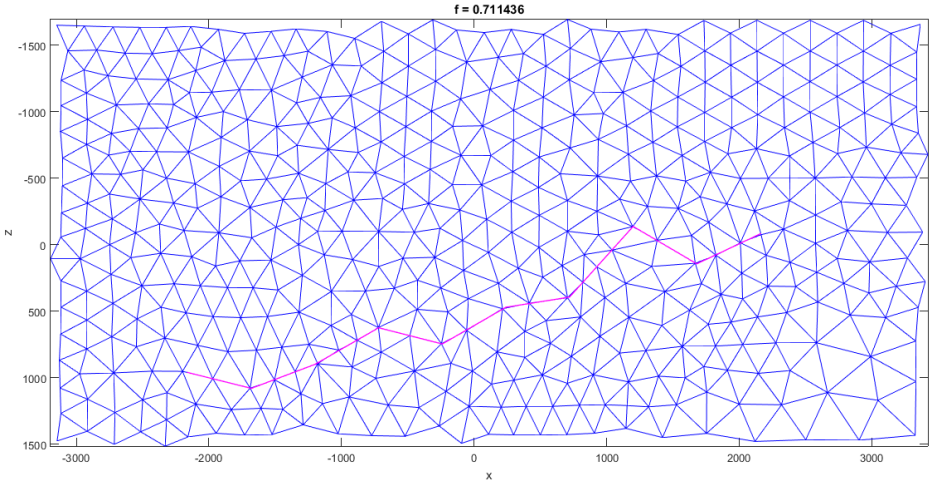
More examples

A-1 Smoothness of the curve

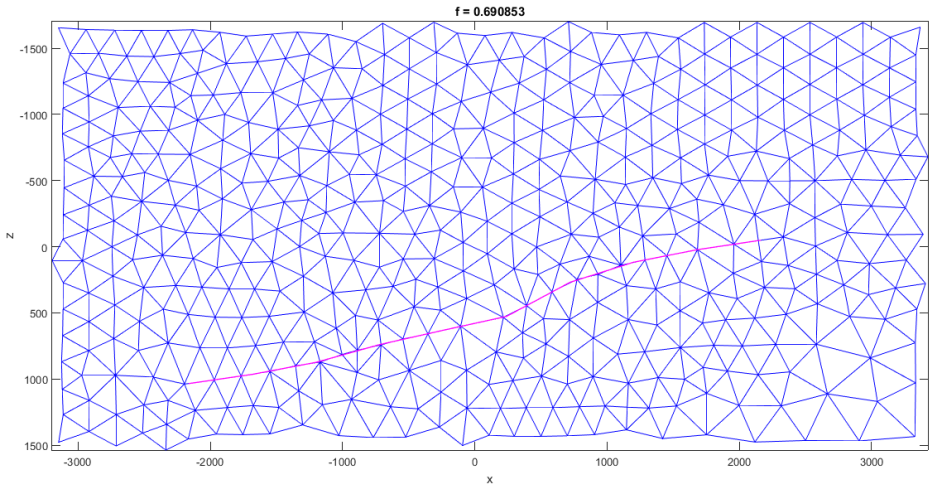
Figures A-1 to A-5 depict meshes generated for models with different curves with and without smoothing. If the corners of the curve are not so sharp, and the distances between corners are not very short compared to the element size, the curve can still be well fit by the mesh despite of its non-smoothness (c.f. Figure A-1a). But as the corners become sharper and closer to each other, the mesh cannot follow the curve anymore, and the quality of elements decreases. Therefore, smoothing of the curve should be applied before the optimization under this condition.

A-2 Elements sizes

Figure A-6 illustrates the influence of the reference number $scaleref$ and the corresponding element sizes on the performance of curve fitting as well as on the mesh quality. Although the curve can still be followed as the reference number $scaleref$ or the element size increases, the quality of the elements is becoming worse.

