DISSERTATION

GLOBAL MINIMIZATION OF HOPF BIFURCATION SURFACES WITH APPLICATION TO NEMATIC ELECTROCONVECTION

Submitted by

Ibraheem Alolyan

Department of Mathematics

Adviser: Dr. Eugene Allgower

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 04

COLORADO STATE UNIVERSITY

February 14, 2004

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY IBRAHEEM ALOLYAN ENTITLED "GLOBAL MINIMIZATION OF HOPF BIFURCATION SURFACES WITH APPLICATION TO NEMATIC ELECTROCONVECTION" BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Co-Adviser

Department Head

ABSTRACT OF DISSERTATION

GLOBAL MINIMIZATION OF HOPF BIFURCATION SURFACES WITH APPLICATION TO NEMATIC ELECTROCONVECTION

This dissertation addresses two problems which frequently arise in applications: locating the Hopf bifurcations for autonomous systems of ordinary differential equations and finding the global minimum of continuous functions. These problems are important in dynamical systems and optimization, respectively.

The first problem is to locate Hopf bifurcation points of dynamical systems. Two approaches are used to find the Hopf points. The first approach is the polynomial resultants method. In this method, the characteristic polynomial of a special matrix is introduced and then a necessary and sufficient condition for this polynomial to have two purely imaginary roots is given. The second approach is the Werner Method. In this method, bordered matrices are used to classify the set of matrices with rank deficiency two, at which Hopf points occur.

The second problem is to compute the global minimum of a continuous function defined on a compact region. We use two approaches to find the global minimum. The first approach is the Nelder-Mead method. This method attempts to find the minimum of a continuous function using only functions values without using any information about derivatives. The second approach is the Cell Exclusion Algorithm. We begin with some region on which we wish to determine the global minimum. If a cell fails the minimization condition, it is discarded. If a cell

satisfies the condition, it is subdivided and the condition is then applied to the new cells. For the minimization condition, we choose the monotonicity condition and we then introduce the space of functions of bounded variation which contains all functions that have finite arclength in 1-dimension. We consider the Jordan Decomposition Theorem and use it in the monotonicity condition.

Finally, we apply those methods to the mathematical model of the electroconvection in nematic liquid crystals. The surface to be minimized consists of Hopf bifurcation points and comes from a linear stability analysis performed on the weak electrolyte model and is used to test previous algorithms and compare them.

In this dissertation we present the theory behind Hopf bifurcation and global optimization. We present both types of algorithms and give multiple numerical examples.

Ibraheem Alolyan
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Spring 04

ACKNOWLEDGEMENTS

I gratefully acknowledge my advisor, Professor Eugene Allgower, for his assistance, patience and encouragement through the course of my work and my research at Colorado State University. His constructive suggestions helped to define the direction of this work. Professor Allgower kindly helped to review this dissertation and made very useful suggestions. His support, guidance and friendship is greatly appreciated.

My sincere appreciation is also due to my co-advisor Dr. Iuliana Oprea, for suggesting part of this dissertation and contributing valuable time and ideas to this work. Dr. Oprea's constructive suggestions helped to define the direction of this work. I also wish to thank Professor Allgower and Dr. Oprea for the countless hours they have spent reviewing and critiquing this manuscript. Their help and comments have greatly improved the quality of this work.

A special thanks is due to Dr. Gerhard Dangelmayr and Dr. Paul Heyliger for having served on my committee and providing their help.

Sincere acknowledgement goes out to Professor Kurt Georg who was my co advisor before he passed away suddenly on February 14, 2003. He was a numerical analyst and a leading exponent of numerical continuation methods. I have learned a lot of numerical methods during my discussions with Professor Georg.

I am deeply grateful to my parents, for instilling a learning spirit in me during my childhood and for their love, care and encouragement throughout my whole life.

Lastly, I would like to thank my wife for her patience, encouragement, and moral support throughout my graduate study. I appreciate her help, especially with writing this dissertation. Unfortunately, my daughter Cady was too little to be of much help.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Mathematical models in science are usually nonlinear, often formed by complicated systems of algebraic, ordinary differential or partial differential equations and include a number of characteristic parameters. In problems of theoretical interest as well as engineering practice we are concerned with the dependence of solutions on parameters and particularly with the values of parameters where qualitatively new types of solutions, e.g. new stationary solutions, oscillatory solutions or chaotic attractors appear (bifurcate). In this dissertation we will consider the global minimization of the surfaces of Hopf bifurcation points that may arise in multiparametric stability problems.

This dissertation will address two problems: finding the Hopf bifurcation surface for autonomous sets of ordinary differential equations depending on parameters and locating the global minimum of a continuous function of several variables. These problems are important in dynamical systems and in optimization, respectively.

The first problem we address here is the computation of the global minimum of a continuous function. We consider the case where $f : \Lambda \subset \mathbb{R}^n \rightarrow \mathbb{R}$ is a continuous function and Λ is a compact region. This problem arises very often in global optimization. We use two approaches to find the global minimum.

The first approach is the Cell Exclusion Algorithm. Related algorithms have been used for many years in the area of interval analysis [12, 35, 48]. We begin with some region, e.g. a cell, on which we wish to determine the global minimum.

The algorithm is based on a given optimization condition that can be applied to each cell. If a cell fails the condition, we know it will not contain a point at which a function achieves its global minimum, and it can be discarded. If a cell satisfies the condition, it is subdivided and the condition is then applied to the new cells. For the minimization condition, we choose the monotonicity condition. This condition has not been used in the numerical investigation of global optimization. In fact, it has only been used in the case where f is a polynomial. The monotonicity condition was introduced in [66] for finding zeroes of nonlinear systems of equations, and was then generalized to find the global minima in [17]. The Cell Exclusion Algorithm is discussed in Chapter 2.

In order to use the monotonicity condition in the Cell Exclusion Algorithm, we first present the concept of functions of bounded variation [5, 43, 64]. Connection of the Cell Exclusion Algorithm with functions of bounded variation has not been discussed in the literature before. In this topic, we have introduced new definitions and theorems. The space of functions of bounded variation in 1-dimension contains all functions that have finite arclength, in particular, it includes Lipschitz functions. Among the properties of these functions, we focus our attention to Jordan Decomposition Theorem which states that each function of bounded variation can be written as the difference of two increasing functions p and n. We define a weaker definition of increasing: that is ϵ-increasing concept where ϵ is a positive real number and we then introduce the Decomposition Algorithm. The space of functions of bounded variation is investigated in Chapter 3.

The Cell Exclusion Algorithm for minimization of general multivariate functions and the Decomposition Algorithm are the main contribution of this dissertation. Therefore, several theorems and examples are presented in Chapters 3 and 2. Since this algorithm requires many evaluations of the function, we can use this algorithm to narrow down the size of the cell to get one or more cells in which the

global minima may be and then apply one of the methods that computes the local minimum, e.g., Nelder-Mead Algorithm.

The second approach is the Nelder-Mead method [50]. This method is one of the direct search methods which are easy to program and does not require derivatives [58, 63]. Since its publication in 1965, the Nelder-Mead algorithm has become one of the widely used methods for nonlinear unconstrained optimization. The Nelder-Mead method attempts to find a local minimum for a scalar-valued nonlinear function of n real variables using only functions values without any derivative information (explicit or implicit) [40, 46, 52]. The algorithm for this method and some examples are discussed in Chapter 4.

The second problem we address in this dissertation is to compute numerically Hopf bifurcation points of dynamical systems [24, 30, 47, 62]. We consider the dynamical systems $\dot{x} = f(x, \mu)$, where $x \in \mathbb{R}^n$, $\mu \in \mathbb{R}^m$ and $f : \mathbb{R}^{n+m} \to \mathbb{R}^n$ is continuously differentiable. Then the stability of the stationary solution $(x(\mu), \mu)$ of $f(x, \mu) = 0$ changes at a Hopf bifurcation point which can be characterized by computing the eigenvalues of $f_x(x(\mu), \mu)$ [30]. We discuss Hopf bifurcation points and give introductory information about Hopf points in Chapter 5. We use two approaches to find the Hopf bifurcation points for the dynamical systems $\dot{x} = f(x, \mu)$.

The first approach is to use the characteristic polynomial of the matrix $f_x(x(\mu), \mu)$ (see for example [10, 11, 37, 41, 56]). In this approach we define the Sylvester matrix and the resultant of two polynomials. After that we give the necessary and sufficient condition for a characteristic polynomial to have two purely imaginary roots [29]. This method is known as the polynomial resultants method and is discussed in Chapter 6. This method can often be numerically unstable. Therefore, we will need to use another approach if the dimension of this matrix is high. However, the polynomial resultants method can be used to find the starting value of μ in the second approach.

The second approach is known as the bordering method [61]. This is a recent method which is finding widespread use to compute Hopf bifurcation points. In this method, bordered matrices are used to classify the set of matrices with rank deficiency two, at which Hopf bifurcation occurs. A necessary and sufficient condition for a matrix to have two purely imaginary eigenvalues is given in Chapter 7. This method can be used for higher dimensional matrices.

Both approaches are discussed in this dissertation and numerical algorithms are written to find the Hopf bifurcation points. Furthermore, examples are presented to illustrate how these algorithms work.

Finally, in Chapter 8 we will apply those methods to the mathematical model of electroconvection in nematic liquid crystals. This system is studied in detail in [13]. The surface to be minimized comes from a linear stability analysis performed on the system of partial differential equations describing the weak electrolyte model for the electroconvection of Nematic Liquid Crystals. We apply the bordered matrices algorithm to numerically compute the surface. Then we compute the global minimum of this surface in a box using the cell exclusion algorithm to get smaller boxes at which the global minimum is guaranteed to be in. After that we apply the Nelder-Mead algorithm to these boxes and we obtain the global minimum of the surface.

CHAPTER 2

CELL EXCLUSION ALGORITHM FOR OPTIMIZATION

We consider the constrained minimization problem

$$\min_{x \in \Lambda} f(x),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous function and $\Lambda \subset \mathbb{R}^n$ is a cell. In order to solve this problem, we use the *Cell Exclusion Algorithm*. Related algorithms have been used for many years in the area of interval analysis to solve systems of equations [12, 35, 48]. One can easily describe the basic structure of such an algorithm. We begin with some region, e.g. a cell, on which we wish to determine the global minimum. The algorithm is based on a given optimization condition that can be applied to each cell. If a cell fails the condition, we know it will not contain a point at which a function achieves its global minimum, and it can be discarded. If a cell satisfies the condition, it is subdivided and the condition is then applied to the new cells. This leads to a recursive algorithm. We will investigate how a cell exclusion algorithm may be used to find the global minimum of f on a cell Λ. In Section 2.1 we give some introductory definitions about cells and partitions. In Section 2.2 we describe how a cell exclusion algorithm works and write a general algorithm. We then prove in Section 2.3 that this algorithm converges to the global minimum. In Section 2.4 we give the minimization condition and then conclude this chapter with some numerical examples in Section 2.5.

2.1 Introductory Definitions

We consider the constrained minimization problem

$$\min_{x \in \Lambda} f(x), \tag{2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous function and $\Lambda \subset \mathbb{R}^n$ is a cell. We now give the definition of a cell and other background definitions that will be used in our algorithm. These definitions are based on [3].

Definition 2.1.1 (Cell). A set σ is said to be a cell if it is an n-dimensional region of the form

$$\sigma = \prod_{i=1}^{n} [a_{i1}, a_{i2}],$$

where $a_{i1}, a_{i2} \in \mathbb{R}$, $a_{i1} < a_{i2}$, $i = 1, \ldots, n$.

(Midpoint) The midpoint of a cell σ is defined to be

$$m_\sigma := \begin{pmatrix} \frac{1}{2}(a_{11} + a_{12}) \\ \vdots \\ \frac{1}{2}(a_{n1} + a_{n2}) \end{pmatrix}.$$

(Mesh Size) The mesh size of a cell σ is defined to be $\|d_\sigma\|$, where

$$d_\sigma := m_\sigma - \begin{pmatrix} a_{11} \\ \vdots \\ a_{n1} \end{pmatrix}.$$

We can use any norm to compute $\|d_\sigma\|$; however, unless otherwise noted, the norm which we use is the infinity norm. We now define the partition of a cell and its refinement.

Definition 2.1.2 (Cellular Partition). Let Γ be a finite set of cells and let Λ be a cell in $\mathbb{R}^n$. We say that Γ is a cellular partition of Λ if

1. $\Lambda = \bigcup_{\sigma \in \Gamma} \sigma$.