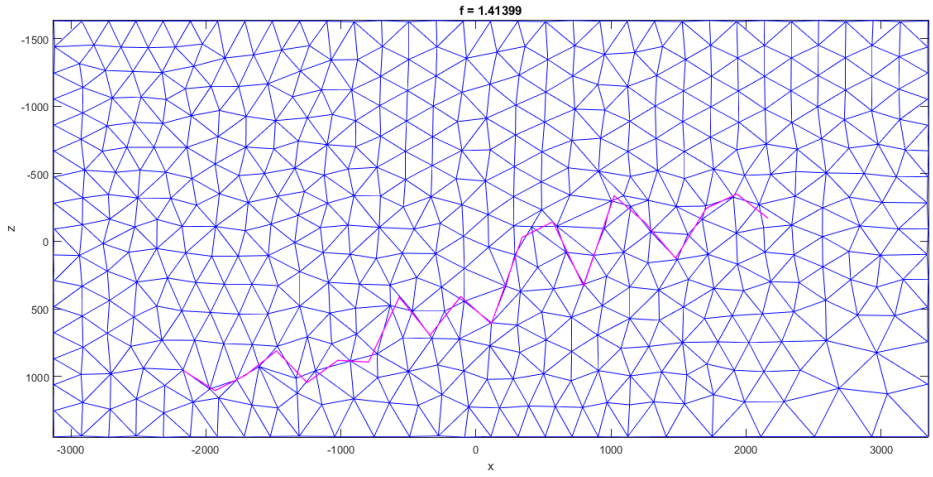


(a) Without curve smoothing

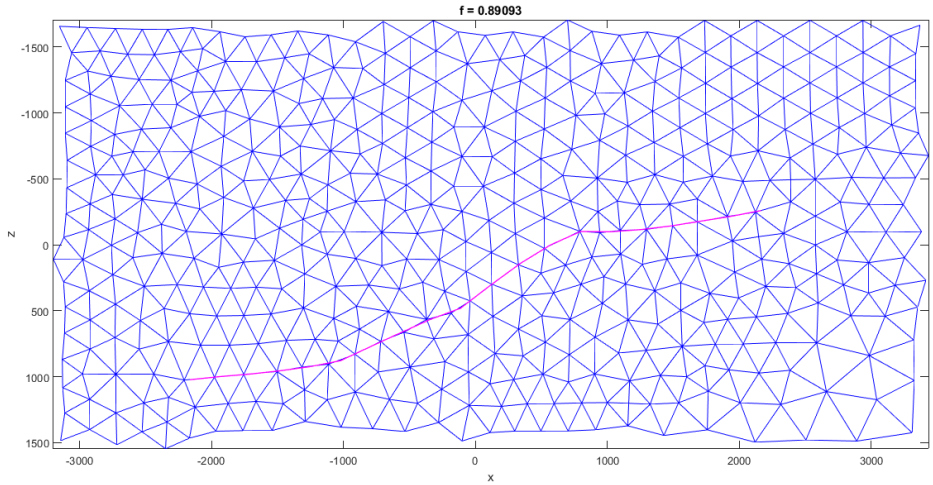


(b) With curve smoothing

Figure A-1: The meshes generated for the model with curve-1 before and after smoothing applied.