2. If $\sigma_1, \sigma_2 \in \Gamma$, then $\sigma_1 \cap \sigma_2$ is a common face of σ_1 and σ_2 or $\sigma_1 \cap \sigma_2 = \phi$.

3. The number of cells, $\sigma \in \Gamma$ such that $\sigma \cap \Lambda \neq \phi$ is finite.

Definition 2.1.3 (Refinement). Let Γ_1 and Γ_2 be any two cellular partitions of Λ. Γ_2 is said to be a refinement of Γ_1 if for all $\sigma_2 \in \Gamma_2$ there exists $\sigma_1 \in \Gamma_1$ such that $\sigma_2 \subset \sigma_1$ with strict inclusion holding in at least one case. We also say Γ_2 is finer than Γ_1.

A cell exclusion algorithm looks for a global minimum in some initial cell Λ. As the algorithm executes, Λ is partitioned into successively refined partitions. We give the explicit formulation for the algorithm in the next section.

2.2 Cell Exclusion Algorithm

A cell exclusion algorithm systematically discards cells as it progresses. In order to do this, the algorithm makes use of some test which we will refer to as a *minimization condition (M.C.)*. A minimization condition is a necessary, but not necessarily sufficient, condition which must be satisfied if a global minimum point is present in a cell. Therefore, if a cell fails the minimization condition, we know it does not contain the global minimum and may be discarded immediately.

Definition 2.2.1 (Minimization Condition). A minimization condition is a computationally verifiable necessity test for the presence of a global minimum in a cell.

The algorithm is based on a given sequence of refining partitions Γ_k and uses at each stage a given minimization condition to exclude all cells which cannot contain the global minimum of f. The remaining cells are then stored in Ω_k.

Definition 2.2.2 ($\mathcal{M}_f(\Lambda)$). We define $\mathcal{M}_f(\Lambda)$ to be the set of points where a function $f : \Lambda \subset \mathbb{R}^n \to \mathbb{R}$ attains its global minimum in the cell Λ.

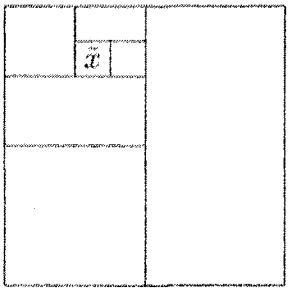


Figure 2.1: Illustration of the sets Ω_k.

Definition 2.2.3 (Ω_k)**.** The set of cells which satisfy the minimization condition on the k^{th} level of partitioning is called Ω_k (see Figure 2.1).

Definition 2.2.4 (Γ_k)**.** The set of cells which result when Ω_{k-1} undergoes one level of partitioning is called Γ_k.

Here is the general minimization algorithm which we will use to obtain the global minimum:

Algorithm 2.2.1. [17]

1. Let Γ_k be the sequence of cellular partitions of Λ defined above with $\Gamma_0 = \{\Lambda\}$ such that Γ_{k+1} is finer than $\Gamma_k, k = 0, 1 \ldots$, and the mesh sizes $\|d_k\| \to 0$ as $k \to \infty$. Let M_k be the current approximation minimum value generated by the algorithm at level k.

2. Let $f : \Lambda \subset \mathbb{R}^n \to \mathbb{R}$.

3. We assume that a given minimization condition can be implemented for each cell σ with midpoint m_σ and $\sigma \subseteq \Lambda$.

4. Set $\Omega_0 \leftarrow \{\Lambda\}$, $M_0 \leftarrow f(m_\Lambda)$. (Initialization)

5. For $k = 0, 1, 2, \ldots$

   If $\|d_k\| < \text{tol}$,

then print M_k.

Else

$\Omega_{k+1} \leftarrow \phi$.

$M_{k+1} \leftarrow M_k$.

For $\sigma \in \Gamma_{k+1}$ such that $\sigma \subset \tau$ for some $\tau \in \Omega_k$

If σ satisfies the minimization condition,

then $\Omega_{k+1} \leftarrow \Omega_{k+1} \cup \{\sigma\}$.

For $\sigma \in \Omega_{k+1}$

If $f(m_\sigma) < M_{k+1}$,

then $M_{k+1} \leftarrow f(m_\sigma)$.

Note that $M_k \leq \min_{\sigma \in \Omega_k} f(m_\sigma)$ and $\{M_k\}$ is a decreasing sequence.

2.3 Convergence of the Optimization Algorithm

The sequence we obtain using the minimization Algorithm 2.2.1 does indeed converge to the global minimum $\tilde{f} := f(\tilde{x})$ for some $\tilde{x} \in \Lambda$.

Theorem 2.3.1. *[17] Let $f : \Lambda \subset \mathbb{R}^n \to \mathbb{R}$ be a continuous function on some cell Λ, and let $\tilde{x} \in \mathcal{M}_f(\Lambda)$. The decreasing sequence of values $\{M_k\}$ generated by the minimization Algorithm 2.2.1 converges to the global minimum $\tilde{f} = f(\tilde{x})$.*

Proof. Since f is uniformly continuous on Λ, for every $\epsilon > 0$ we can find a $\delta > 0$ such that

$$\|x - y\| < \delta \Rightarrow |f(x) - f(y)| < \epsilon \qquad \forall x, y \in \Lambda.$$

Since $\tilde{x} \in \mathcal{M}_f(\Lambda)$, it follows that $\tilde{x} \in \sigma_k \subset \Lambda$ for a sequence of cells $\{\sigma_k\}_{k=1}^{\infty}$ with mesh vectors d_k and midpoints m_k such that $\|d_k\| \to 0$ as $k \to \infty$.

For sufficiently large k, we have

$$\|m_k - \tilde{x}\| < \delta \Rightarrow f(m_k) - f(\tilde{x}) < \epsilon$$

and as a result $f(\tilde{x}) \leq M_k \leq f(m_k) < f(\tilde{x}) + \epsilon$. Thus, since ϵ is arbitrary, M_k must approach $f(\tilde{x})$ as $k \to \infty$. □

The following lemma shows the the relationship between the global minimum $\tilde{f}$ and its approximation M_k at level k.

Lemma 2.3.1. *Let f be C^2 on Λ and let $f : \Lambda \subset \mathbb{R}^n \to \mathbb{R}$ attain its global minimum at a regular zero point, $\tilde{x}$, of the gradient on the interior of the cell Λ, then*

$$M_k \leq f(\tilde{x}) + C\|d_k\|^2$$

where M_k is the sequence defined in the minimization algorithm 2.2.1, d_k is the sequence of mesh vectors in Ω_k, and C is some positive constant independent of k.

Proof. The subdivision process ensures that $\|d_k\| \to 0$ and

$$M_k \leq \min_{\sigma \in \Omega_k} f(m_\sigma)$$

We know that $\tilde{x} \in \sigma$ for some $\sigma \in \Omega_k$ with midpoint m_σ. We now expand f about $\tilde{x}$ and we get

$$f(m_\sigma) - f(\tilde{x}) = \frac{1}{2}(m_\sigma - \tilde{x})^T H_f(\tilde{x})(m_\sigma - \tilde{x}) + O(\|m_\sigma - \tilde{x}\|^3),$$

because the gradient of f vanishes at $\tilde{x}$. Here $H_f(\tilde{x})$ is the Hessian matrix of f at $\tilde{x}$. Since $\tilde{x}$ is a regular point of the gradient, there exists a constant λ^* such that

$$0 < \lambda^* = \|H_f(\tilde{x})\|.$$

Thus, for sufficiently large n we may take C larger than λ^* and neglect the term $O(\|m_\sigma - \tilde{x}\|^3)$ to get

$$f(m_\sigma) - f(\tilde{x}) \leq C\|m_\sigma - \tilde{x}\|^2 \Rightarrow M_k \leq f(m_\sigma) \leq f(\tilde{x}) + C\|d_k\|^2.$$

□

2.4 Minimization Condition

There are several choices for the minimization condition (see for example [17]); nevertheless, we choose the monotonicity condition. This condition has not been previously used in the numerical investigation of global optimization. In fact, it has only been used in the case where f is a polynomial. The monotonicity condition was introduced in [66] for finding zeroes of nonlinear system of equations, and was then generalized to find the global minima in [17].

Definition 2.4.1 (Monotonically Decomposable). The function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be monotonically decomposable if there are two increasing functions g and h such that $f = g - h$.

Given a cell σ, we define the "lowermost" and "uppermost" corners, $\underline{\sigma}$ and $\overline{\sigma}$ respectively, of the cell as follows

$$\underline{\sigma} = (a_{11}, a_{21}, \ldots, a_{n1}),$$

$$\overline{\sigma} = (a_{12}, a_{22}, \ldots, a_{n2}).$$

Notice that all polynomial systems are monotonically decomposable. If we consider a positive domain, we can decompose the polynomials according to terms having positive and negative coefficients (see Example 2.4.1). If we consider a negative domain, we can expand the polynomials about the lowermost corner of the cell then decompose the polynomials according to the positive and negative coefficients of this expansion (see Example 2.4.2).

Example 2.4.1. Consider the following polynomial with $\Lambda = [1, 3]$:

$$f(x) = (x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6$$

We may take $g(x) = x^3 + 11x$ and $h(x) = 6x^2 + 6$. Indeed, $f(x)$ is monotonically decomposable on Λ with $f(x) = g(x) - h(x)$.

Example 2.4.2. Consider the following system of polynomials, $f : \mathbb{R}^2 \to \mathbb{R}$, $f(x) = (f_1(x), f_2(x))$, with $\Lambda = [-3, 3] \times [-3, 3]$, and

$$f_1(x_1, x_2) = (x_1 + 1)x_2,$$
$$f_2(x_1, x_2) = x_2^2 - 1.$$

Expanding $f(x)$ around $(-3, -3)$, we obtain

$$f_1(x_1, x_2) = (x_1 + 3)(x_2 + 3) - 3(x_1 + 3) - 2(x_2 + 3) + 6,$$
$$f_2(x_1, x_2) = (x_2 + 3)^2 - 6(x_2 + 3) + 8.$$

Thus, we may take the functions

$$g, h : \mathbb{R}^2 \to \mathbb{R}$$

to be $g(x) = (g_1(x), g_2(x))$, with $g_1(x_1, x_2) = (x_1 + 3)(x_2 + 3) + 6$, $g_2(x_1, x_2) = (x_2 + 3)^2 + 8$ and $h(x) = (h_1(x), h_2(x))$, where $h_1(x_1, x_2) = 3(x_1 + 3) + 2(x_2 + 3)$, $h_2(x_1, x_2) = 6(x_2 + 3)$. It is easy to see that $f(x)$ is monotonically decomposable on Λ with $f(x) = g(x) - h(x)$.

The following theorem can be applied to give an exclusion test for a cell to have a global minima [17].

Theorem 2.4.1. *Let σ be a cell and $f : \mathbb{R}^n \to \mathbb{R}$ be a Lipschitz function, then f is monotonically decomposable with $f = g - h$. Moreover, if $M \in \mathbb{R}$ and g, h satisfy the following condition:*

$$g(\underline{\sigma}) - h(\overline{\sigma}) > M,$$

then f does not attain a value smaller than or equal to M in σ.

Proof. Since f is a Lipschitz function, f is monotonically decomposable from the Jordan Theorem . By monotonicity, $g(\underline{\sigma}) - h(\overline{\sigma}) \leq g(x) - h(x) = f(x)$ for all $x \in \sigma$. Thus, if $M < g(\underline{\sigma}) - h(\overline{\sigma})$ then $M < f(x)$ for all $x \in \sigma$. □

Suppose that f is a Lipschitz function that is defined on a cell Λ, then we can define the functions g and h so that g and h are increasing on Λ. Then use these functions for the minimization condition in Theorem 2.4.1. However, most of the cells σ will satisfy the condition $g(\underline{\sigma}) - h(\overline{\sigma}) > M$; therefore, we need to find another way to improve this condition. In fact, it suffices to choose g and h such that they are increasing in the small cell σ. Consequently, for each cell σ we should compute the functions g and h such that $f = g - h$ and g and h are increasing in σ. We illustrate this idea in Example 2.5.1 in the next section.

2.5 Numerical Results

In this section we use Algorithm 2.2.1 to find the global minima of several examples. We have programmed a Matlab implementation to solve these examples. We described a method to find the functions g and h if the function f is a polynomial. In the next chapter, Chapter 3, we will give another way to compute the functions g and h so that they are increasing in each subintervals of the domain. The first example shows the advantage of choosing g and h to be increasing in each subinterval rather than choosing g and h to be globally increasing on $[a, b]$.