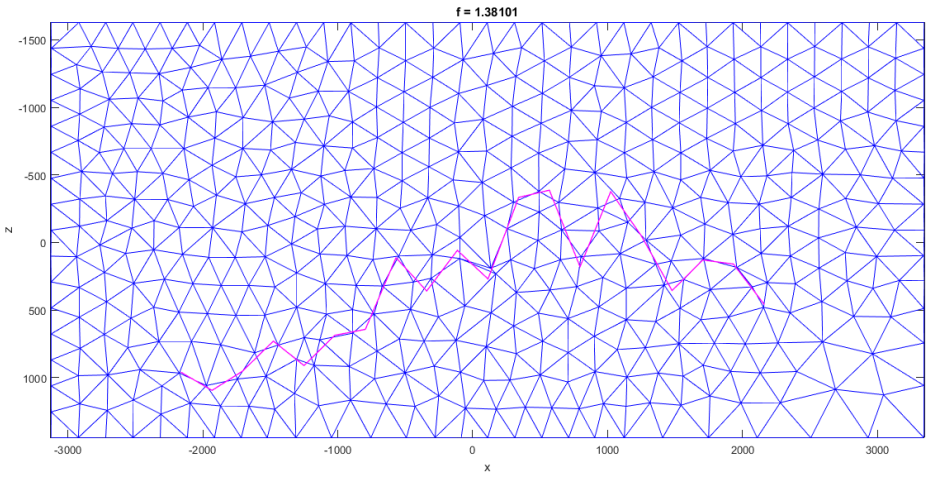


(a) Without curve smoothing

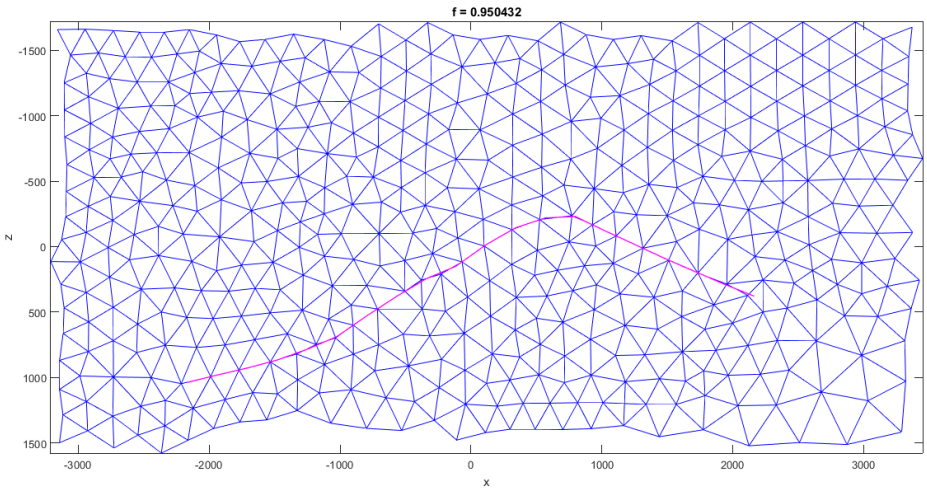


(b) With curve smoothing

Figure A-2: The meshes generated for the model with curve-2 before and after smoothing applied.