Example 2.5.1. Let $f : [-3, 2] \to \mathbb{R}$, be defined by

$$f(x) = 3x^4 + 4x^3 - 12x^2 + 4.$$

We plot the function f in Figure 2.2 and we can see that this function has its global minimum at the point $(-2, -28)$. We will compute the functions g and h using two different approaches.

1. The first approach is to compute g and h analytically. Since we have a negative domain we expand f about the point -3 and decompose the polynomials

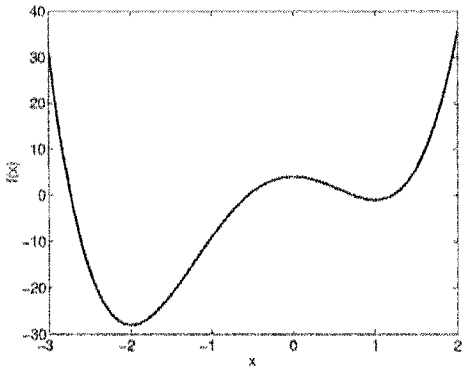


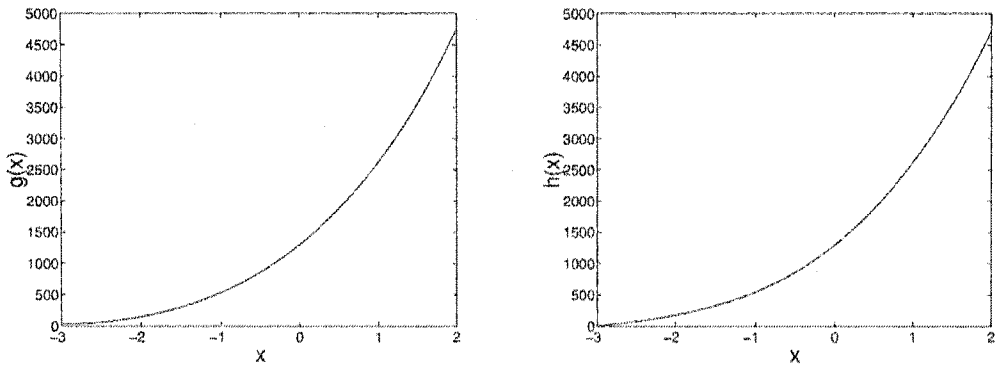
Figure 2.2: The plot of $f(x) = 3x^4 + 4x^3 - 12x^2 + 4$.

g and h according to the positive and negative coefficients of this expansion. Expanding $f(x)$ about the point -3, we get

$$f(x) = 3(x+3)^4 - 32(x+3)^3 + 114(x+3)^2 - 144(x+3) + 31.$$

We now define g and h to be

$$g(x) := 3(x+3)^4 + 114(x+3)^2 + 31,$$
$$h(x) := 32(x+3)^3 + 144(x+3).$$



(a) The function $g(x)$. (b) The function $h(x)$.

Figure 2.3: The functions g and h for Example 2.5.1

Figure 2.3 plots the functions g and h in the domain $[-3, 2]$. We can see immediately that these functions are increasing very fast and therefore, we

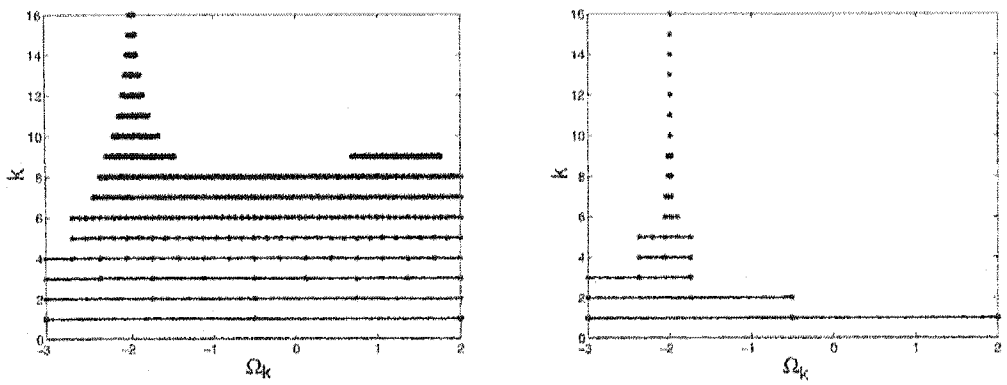
do not expect the condition $g(\underline{\sigma}) - h(\overline{\sigma}) > M$ to be strong enough to discard cells. For example, if we consider the cell $[1, 2]$ then $g(1) - h(2) = -2097$ a value much smaller than $\tilde{f} = -28$. Now we consider a smaller interval $[1.9, 2]$ and then we have $g(1.9) - h(2) = -222.42$ which is again smaller than $\tilde{f} = -28$.

2. The second approach is to use the decomposition algorithm, which will be described in Chapter 3, to compute the functions g and h for each cell σ. In this algorithm we compute $g(1) - h(2) = -5.33$ which is greater than $\tilde{f} = -28$; therefore, this cell can be discarded from the beginning. This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the minimization condition and $2\|d\| <$ tolerance, then σ is in Ω. We now apply Algorithm 2.2.1 with tolerances equal to .1, .01, .001 and .0001. The results appear in Table 2.1 where M. C. C. means Minimization Condition Checks and D. A. means Decomposition Algorithm.

			Analytically		D. A.	
Tol	Level	$\tilde{f}$	Cells in Ω	M. C. C.	Cells in Ω	M. C. C.
0.1	6	-27.991287	60	121	2	15
0.01	9	-27.999965	194	653	4	23
0.001	13	-28.000000	298	1427	4	33
0.0001	16	-28.000000	838	3275	2	39

Table 2.1: Approximation of the minimum of $f = 3x^4 + 4x^3 - 12x^2 + 4$.

We can see from Table 2.1 that the decomposition algorithm requires much less computation than the analytic approach. The reason for that goes back to the way g and h were constructed analytically. They are required to be increasing on the whole interval $[-3, 2]$; therefore, g and h are increasing very fast in such away that it is difficult to check the M. C. In Figure 2.4, we fix the tolerance to be .0001



(a) g, h computed analytically (b) g, h computed by D. A.

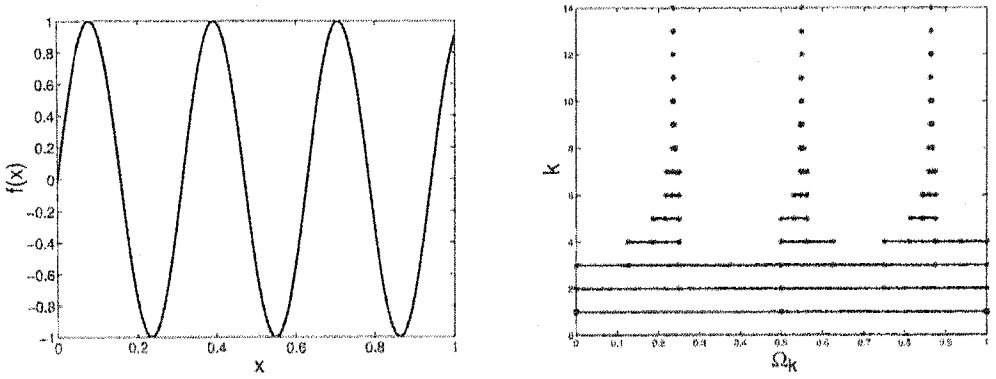
Figure 2.4: The refinement of $[-3, 2]$ for Example 2.5.1

and apply Algorithm 2.2.1 to f using the two approaches that we discussed. We keep intervals that satisfy the M. C. and discards other intervals.

If f is not a polynomial, we might need to use the decomposition algorithm. The following example involves the sine function.

Example 2.5.2. Let $f : [0, 1] \to \mathbb{R}$, be defined by

$$f(x) = \sin(20x).$$



(a) The function $f(x) = \sin(20x)$. (b) Refinement of the domain.

Figure 2.5: The function and the refinement of the domain for Example 2.5.2

The function f has 3 global minimizers in the interval $[0, 1]$. The global minimum is $\tilde{f} = -1$. We plot the function $f(x)$ in Figure 2.5(a). Since this function is not a polynomial, we cannot readily compute an explicit formula for the functions g and h. Therefore; we use the decomposition algorithm to compute the global minimum. We compute the number of levels needed to achieve the desired tolerance, the global minima, the number of cells in Ω, and the number of checks for the minimization condition. Table 2.2 shows the results for different choices of Tolerance. In Figure 2.5(b), we plot the refinement steps and we can see that Algorithm 2.2.1 refines in a neighborhood of the three minimizers.

Tolerance	Level	$\tilde{f}$	Cells in Ω	M. C. Checks
0.1	4	-0.975626	8	23
0.01	7	-0.999914	8	43
0.001	10	-1.000000	6	61
0.0001	14	-1.000000	6	87

Table 2.2: Approximation of the minimum of $f = \sin(20x)$.

We now give other examples for functions in two variables.

Example 2.5.3. Let $f : [-2, 3] \times [-3, 3] \to \mathbb{R}$, be defined by

$$f(x, y) = (y - x^2)^2 + (1 - x)^2.$$

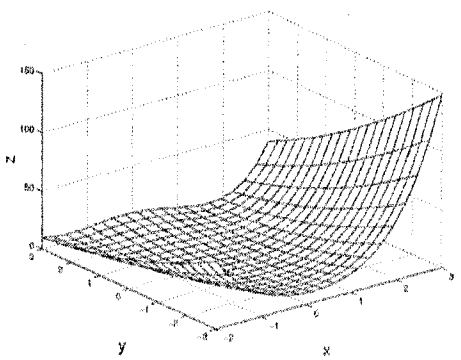


Figure 2.6: The function $f(x, y) = (y - x^2)^2 + (1 - x)^2$.

This function is a well known test function in optimization that has its global minimum 0 at the point $(1, 1)$. We plot the function $f(x, y)$ in Figure 2.6. We define the functions g and h analytically by first expanding $f(x, y)$ about $(-2, -3)$ to get

$$f(x,y) = (x+2)^4 - 2(y+3)(x+2)^2 - 8(x+2)^3 + 8(x+2)(y+3) + (y+3)^2 + 31(x+2)^2 - 62(x+2) - 14(y+3) + 58.$$

Then we decompose the polynomials g and h according to the positive and negative coefficients of this expansion to get

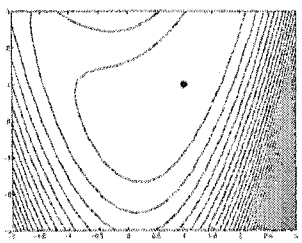
$$g(x,y) = (y+3)^2 + (x+2)^4 + 31(x+2)^2 + 8(x+2)(y+3) + 58,$$
$$h(x,y) = 2(y+3)(x+2)^2 + 8(x+2)^3 + 62(x+2) + 14(y+3).$$

			Analytically		D. A.	
Tol	Level	f	Cells in Ω	M. C. C.	Cells in Ω	M. C. C.
0.1	6	0.000978	3664	4977	96	325
0.01	10	0.000004	-	-	96	721
0.001	13	0.000000	-	-	104	1017
0.0001	16	0.000000	-	-	96	1309

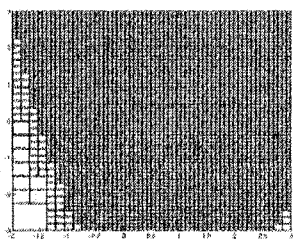
Table 2.3: Approximation of the minimum of $f(x, y) = (y - x^2)^2 + (1 - x)^2$.

We then compute the functions g and h using the decomposition algorithm. Finally, we apply Algorithm 2.2.1 with tolerances .1, .01, .001 and .0001. The results appear in Table 2.3. In the case where g and h are computed analytically, we can see that the number of cells is large even for Tol = 0.1; therefore, we do not proceed in this direction. Figure 2.7 illustrates the level sets of the function f and the refinement of the domain in the cases where g and h are computed analytically and when they are computed using the decomposition algorithm.

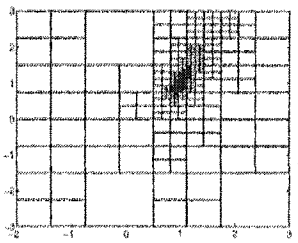
We now present an example where the function $f(x, y)$ oscillates several times in the x-direction and in the y-direction.



(a) The level sets of f



(b) g, h computed analytically



(c) g, h computed by D. A.

Figure 2.7: Refinement of the domain for Example 2.5.3

Example 2.5.4. Let $f : [-1, 1] \times [-1, 2] \to \mathbb{R}$, be defined by

$$f(x, y) = x^2 + y^2 - \sin(10xy).$$

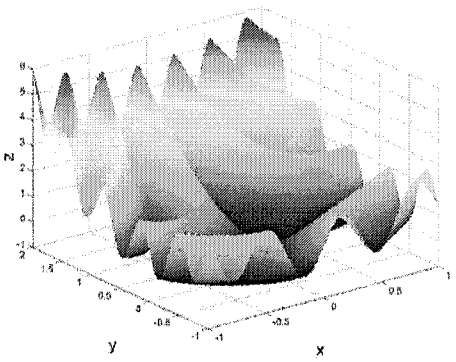


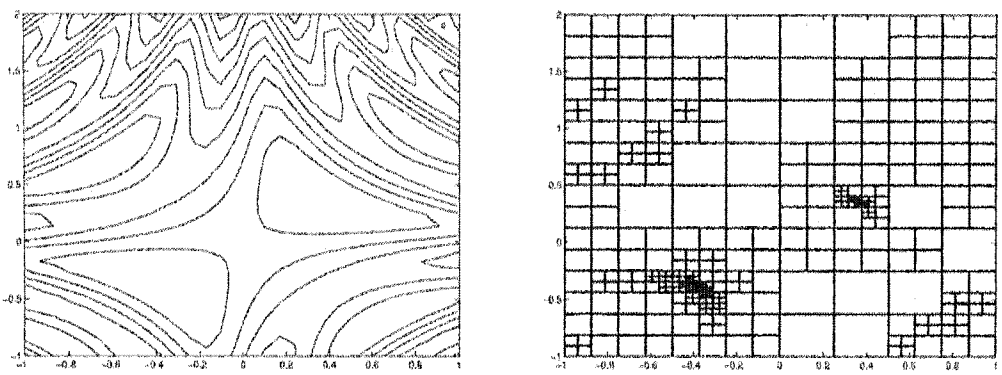
Figure 2.8: The function $f(x, y) = x^2 + y^2 - \sin(10xy)$.

We plot the function $f(x, y)$ in Figure 2.8. Note that f is symmetric in the sense that $f(x, y) = f(y, x)$. Since this function is not a polynomial, we cannot compute an explicit formula for the functions g and h . Therefore, we compute the functions g and h using the Decomposition Algorithm. Finally, we apply Algorithm 2.2.1 with tolerances .1, .01, .001 and .0001. The results appear in Table 2.4.
Figure 2.9 illustrates the level sets of the function f and the refinement of the domain where g and h are computed using the decomposition algorithm.

The following example is taken from [34] where the function f has two global minimizers.

Tolerance	Level	f	Cells in Ω	M. C. Checks
0.1	5	-0.703363	112	373
0.01	9	-0.705904	60	625
0.001	12	-0.705908	72	817
0.0001	15	-0.705908	56	1005

Table 2.4: Approximation of the minimum of $f(x, y) = x^2 + y^2 - \sin(10xy)$.



(a) The level sets of f.

(b) Refinement of the domain.

Figure 2.9: Refinement of the domain for Example 2.5.4

Example 2.5.5. Let $f : \Lambda := [-10, 20] \times [-3, 10] \to \mathbb{R}$, be defined by

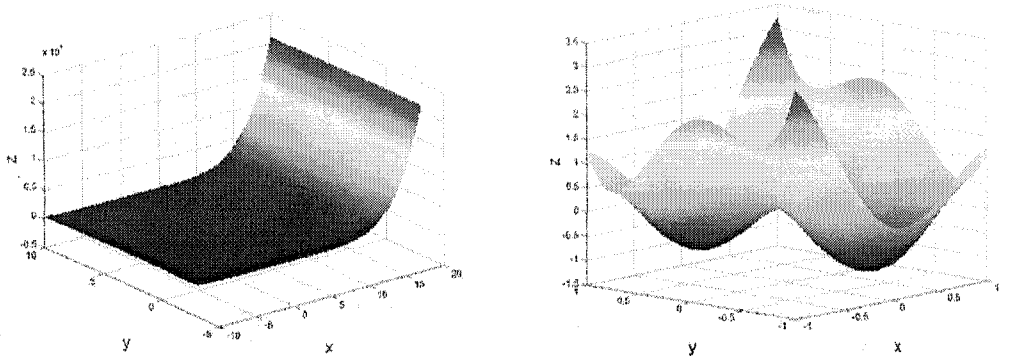
$$f(x, y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4.$$

This test function is known as *six hump camel back function* and it has two global minimizers in this region. We plot the function f on Λ in Figure 2.10(a) and then plot f in a neighborhood of its minimizers in Figure 2.10(b).

Tolerance	Cells in Ω	Cells in Ω in [34]
0.1	9	105
0.1/2	10	149
0.1/4	11	291
0.1/8	12	571
0.1/16	13	1140

Table 2.5: Approximation of the minimum of f for Example 2.5.5.

We use the decomposition algorithm to compute the global minimum and we get $\tilde{f} = -1.02935208$. In Table 2.5, we compare the number of cells we obtain with the number of cells in [34].

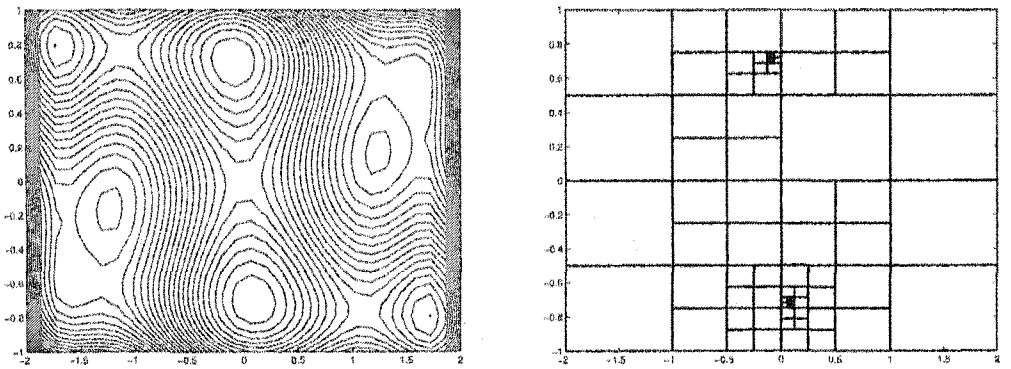


(a) The function f.

(b) Close look at f near the minimizers.

Figure 2.10: The function $f(x,y) = 4x^2 - 2.1x^4 + 1/3x^6 + xy - 4y^2 + 4y^4$.

Figure 2.11 shows the level sets of f and the refinement step for Tol $= 0.01$.



(a) The level sets of f.

(b) Refinement of the domain.

Figure 2.11: Refinement of the domain for Example 2.5.5

We conclude this chapter with an example that has infinitely many minimizers and we see that our algorithm capture the global minima and makes refinements in the circle where this function has its global minima.

Example 2.5.6. Let $f : [-8,8] \times [-8,8] \longrightarrow \mathbb{R}$ be defined by

$$f(x,y) = \frac{\sin(\sqrt{x^2+y^2} + \epsilon_{\text{mach}})}{\sqrt{x^2+y^2} + \epsilon_{\text{mach}}},$$

where ϵ_{mach} is the machine ϵ. This function has infinite global minimizers as can be immediately seen from its graphical representation in Figure 2.12. We use the

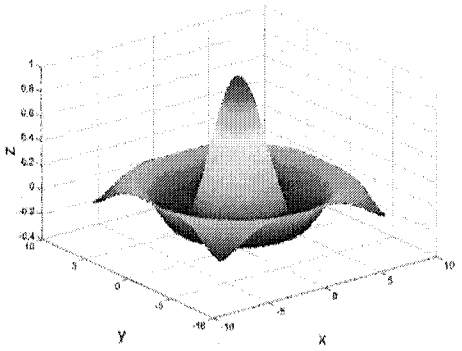


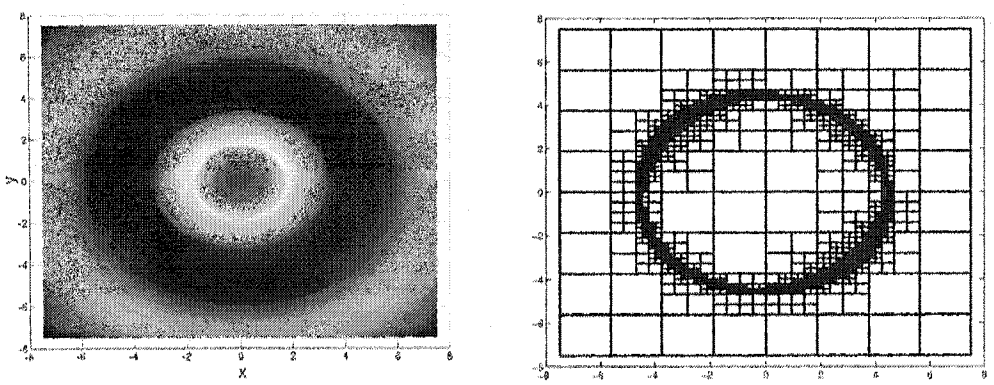
Figure 2.12: The function $f(x,y) = \dfrac{\sin(\sqrt{x^2+y^2}+\epsilon_{\text{mach}})}{\sqrt{x^2+y^2}+\epsilon_{\text{mach}}}$.

Tolerance	Level	f	Cells in Ω	M. C. Checks
1	4	-0.216998	108	193
0.5	5	-0.217210	252	445
0 .25	6	-0.217229	520	965
0.1	8	-0.217234	2252	4353

Table 2.6: Approximation of the minimum of f for Example 2.5.6.

decomposition algorithm to compute g and h then use Algorithm 2.2.1 to find the global minimum and we get $\tilde{f} = -0.217234$. We see from Table 2.6 that the number of cells is 2252 as expected.

Figure 2.13 shows the set level of f and the refinement step for Tol $= 0.1$.



(a) The level sets of f.

(b) Refinement of the domain.

Figure 2.13: Refinement of the domain for Example 2.5.6

Chapter 3

FUNCTIONS OF BOUNDED VARIATION

In Chapter 2 we formulated a cellular exclusion algorithm for functions which can be expressed as a difference of two monotone functions. For this reason we study how to construct such a difference. Functions of bounded variation can be expressed as such a difference and hence we can briefly review the concept of ϵ-increasing. This concept is introduced to find the difference of two functions numerically. It can also be used if we do not have an explicit formula for the function, i.e., in the case where we have a black box that gives the behavior of the function at each point.

The notion of functions of bounded variation (FBV) plays a very significant and important role in the theory of real functions [5, 38], measure and integration [19, 20], partial differential equations [25, 45], numerical analysis [25, 26], applied mathematics [18, 42], mathematics economics, optimization and optimal control [64]. Moreover, there has been increasing interest in these functions because of their importance in Fourier series and probability theory [60].

The space of functions of bounded variation contains all functions that have finite arclength, in particular, it includes Lipschitz functions, functions that are C^1 (i.e., have continuous first derivative), monotone functions, ... etc. In the literature, several properties of these functions have been discussed in one dimension, e.g., [5, 60, 64, 68] and in higher dimensions [1, 43, 44]. We focus our attention to one of these properties that will be used in implementing the Cell Exclusion Algorithm. This property is known as *Jordan Decomposition* and it states that each

function of bounded variation can be written as the difference of two increasing functions p and n. In Section 3.1 we give general information about FBV in one dimension and state the Jordan theorem. In Section 3.2 we write the *Decomposition Algorithm 3.2.1* to find p and n. This algorithm produces two functions p and n that are not increasing; therefore, we define another concept of increasing that is ϵ-increasing where ϵ is a positive real number. We then prove, in Theorem 3.2.1, that the functions p and n, computed by Algorithm 3.2.1, are ϵ-increasing under certain assumptions. Then we generalize our discussion to functions in N dimensions in Sections 3.3 and 3.4.

3.1 Bounded Variation in One Dimension

In this section we will point out some properties of functions of bounded variation in one dimension. Let $f : [a,b] \to \mathbb{R}$ be a function and let $\pi := \{x_i \in [a,b], i = 0,1,\ldots,n : a = x_0 < x_1 < \ldots < x_n = b\}$ be a partition of $[a,b]$. The variation of f over π is the nonnegative real number

$$V_\pi[f;a,b] = \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})|.$$

Definition 3.1.1. If Γ and Λ are two partitions of $[a,b]$, then Γ is finer than Λ if $\Lambda \subseteq \Gamma$.