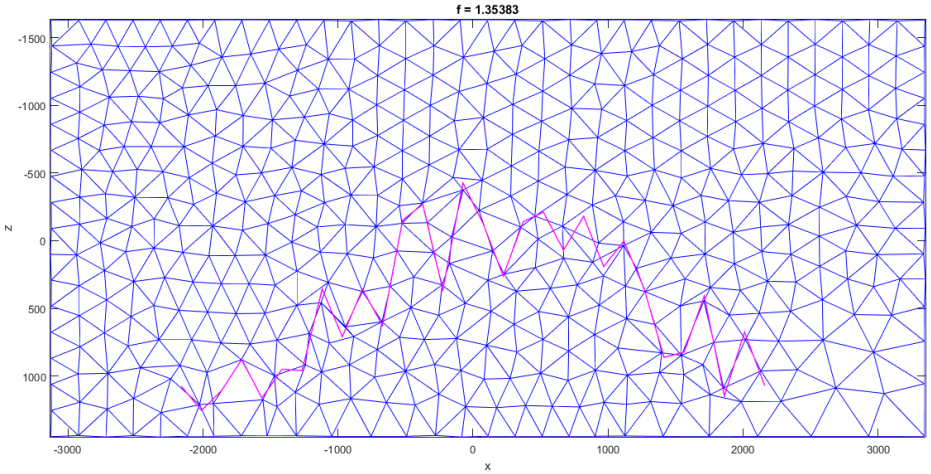


(a) Without curve smoothing

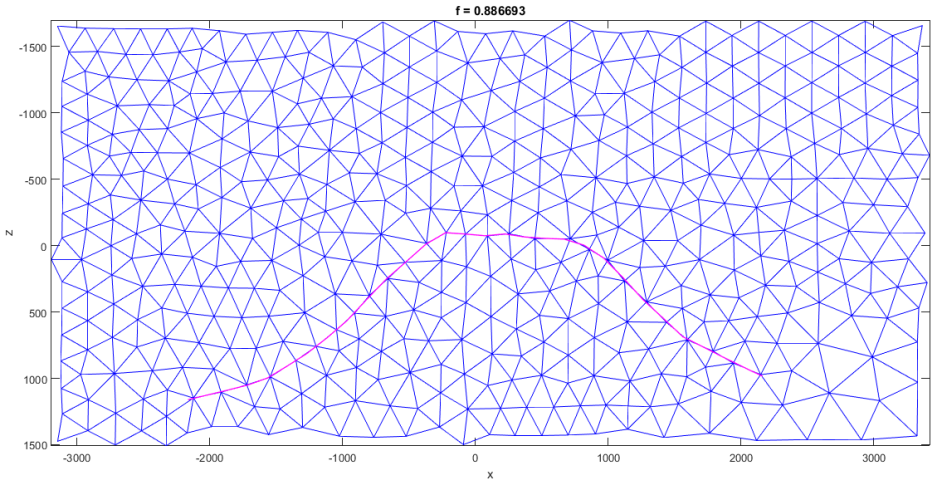


(b) With curve smoothing

Figure A-3: The meshes generated for the model with curve-3 before and after smoothing applied.

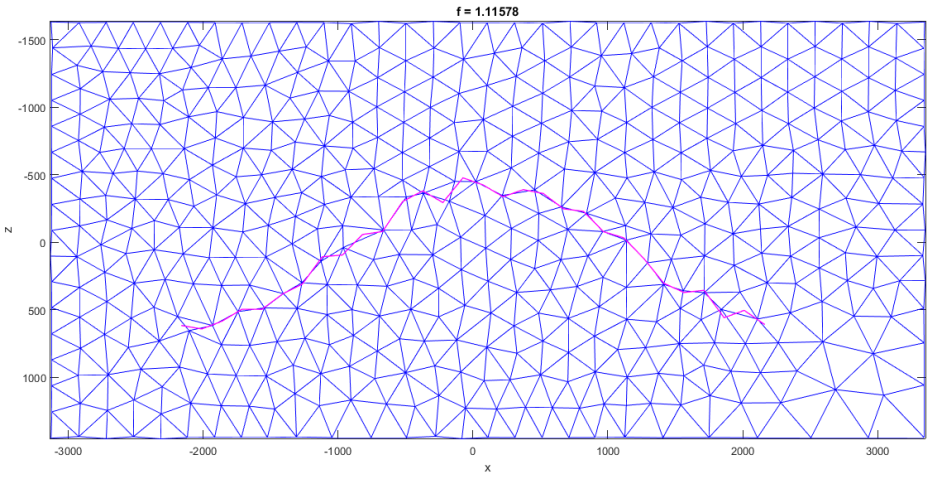


(a) Without curve smoothing

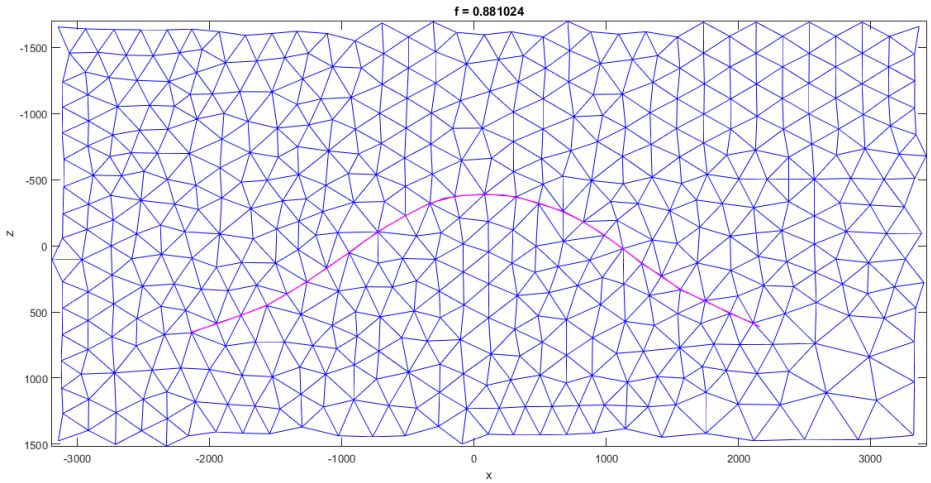


(b) With curve smoothing

Figure A-4: The meshes generated for the model with curve-4 before and after smoothing applied.