The variation of f over Λ will either increase or at least be the same if we add more points to Λ. In other words,

Proposition 3.1.1. *If Γ and Λ are two partitions of $[a,b]$ and Γ is finer than Λ then*

$$V_\Gamma[f;a,b] \geq V_\Lambda[f;a,b].$$

The proof of this theorem is standard and can be found in [60]. We define next the concept of *oscillation of a function.*

Definition 3.1.2. We say that a continuous function $f : [a, b] \to \mathbb{R}$ *oscillates* k times on the interval $[a, b]$ if there are exactly k points $s_1, s_2, \ldots, s_k \in (a, b)$ such that for all $i = 1, \ldots, k$, the value $f(s_i)$ is either a strict local maximum or a strict local minimum. If the function f has an infinite number of maximum and minimum points on (a, b), we say that f oscillates infinitely often.

Example 3.1.1. 1. The function $f : [-4, 4] \to \mathbb{R}$, defined by $f(x) = x(x-1)(x-2)(x-3)$, oscillates 3 times.

2. The function $g : (0, 1) \to \mathbb{R}$, defined by $g(x) = x \sin(1/x)$, oscillates infinitely often.

We now consider all functions f which have the property that $V_\pi[f; a, b]$ is bounded for all partitions π of $[a, b]$.

Definition 3.1.3. A function f defined on an interval $[a, b]$ is a function of *bounded variation* if there exists a number M such that for every partition π of $[a, b]$, we have

$$\sum_{i=1}^{n} |f(x_i) - f(x_{i-1})| \leq M.$$

We then define the total variation of f on $[a, b]$ to be the number

$$V_f[a, b] := \sup_{\pi | [a,b]} \sum_{i=1}^{n} |f(x_i) - f(x_{i-1})|, \tag{3.1}$$

where $\pi | [a, b]$ means "π is a partition of $[a, b]$."

Geometrically, the variation of a function is a measure of how much the function oscillates over an interval. We denote the set of all functions of bonded variation on $[a, b]$ by $\mathbb{BV}[a, b]$. For simplicity, we will write $\sup_\pi$ instead of $\sup_{\pi | [a,b]}$.

The following continuous function is an example which is not of bounded variation.

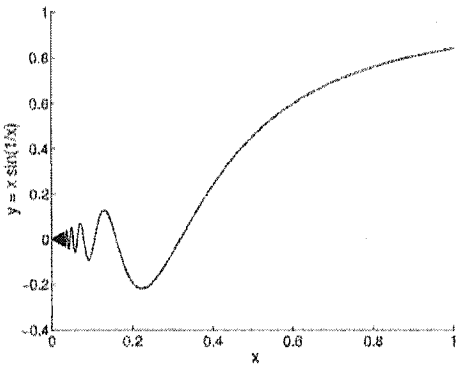


Figure 3.1: The function $f(x) = x\sin(1/x)$.

Example 3.1.2. Let f be the continuous function defined on $[0,1]$ by

$$f(x) = \begin{cases} x\sin\dfrac{1}{x} & \text{if} \quad 0 < x \leq 1, \\ 0 & \text{if} \quad x = 0. \end{cases}$$

The function f is not of bounded variation because f has infinite "length" over the interval $[0,1]$. In other words, continuous functions are not guaranteed to be of bounded variation. The following proposition gives sufficient conditions for a function to be of bounded variation.

Proposition 3.1.2. *[60]*

1. *If f is a* Lipschitz function *on $[a,b]$ then $f \in \mathbb{BV}[a,b]$.*

2. *If $f \in \mathcal{C}^1[a,b]$, then $f \in \mathbb{BV}[a,b]$.*

3. *If f is monotone on $[a,b]$, then $f \in \mathbb{BV}[a,b]$.*

One of the most interesting properties of such functions is that each function of bounded variation can be written as the difference of two increasing functions. We develop some definitions and lemmas to prove this property for FBV. We first view the sum in (3.1) as a sum of positive and negative parts of the differences

$f(x_i) - f(x_{i-1})$. Now we define $P_f[a,b]$ to be the summation of the positive parts of $f(x_i) - f(x_{i-1})$ and $N_f[a,b]$ to be the summation of the negative parts, i.e.,

$$\begin{aligned} P_f[a,b] &:= \sup_{\pi} \sum_{i=1}^{n} (f(x_i) - f(x_{i-1}))^+, \\ N_f[a,b] &:= \sup_{\pi} \sum_{i=1}^{n} (f(x_i) - f(x_{i-1}))^-, \end{aligned} \tag{3.2}$$

where $x^+ := \max\{0, x\}$ and $x^- := \max\{0, -x\}$. Then we have $|x| = x^+ + x^-$ and $x = x^+ - x^-$. Using these formulas, we get

$$\begin{aligned} V_f[a,b] &= P_f[a,b] + N_f[a,b], \\ f(b) - f(a) &= P_f[a,b] - N_f[a,b]. \end{aligned}$$

Now, we will vary b and denote it by x and keep a fixed. Obviously, if f is a function of bounded variation on $[a,b]$ then it is also of bounded variation on each subinterval $[a,x]$ of $[a,b]$. Therefore, if $f \in \mathbb{BV}[a,b]$ then we can define the functions $P_f[a,.], N_f[a,.] : [a,b] \to \mathbb{R}^+ \cup \{0\}$ as we did in (3.2).

Lemma 3.1.1. *If $f \in \mathbb{BV}[a,b]$, then the functions $P_f[a,x]$ and $N_f[a,x]$ are increasing on the interval [a, b].*

Proof. If $x = a$, then $P_f[a,x] = N_f[a,x] = 0$, so let us assume that $x, y \in (a,b]$ and $x < y$, then $[a,x] \subset [a,y]$. Consequently, for every partition Γ of $[a,x]$, we have $\Lambda := \Gamma \cup \{y\}$ is a partition of $[a,y]$. Thus we have

$$\begin{aligned} \sum_{i=1}^{n_\Gamma} (f(x_i) - f(x_{i-1}))^+ &\leq \sum_{i=1}^{n_\Gamma} (f(x_i) - f(x_{i-1}))^+ + (f(y) - f(x))^+ \\ &= \sum_{i=1}^{n_\Lambda} (f(x_i) - f(x_{i-1}))^+. \end{aligned}$$

If we take the supremum in both sides of all partitions Γ, we get $P_f[a,x] \leq P_f[a,x] + (f(y) - f(x))^+ \leq P_f[a,y]$. The same argument can be made to prove that $N_f[a,x]$ is increasing on $[a,b]$. □

This observation leads to the following theorem, known as the *Jordan Decomposition Theorem*, which shows that a function of bounded variation can be expressed as the difference of two increasing functions.

Theorem 3.1.1 (Jordan Decomposition). *[60] If f is a function of bounded variation on $[a, b]$ then f can be written as the difference of two increasing functions*

$$f(x) = p(x) - n(x).$$

Proof. We have $f(x) - f(a) = P_f[a, x] - N_f[a, x]$ for all $x \in [a, b]$. Choose $p(x) = f(a) + P_f[a, x]$ and $n(x) = N_f[a, x]$ then apply Lemma 3.1.1. □

In Section 3.2 we will present and algorithm to find the functions p and n; the generalization of Theorem 3.1.1 to functions of N variables will be given in Section 3.3.

3.2 Decomposition Algorithm and Examples

Even though the Jordan decomposition theorem is a familiar theorem, it has not been exploited numerically to find p and n. That is because the supremum sum over all partitions is impossible to be computed numerically. We will describe below an algorithm that approximates p and n with respect to the uniform partition $\pi := \{a + i(x - a)/(m + 1)\}_{i=0}^{m+1}$ of the interval $[a, x]$. In this algorithm, we choose m to be the number of points between a and x to approximate $P_f[a, x]$ and $N_f[a, x]$. Consequently, the functions p and n will not be always increasing. We give Example 3.2.1 to show that p and n cannot be increasing on $[a, b]$ even for smooth functions and for any value of m. To avoid this problem, we define ϵ-increasing, a weaker definition than increasing, and then show that p and n are ϵ-increasing if f is a Lipschitz function and m is chosen large enough.

Algorithm 3.2.1 (Decomposition Algorithm). INPUT: a, f and x.

OUTPUT: p and n evaluated at x.

choose m (number of points between a and x).

$d := (x - a)/(m + 1)$.

for $i = 0 : m + 1$

$x_i := a + id$.

$p = f(a)$.

$n = 0$.

for $i = 1 : m + 1$

if $s = f(x_i) - f(x_{i-1}) \geq 0$

$p = p + s$.

else

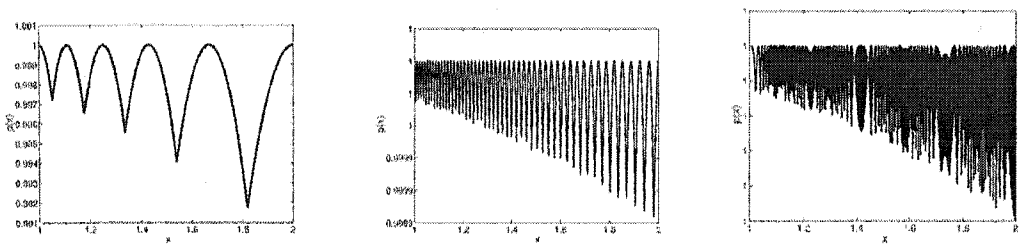
$n = n + s$.

Note that we can choose $m + 1 = 2^k$ to decrease the number of computations when we increase m (i.e., if we choose $m + 1 = 2^{k+1}$ then we already computed the the values $f(x_i)$ for i is even). The main question about this algorithm is how to compute m such that p and n are increasing. Unfortunately, m cannot always be computed even for smooth functions. We give the following example to illustrate this point.

Example 3.2.1. Let $f : [0, 2] \to \mathbb{R}$ be defined by $f(x) := 1 - (1 - x)^2$. In theory, this function can be written as the difference of two increasing functions, e.g., $p(x) = 2x$ and $n(x) = x^2$. In order to compute p and n numerically, we notice that this function is increasing on the interval $[0, 1]$; therefore, the value of $p(x)$ will increase on this interval for any choice of m. Nevertheless, on the interval $(1, (m+1)/m)$, the function p will take values less than one. To explain this, let x

be any point in $(1,(m+1)/m)$, and $\pi : \{ix/(m+1)\}_{i=0}^{m+1}$ be the uniform partition of $[0,x]$, then $f(x) < f(1)$ and $mx/(m+1) \in (m/(m+1),1)$ and we have

$$\begin{aligned} p(x) &= \sum_{i=1}^{m+1}(f(x_i) - f(x_{i-1}))^+ \\ &= f(mx/(m+1)) - f(0) + (f(x) - f(mx/(m+1)))^+ \\ &< f(mx/(m+1)) - f(0) + (f(1) - f(mx/(m+1))) \\ &= f(1) - f(0) = 1 = p(1). \end{aligned}$$



(a) m = 10 (b) m = 100 (c) m = 1000

Figure 3.2: The function p in Example 3.2.1

Therefore, the function p is not increasing on $(1,(m+1)/m)$. In fact, p equals the value one if there exists an i, $1 \leq i \leq m+1$, such that $ix/(m+1) = 1$, i.e., the point 1 is one of the mesh points. Solving for x, we get $x = (m+1)/i$; therefore, p equals one if $x = (m+1)/(m+1) = 1$, $x = (m+1)/m$, $x = (m+1)/(m-1)$, ... or $x = m+1$. Figure 3.2 plots the function p on the interval $[1,2]$ for $m = 10, 100$ and 1000. Although the functions in Figure 3.2 increase in oscillations, the amplitude decreases as m increases. We can see from the plots that for $m = 10$, we have $p = 1$ on the interval $[1,2]$ for the values $1, 10/9, 10/8, 10/7, 10/6$, and $10/5$; however, for all other values, the function p is strictly less than one. For $m = 100$, the function p oscillates more because it equals one at $1, 100/99, 100/98, \ldots, 100/50$ and it is strictly less than one for all other values on $[1,2]$. Finally, the same argument holds for the case where $m = 1000$.

The previous example shows that m cannot be chosen in such a way that p is always increasing. In fact, p will oscillate in a small ϵ-neighborhood of 1. As m increases, the number ϵ becomes smaller and smaller. This idea led us to introduce a new definition that we call ϵ-increasing and we will use it in the Decomposition Algorithm.

Definition 3.2.1 (ϵ-Increasing). We say that the function $f : [a,b] \to \mathbb{R}$ is ϵ-increasing, where ϵ is a positive number, if $f(x) \leq f(y) + \epsilon$ for all $x < y$ and $x, y \in [a,b]$.

We now give the definition of the oscillation point of a continuous function.

Definition 3.2.2 (Oscillation Point). A point $x \in [a,b]$ is said to be an oscillation point of the function f, if for all $\delta > 0$, the function f has an infinite number of oscillations on $[x-\delta, x+\delta] \bigcap [a,b]$.

In our next discussion, we choose a special subspace of the space of functions of bounded variation, that is the space of functions which satisfy a Lipschitz condition because in this space, the value $|f(x_i) - f(x_{i-1})|$ can be controlled by $|x_i - x_{i-1}|$.

Definition 3.2.3 (Lipschitz Functions). [2] A function $f : [a,b] \to \mathbb{R}$ is said to be a Lipschitz function on $[a,b]$ with a Lipschitz constant C, if there is a constant C such that

$$|f(x) - f(y)| \leq C|x - y| \quad \forall x, y \in [a,b].$$

We denote the space of Lipschitz functions on $[a,b]$ by $Lip[a,b]$.

We have discussed in Proposition 3.1.2 that all functions in $Lip[a,b]$ are of bounded variation on $[a,b]$. We come now to the main theorem in this section which shows that p and n in Algorithm 3.2.1 have, in fact, a special property that can be exploited numerically.

Theorem 3.2.1. *If $f : [a, b] \to \mathbb{R}$ is a Lipschitz function with a Lipschitz constant C, and the number of oscillation points is finite (can be 0) then for all $\epsilon > 0$ there exists $m := m_\epsilon \in \mathbb{N}$ such that p and n in Algorithm 3.2.1 are ϵ-increasing.*

In order to prove this theorem, we need to prove the following propositions. We will always take the partitions $\{x_i\}_{i=0}^{m+1}$ and $\{y_i\}_{i=0}^{m+1}$ to be the uniform partitions of $[a, x]$ and $[a, y]$, respectively. Moreover, every time we mention p and n, we refer to p and n previously introduced in Algorithm 3.2.1.

Lemma 3.2.1. *1. If f is increasing on $[a, b]$, then p is increasing and we have $p(x) = f(x)$ and $n(x) = 0$ for all $x \in [a, b]$.*

2. If f is decreasing on $[a, b]$, then n is increasing and we have $p(x) = f(a)$ and $n(x) = f(a) - f(x)$ for all $x \in [a, b]$.

Proof. This follows immediately from the way p and n are constructed in Algorithm 3.2.1 □

In the following discussion, we will consider the function p and prove that this function is ϵ-increasing. A similar argument can be made to prove that the function n is also ϵ-increasing.

Proposition 3.2.1. *If $f : [a, b] \to \mathbb{R}$ is a Lipschitz function with a Lipschitz constant C and there is a point $c \in [a, b]$ such that f is increasing on $[a, c]$ and decreasing on $[c, b]$, then for all $\epsilon > 0$, there exists an integer $m_\epsilon \in \mathbb{N}$ such that if we choose $m = m_\epsilon$ in Algorithm 3.2.1, then the resulting function p is ϵ-increasing on $[a, b]$, i.e.,*

$$p(x) \leq p(c) \leq p(x) + \epsilon \quad \forall x > c.$$

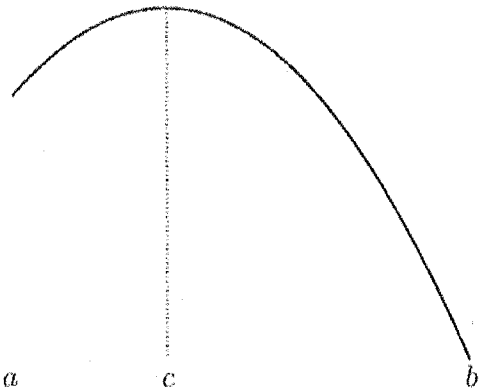


Figure 3.3: The shape of the function $f(x)$ in Proposition 3.2.1

Proof. Let $\epsilon > 0$ be given, then choose $m := m_\epsilon$ such that $1/(m+1) < \epsilon/C(b-a)$. If $x \in (c, b]$ and $\{x_i = a + i(x-a)/(m+1)\}_{i=0}^{m+1}$ is the uniform partition of $[a, x]$, then $|f(x_i) - f(x_{i-1})| \leq C(x_i - x_{i-1}) \leq C(b-a)/(m+1) < \epsilon$. Since $x > c$, there must be an index k between 0 and m such that $x_k \leq c < x_{k+1}$. We now compute $p(x)$ and assume that $m < k$ (if $m = k$, then the last summation $\sum_{i=k+2}^{m+1}$ will not appear).

$$
\begin{aligned}
p(x) &= f(a) + \sum_{i=1}^{m+1} (f(x_i) - f(x_{i-1}))^+ \\
&= f(a) + \underbrace{\sum_{i=1}^{k} (f(x_i) - f(x_{i-1}))^+}_{= f(x_k) - f(a)} + \underbrace{(f(x_{k+1}) - f(x_k))^+}_{\geq 0} \\
&\quad + \underbrace{\sum_{i=k+2}^{m+1} (f(x_i) - f(x_{i-1}))^+}_{=0} \\
&\geq f(a) + f(x_k) - f(a) \\
&\geq f(c) - \epsilon \qquad \text{because} f(c) - f(x_k) \leq \epsilon \\
&= p(c) - \epsilon.
\end{aligned}
$$

To prove the left inequality, we note that $(f(x_{k+1}) - f(x_k))^+ \leq f(c) - f(x_k)$ because $f(x_{k+1}) < f(c)$. Then it follows that $p(x) \leq f(c) = p(c)$. □

Corollary 3.2.1. *If $f[a,b] \to \mathbb{R}$ satisfies the conditions in Proposition 3.2.1 then p is ϵ-increasing on $[a,b]$.*

Proof. Let $x, y \in [a,b]$ and $x < y$ then we have three cases:

1. If $x < y < c$, then $p(x) \leq p(y)$ because p is increasing on $[a,c]$.
2. If $x < c < y$, then $p(x) \leq p(c) \leq p(y) + \epsilon$ (from Proposition 3.2.1).
3. If $c < x < y$, then $p(x) \leq p(c)$ and $p(c) \leq p(y) + \epsilon$; therefore, $p(x) \leq p(y) + \epsilon$.

□

Proposition 3.2.2. *If $f : [a,b] \to \mathbb{R}$ is a Lipschitz function with a Lipschitz constant C and there is a point $c \in [a,b]$ such that f is decreasing on $[a,c]$ and increasing on $[c,b]$ then there exists an $m_\epsilon \in \mathbb{N}$ such that if we choose $m = m_\epsilon$ in Algorithm 3.2.1, then the resulting function p is ϵ-increasing on $[a,b]$.*

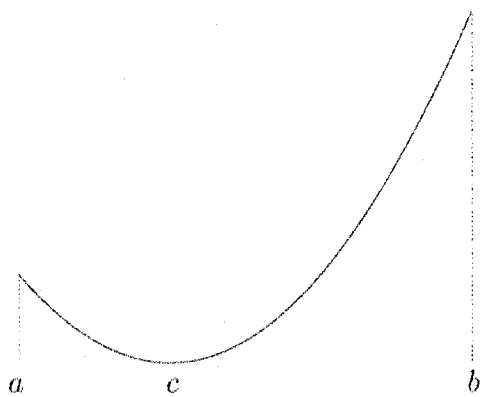


Figure 3.4: The shape of the function $f(x)$ in Proposition 3.2.2

Proof. Let $x, y \in [a,b]$ be such that $x < y$. Choose $m := m_\epsilon$ such that the following inequality holds $C(b-a)/(m+1) < \epsilon/3$ and consider the two uniform partitions $\{x_i\}_{i=0}^{m+1}$ and $\{y_i\}_{i=0}^{m+1}$ for $[a,x]$ and $[a,y]$, respectively. If $x \leq c$, then $f(x) = f(a)$ and $f(y) \geq f(a)$ because $p(y) - f(a) = \sum_{i=1}^{m+1}(f(y_i) - f(y_{i-1}))^+ \geq 0$. So, we assume that $c < x < y$. Since $\{y_i\}$ is a partition of $[a,y]$, there is k between 0 and

m such that $y_k \leq c < y_{k+1}$. If $y_k = c$, then $p(y) = f(a) + f(y_{m+1}) - f(y_k)$ which is obviously greater than $p(x)$. If $y_k < c < y_{k+1}$, then there are some points x_i such that $y_k \leq x_i \leq y_{k+1}$ for $i = r+1, r+2, \ldots, r+s$. Now either $r+s = m+1$ or $r+s < m+1$. If $r+s = m+1$ then

$$\begin{aligned}\sum_{i=r+1}^{r+s} (f(x_i) - f(x_{i-1}))^+ &= (f(x_{r+1}) - f(x_r))^+ + \sum_{i=r+2}^{r+s} (f(x_i) - f(x_{i-1}))^+ \\ &\leq C(x_{r+1} - x_r) + C \sum_{i=r+1}^{r+s} (x_i - x_{i-1}) \\ &\leq C(x_{r+1} - x_r) + C(y_k - y_{k-1}) \\ &< 2C\frac{b-a}{m+1} < \frac{2\epsilon}{3}.\end{aligned}$$

Therefore,

$$p(x) \leq f(a) + \frac{2\epsilon}{3} < p(y) + \epsilon.$$

If $r+s < m+1$ then we have $y_k \leq x_i \leq y_{k+1} < x_{r+s+1}$ and the value of $p(x)$ is

$$\begin{aligned}p(x) =& f(a) + \underbrace{\sum_{i=1}^{r} (f(x_i) - f(x_{i-1}))^+}_{=0} + \underbrace{\sum_{i=r+1}^{r+s} (f(x_i) - f(x_{i-1}))^+}_{\leq 2\epsilon/3} \\ &+ \underbrace{(f(x_{r+s+1}) - f(x_{r+s}))^+}_{\leq \epsilon/3} + \sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+.\end{aligned}$$

For the last part of the expression of $p(x)$ we have $f(y_{k+1}) < f(x_{r+s+1})$ and $f(y_{m+1}) > f(x_{m+1})$, this implies

$$\begin{aligned}\sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+ =& f(x_{m+1}) - f(x_{r+s+1}) \\ <& f(y_{m+1}) - f(y_{k+1}) \\ =& \sum_{i=k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+.\end{aligned}$$

Now we compute $p(y)$,

$$\begin{aligned} p(y) =& f(a) + \underbrace{\sum_{i=1}^{k} (f(y_i) - f(y_{i-1}))^+}_{=0} + \underbrace{(f(y_{k+1}) - f(y_k))^+}_{\geq 0} \\ &+ \sum_{k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+ \\ \geq& p(x) - \epsilon. \end{aligned}$$

□

We treated the case where f oscillates one time. In the following proposition we will prove that if p is ϵ-increasing on the interval $[a, c]$ where the function f is increasing (decreasing), then p will remain ϵ-increasing in the interval $[c, b]$ where f is decreasing (increasing).

Proposition 3.2.3. *If $f : [a, b] \to \mathbb{R}$ is a Lipschitz function and there are $c, d, e \in (a, b)$ with $a < c < d < e < b$ such that f is increasing on $[a, c]$ and $[d, e]$ and decreasing on $[c, d]$ and $[e, b]$ then for all $\epsilon > 0$, there is an $m_\epsilon \in \mathbb{N}$ such that if we choose $m = m_\epsilon$ in Algorithm 3.2.1, then the resulting function p is ϵ-increasing on $[a, b]$.*

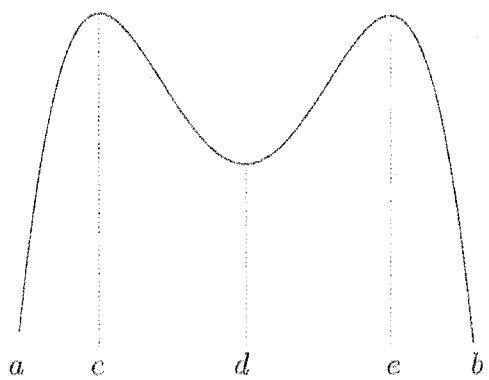


Figure 3.5: The shape of the function $f(x)$ in Proposition 3.2.3

Proof. Let $\epsilon > 0$ be given, choose $m := m_\epsilon$ such that $C(b-a)/(m+1) < \epsilon/5$. We will consider three cases

1. On the interval $[a, d]$; it follows from Proposition 3.2.1 that the function p is ϵ-increasing. Furthermore, for all $x \in [a, d]$, we have $p(x) \leq p(c)$.