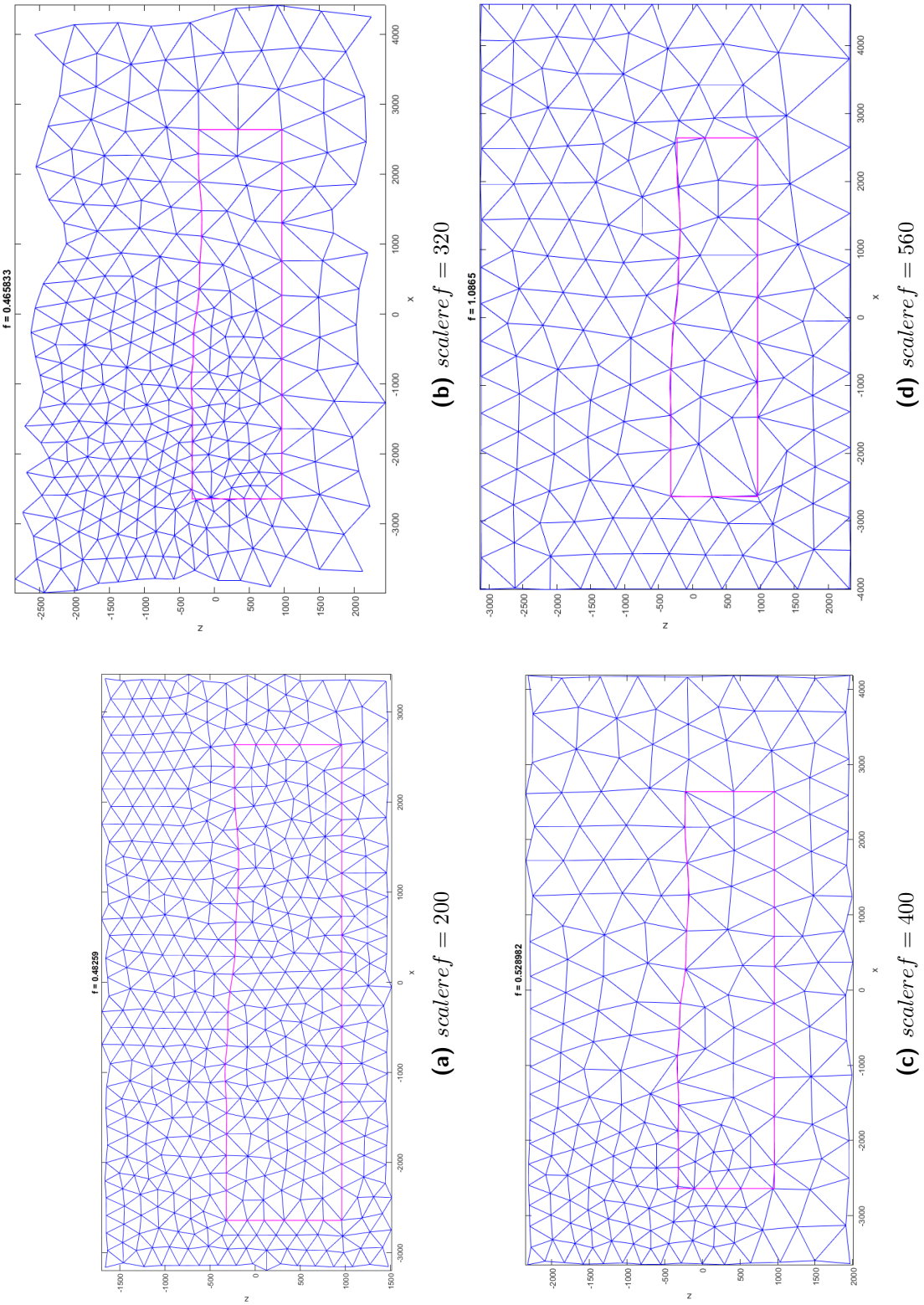


(a) Without curve smoothing



(b) With curve smoothing

Figure A-5: The meshes generated for the model with curve-5 before and after smoothing applied.



(a) $scalref = 200$ (b) $scalref = 320$

(c) $scalref = 400$ (d) $scalref = 560$

Figure A-6: The influence of $scalref$ on the performance of curve fitting as well as the mesh quality.