2. On the interval $[d, e]$; let $y \in [d, e]$ and $\{y_i\}_{i=0}^{m+1}$ be the uniform partition of $[a, y]$ then we claim that $p(c) \leq p(y) - \epsilon/5$. To show this, we notice the existence of k such that $y_k \leq d < y_{k+1}$ and

$$p(y) = \underbrace{f(a) + \sum_{i=1}^{k} (f(y_i) - f(y_{i-1}))^+}_{=p(y_k)} + \underbrace{(f(y_{k+1}) - f(y_k))^+}_{\geq 0} + \underbrace{\sum_{i=k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+}_{\geq 0}.$$

Since $p(c) - p(y_k) \leq \epsilon/5$, we have $p(y) \geq p(c) - \epsilon/5$. Now, choose $x < y$, then either $x \in [a, d]$ and therefore, $p(x) \leq p(c) \leq p(y) + \epsilon/5$ or $x \in (d, e]$. If $x \in (d, e]$ then there are r and s such that $y_k \leq x_i \leq y_{k+1}$ for $i = r+1, \ldots, r+s$. If $r + s = m + 1$ then we have

$$\begin{aligned}
\sum_{i=r+1}^{r+s} (f(x_i) - f(x_{i-1}))^+ &= (f(x_{r+1}) - f(x_r))^+ + \sum_{i=r+2}^{r+s} (f(x_i) - f(x_{i-1}))^+ \\
&\leq C(x_{r+1} - x_r) + C \sum_{r+1}^{r+s} (x_i - x_{i-1}) \\
&\leq C(x_{r+1} - x_r) + C(y_k - y_{k-1}) \\
&< 2C \frac{b-a}{m+1} < \frac{2\epsilon}{5}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
p(x) &= \underbrace{f(a) + \sum_{i=1}^{r} (f(x_i) - f(x_{i-1}))^+}_{=p(x_r)} + \underbrace{\sum_{i=r+1}^{r+s} (f(x_i) - f(x_{i-1}))^+}_{\leq 2\epsilon/5} \\
&\leq p(c) + \frac{2\epsilon}{5} \\
&< p(y) + 3\epsilon/5.
\end{aligned}$$

If $r+s < m+1$, then we have $y_k \leq x_i \leq y_{k+1} < x_{r+s+1}$ for $i = r+1, \ldots, r+s$. Moreover,

$$\sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+ \leq \sum_{i=k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+.$$

We now compute $p(y)$ and $p(x)$ and compare them

$$\begin{aligned} p(y) &= \underbrace{f(a) + \sum_{i=1}^{k} (f(y_i) - f(y_{i-1}))^+}_{=p(y_k)} + \underbrace{(f(y_{k+1}) - f(y_k))^+}_{\geq 0} \\ &\quad + \sum_{i=k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+ \\ &\geq p(c) - \epsilon/5 + \sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+, \end{aligned}$$

and

$$\begin{aligned} p(x) &= \underbrace{f(a) + \sum_{i=1}^{r} (f(x_i) - f(x_{i-1}))^+}_{=p(x_r)} + \underbrace{\sum_{i=r+1}^{r+s} (f(x_i) - f(x_{i-1}))^+}_{\leq 2\epsilon/5} \\ &\quad + \underbrace{(f(x_{r+s+1}) - f(x_{r+s}))^+}_{\leq \epsilon/5} + \sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+ \\ &\leq p(c) + 2\epsilon/5 + \epsilon/5 + p(y) - p(c) + \epsilon/5 \\ &= p(y) + 4/5\epsilon. \end{aligned}$$

3. On the interval $[e, b]$; let $x \in [a, b]$ and $y \in (e, b]$ such that $x < y$. We first show that $p(e) \leq p(y) + \epsilon/5$. For the partition $\{y_i\}$, there is an index k such that $y_k \leq e < y_{k+1}$ and we have

$$\begin{aligned} p(y) &= \underbrace{f(a) + \sum_{i=1}^{k} (f(y_i) - f(y_{i-1}))^+}_{=p(y_k)} + \underbrace{(f(y_{k+1}) - f(x_k))^+}_{\geq 0} \\ &\quad + \underbrace{\sum_{i=k+2}^{m+1} (f(y_i) - f(y_{i-1}))^+}_{=0} \\ &\geq p(e) - \epsilon/5. \end{aligned}$$

If $x \in [a, e]$, then $p(x) \leq p(e) + 4\epsilon/5$ and $p(e) \leq p(y) + \epsilon/5$ which implies $p(x) \leq p(y) + \epsilon$. So we assume that $x \in (e, b]$. Then for the partition $\{x_i\}$, we have $y_k \leq x_i \leq y_{k+1}$ for $i = r+1, \ldots, r+s$. If $r + s = m + 1$, we have

$$\begin{aligned} p(x) &= \underbrace{f(a) + \sum_{i=1}^{r} (f(x_i) - f(x_{i-1}))^+}_{\leq p(e)+\epsilon/5} + \underbrace{(f(x_{r+1}) - f(x_r))^+}_{<\epsilon/5} \\ &\quad + \underbrace{\sum_{i=r+2}^{r+s} (f(x_i) - f(x_{i-1}))^+}_{<\epsilon/5} \\ &\leq p(e) + 3\epsilon/5. \end{aligned}$$

If $r + s < m + 1$, we will have two more terms added to the previous $p(x)$

$$\underbrace{(f(x_{r+s+1}) - f(x_{r+s}))^+}_{<\epsilon/5} + \underbrace{\sum_{i=r+s+2}^{m+1} (f(x_i) - f(x_{i-1}))^+}_{=0} \leq \epsilon/5;$$

therefore, $p(y) \geq p(e) - \epsilon/5 \geq p(x) - \epsilon$. □

After stating and proving the previous propositions, we now have the tools to prove Theorem 3.2.1.

Proof of Theorem 3.2.1. Since $f \in Lip[a, b]$ and has a finite number of oscillation points, we have that either f is monotone, has finite number of oscillations or has finite number of oscillation points.

1. If f is monotone then p is increasing.

2. If f oscillates finitely many times then there is a partition $\Gamma := \{\xi_i\}_{i=0}^{l}$ (needs not be uniform) such that f is increasing on some intervals and decreasing on others. We then use induction on Proposition 3.2.3 to prove that p is ϵ-increasing.

3. If f has finite number of oscillation points in $[a, b]$, say $s_1, \ldots, s_k$, then for

all $\delta > 0$, the function f oscillates at most finitely many times on $[a,b] - \bigcup([s_i - \delta, s_i + \delta] \bigcap [a,b])$. Since this is true for any $\delta > 0$, we can choose $\delta < \epsilon/4Ck$. For $i = 1, 2, \ldots, k$, let $\{t_{ij}\}_{j=1}^{n_i}$ be a partition of the interval $[s_i - \delta, s_i + \delta] \bigcap [a,b]$ then we have

$$\sum_{j=1}^{n_i} (f(t_{ij}) - f(t_{i(j-1)}))^+ \leq C \sum_{j=1}^{n_i} (t_{ij} - t_{i(j-1)}) \leq 2C\delta < \epsilon/2k.$$

If we take the summation over all the intervals around $s_1, \ldots, s_k$, we get

$$\sum_{i=1}^{k} \sum_{j=1}^{n_i} (f(t_{ij}) - f(t_{i(j-1)}))^+ < \epsilon/2.$$

Now we will consider the case where we have one oscillation point and then use induction. Suppose that s is the only oscillation point of f on $[a,b]$ and f oscillates n_s times on $[a,b] - [s - \delta, s + \delta]$ where $2\delta < \epsilon/6$. We prove next that the function p is ϵ-increasing on $[a,b]$. Choose $m_\epsilon := m$ to be $C(b-a)/(m+1) < \epsilon/3(2n_s + 4)$ and let $x, y \in [a,b]$ such that $x < y$. We will consider the case where $x > s + \delta$ (if $x \leq s + \delta$ then this case can be proven with the same argument). Since f has finitely many oscillations on $[a, s - \delta]$ and $[s + \delta, b]$, the function p is $\epsilon/3$-increasing on these intervals. We consider the uniform partitions $\{x_i\}_{i=1}^{m+1}$ and $\{y_i\}_{i=1}^{m+1}$ of $[a,x]$ and $[a,y]$, respectively. Let y_k be such that $y_k \leq s - \delta < y_{k+1}$ and $y_{k+r} \leq s + \delta < y_{k+r+1}$ where r can take the value 0. For the partition $\{x_i\}$, let $x_e \leq y_k < x_{e+1}$ and $y_{k+r} \leq x_{e+f} < y_{k+r+1}$. Now p is $\epsilon/3$-increasing on $[a, s - \delta]$; therefore, $p(y_k) > p(x_e) - \epsilon/3$.

$$p(x_{e+f}) - p(x_e) = \sum_{i=e+2}^{e+f} (f(x_i) - f(x_{i-1}))^+ < \epsilon/3. \tag{3.3}$$

We have $y_{k+r} \leq x_{e+f}$, and $y > x$; therefore,

$$p(y) - p(y_{k+r}) > p(x) - p(x_{e+f}) - \epsilon/3.$$

Finally, we compute $p(y)$

$$\begin{aligned} p(y) &= \underbrace{p(y) - p(y_{k+r})}_{\geq p(x)-p(x_{e+f})-\epsilon/3} + \underbrace{p(y_{k+r}) - p(y_{k+1})}_{\geq 0} + \underbrace{p(y_{k+1})}_{\geq p(x_e)-\epsilon/3} \\ &> p(x) - p(x_{e+f}) + p(x_e) - 2\epsilon/3 \\ &> p(x) - p(x_{e+f}) + p(x_e) + p(x_{e+f}) - p(x_e) - \epsilon \qquad \text{(from (3.3))} \\ &= p(x) - \epsilon. \end{aligned}$$

□

If we choose ϵ to be the machine epsilon ϵ_{mach}, then the functions p and n will be increasing in this computer because if $x < y$ then $p(x) \leq p(y) + \epsilon_{\text{mach}} = p(y)$ and $n(x) \leq n(y) + \epsilon_{\text{mach}} = n(y)$. We now use Algorithm 3.2.1 and Theorem 3.2.1 to compute p and n in the following examples, where we will choose ϵ_{mach} to be 10^{-2}.

Example 3.2.2. Let f be the function defined by $f(x) = x^2$. We consider three domains of f; $[0, 1]$, $[-1, 0]$ and $[-1, 1]$.

1. If f is defined on $[0, 1]$, then f is increasing. Therefore, it suffices to choose $m = 0$ and we have $p(x) = f(x)$ and $n(x) = 0$ for all $x \in [0, 1]$.

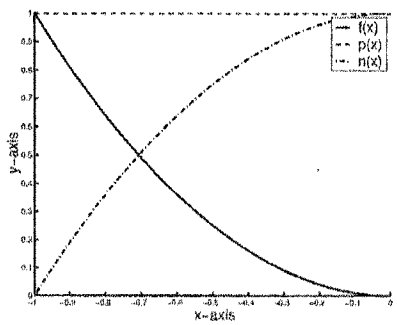
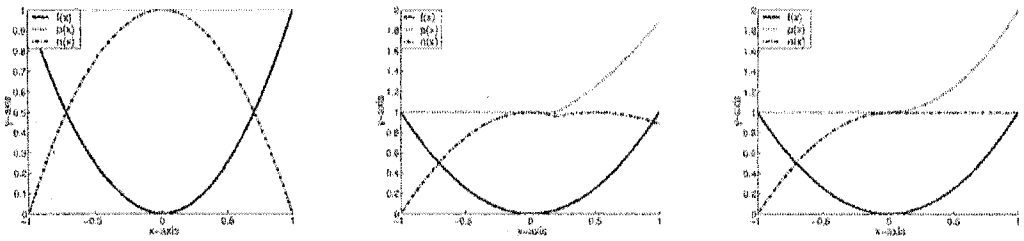


Figure 3.6: The functions f, p and n in part 2 of Example 3.2.2.

2. If f is defined on $[-1, 0]$, then f is decreasing. Hence, we choose $m = 0$ and we have $p(x) = f(-1) = 1$ and $n(x) = f(-1) - f(x) = 1 - f(x)$. We plot the functions f, p and n in Figure 3.6.



(a) m=0 (b) m=2 (c) m=10

Figure 3.7: The functions f, p and n in part 3 of Example 3.2.2.

3. Finally, we combine the two intervals to get a different situation. In this case, f is not monotone and hence, we do not expect $m = 0$ to be sufficient condition for the functions p and n to be increasing. We apply Algorithm 3.2.1 to this function with $m = 0, 2$ and 10 and plot the resulting function in Figure 3.7. We now compute m_ϵ to obtain two ϵ-increasing functions p and n. Since f is smooth, we have $f \in L[-1, 1]$ with a Lipschitz constant $C = \max|f'(x)| = 2$. Suppose that $\epsilon = \epsilon_{\text{mach}} = 10^{-2}$, then the functions p and n are ϵ-increasing if we choose m such that $C(b-a)/(m+1) < \epsilon/3$. Therefore, if we choose $m = 75$, then p and n are increasing in this machine and $f = p - n$. We plot the function $n(x)$ for $x \in [0, 1]$ in Figure 3.8 and notice that n is ϵ_1-increasing where $\epsilon_1 = 2 \times 10^{-4}$ which is much better than ϵ_{mach}-increasing.

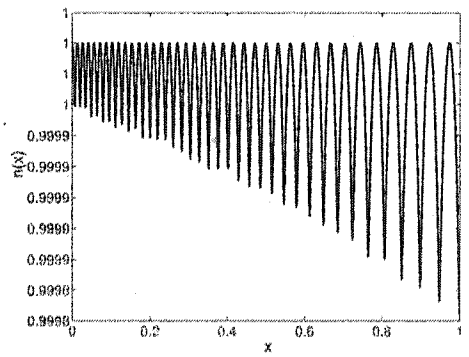
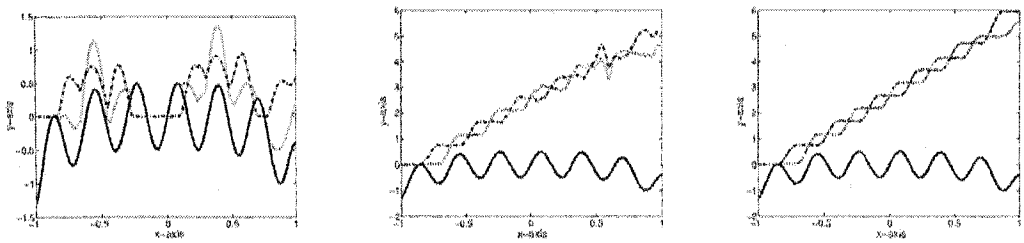


Figure 3.8: The function $n(x)$ in Example 3.2.2, where $m = 75$



(a) m=2 (b) m=20 (c) m=80

Figure 3.9: The functions f, p and n in Example 3.2.3.

We saw that $m = 75$ was enough for Example 3.2.2 to have two ϵ-increasing functions p and n. Nevertheless, this is not always the case. The function in the following example has more oscillations than $f(x) = x^2$.

Example 3.2.3. Let $f : [-1, 1] \to \mathbb{R}$ be defined as $f(x) = \sin(10x)\cos(10x) - x^3 \sin x$. This function is not monotone on $[-1, 1]$; moreover, it oscillates several times on $[-1, 1]$. We illustrate the behavior of p and n as m increases in Figure 3.9. If we want p and n to be ϵ-increasing then it suffices to choose m such that $(20 \times 2)/(m+1) < \epsilon/10$; therefore, the value $m = 4 \times 10^4$ is sufficient to guarantee that p and n are ϵ-increasing in the interval $[-1, 1]$. In figure 3.10, we plot the function p in the interval $[0.9, 1]$ and we can see that p is in fact ϵ_2-increasing where $\epsilon_2 = 10^{-6}$.

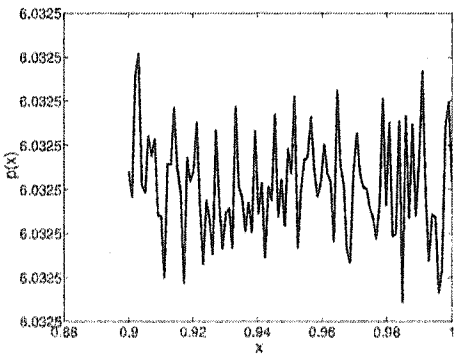
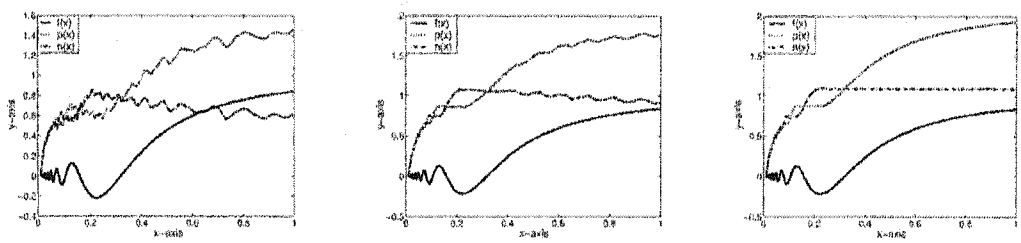


Figure 3.10: The function $p(x)$ in Example 3.2.3, where $m = 4 \times 10^4$

The above examples were smooth and the arclengths were not "very long". However, if we choose any function that is not of bounded variation and remove a small neighborhood of the point at which the function is not $\mathcal{C}^1$, we will get a $\mathcal{C}^1$ function and hence a function of bounded variation. For example if $f : [1,2] - \{\frac{\pi}{2}\} \to \mathbb{R}$ is defined by $f(x) = \sin(\tan x)$ then f is not of bounded variation. However, if we define f on $I := [1,2] - (\frac{\pi}{2} - \delta, \frac{\pi}{2} + \delta)$ for $\delta > 0$ then $f \in \mathbb{BV}(I)$. So, for every $\epsilon > 0$, there must be an $m_0 \in \mathbb{N}$ such that Algorithm 3.2.1 produces two ϵ-increasing functions p and n. As $\delta \to 0$, we have $m_0 \to \infty$. The following example illustrates this idea.

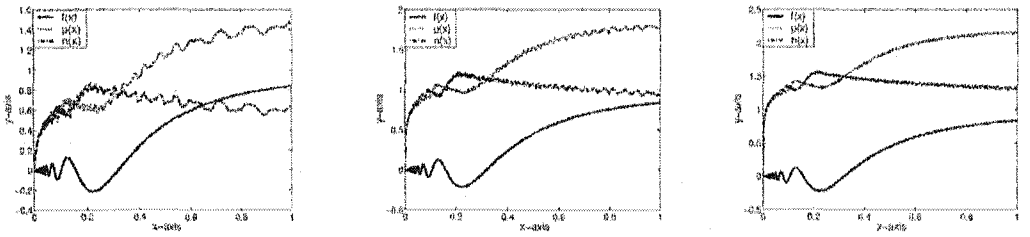


(a) m=100 (b) m=1000 (c) m=10000

Figure 3.11: The function $f(x) = x\sin(1/x)$, $p(x)$ and $n(x)$, with $\delta = 10^{-2}$

Example 3.2.4. Let $f : [\delta, 1] \to \mathbb{R}$ be defined by $f(x) = x\sin(1/x)$, where δ is a positive number less than 1. The function f is smooth and hence is of bounded variation. However, if we consider the interval $(0, 1]$, then f is not of bounded variation. Therefore, as δ approaches 0, the number m_ϵ goes to ∞. We choose δ to be 10^{-2} and 10^{-4}. For the first case where $\delta = 10^{-2}$, we see that choosing $m = 10^4$ gives two functions p and n that are ϵ-increasing when ϵ is a small number. We plot the functions f, p and n in Figure 3.11. However, for the second case where $\delta = 10^{-4}$, we see from Figure 3.12 that $n(0.2) \simeq 1.6$ and $n(1) \simeq 1.3$. So, the function n is not ϵ-increasing for $\epsilon \leq .3$. Now, if we want p and n to be ϵ_{mach}-increasing then we should choose m for each case. For the first case, we can compute $C = \max|f'(x)| = 10^4$; therefore, we can choose m_ϵ such that

$2 \times 10^4/(m+1) < 10^{-2}/10^2$, e.g., $m = 2 \times 10^8$. In the second case, we have $C = \max|f'(x)| = 10^8$ and therefore, we choose $m = 2 \times 10^{13}$.



(a) m=100 (b) m=1000 (c) m=10000

Figure 3.12: The function $f(x) = x\sin(1/x)$, $p(x)$ and $n(x)$, with $\delta = 10^{-4}$

From the previous examples, we notice that the choice of m_ϵ produces two functions p and n that are in fact ϵ_1-increasing where $\epsilon_1 \ll \epsilon$. The reason goes back to our choice of m_ϵ where we choose m_ϵ to satisfy the strong condition $C(b-a)/(m_\epsilon+1) < \epsilon$. This condition was given for the theoretical proof and we can choose m to be less than m_ϵ in our numerical experiments.

3.3 Bounded Variation in Two Dimensions

There are various approaches to the notion of variation for functions of N variables. These definitions and the way they are related are discussed in [1, 9]. To simplify notation, we consider the case of functions of two variables. All theorems obtained for $N = 2$ carry over to the case $N > 2$. Consider the function $f : [a_1, b_1] \times [a_2, b_2] \to \mathbb{R}$ and let $a_1 = t_0 < t_1 < \ldots < t_n = b_1, \quad a_2 = s_0 < s_1 < \ldots < s_n = b_2$ be partitions of the rectangle $[a_1, b_1] \times [a_2, b_2]$. Arzela [31] suggests that f is of bounded variation if the summation

$$\sum_{i=1}^{n} |f(t_i, s_i) - f(t_{i-1}, s_{i-1})|$$

has an upper bound. This approach will not help us to write f as the difference of two increasing functions. Therefore, we will consider another expression that is

more closely related to integration in two variables; this approach was defined by Hardy-Krause [6].

We start by generalizing some of the properties that we have discussed for one variable. Let $I_a^b := [a_1, b_1] \times [a_2, b_2]$ be the basic rectangle with $a = (a_1, a_2)$, $b = (b_1, b_2)$. We say that $x \leq y$ where $x = (x_1, x_2)$, $y = (y_1, y_2)$, if $x_1 \leq y_1$ and $x_2 \leq y_2$.

Let $\xi := \{t_i\}_{i=0}^m$ and $\eta := \{s_j\}_{j=0}^n$ be partitions of $[a_1, b_1]$ and $[a_2, b_2]$, respectively (i.e., $a_1 = t_0 < t_1 < \ldots < t_{m-1} < t_m = b_1$ and $a_2 = s_0 < s_1 < \ldots < s_{n-1} < s_n = b_2$) and let D_{ij} be the rectangle defined by $D_{ij} := [t_{i-1}, s_{j-1}] \times [t_i, s_j]$, then we use the notion

$$F(D_{ij}) := f(t_i, s_j) - f(t_{i-1}, s_j) - f(t_i, s_{j-1}) + f(t_{i-1}, s_{j-1}),$$

to denote the double difference which involves all four vertices of D_{ij}.

Definition 3.3.1. We define the variation of $f : I_a^b \to \mathbb{R}$ over the partition $\tau := (\xi, \eta)$ to be

$$F_\tau(I_a^b) := \sum_{i=1}^{m} \sum_{j=1}^{n} |F(D_{ij})|.$$

Figure 3.3 illustrates the way $F_\tau(I_a^b)$ is defined for $m = n = 2$. If we have two partitions τ_1, τ_2 of I_a^b and τ_1 can be obtained by adding more partitions to τ_2, we say that τ_1 is finer than τ_2. We now define a special case.

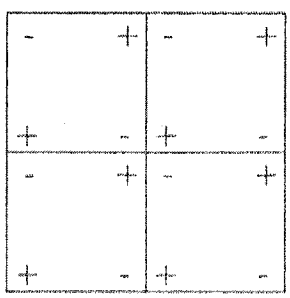


Figure 3.13: $F_\tau(I_a^b)$ in 2-dimensions where $m = n = 2$.

Definition 3.3.2. If $\tau_1 := (\xi_1, \eta_1)$ and $\tau_2 := (\xi_2, \eta_2)$ are two partitions of I_a^b then we say τ_1 is *totally finer* than τ_2 if $\xi_2 \subseteq \xi_1$ and $\eta_2 \subseteq \eta_1$.

Proposition 3.3.1. *If $\tau_1 := (\xi_1, \eta_1)$ is totally finer than $\tau_2 := (\xi_2, \eta_2)$, then*

$$F_{\tau_1}(I_a^b) \geq F_{\tau_2}(I_a^b).$$

Proof. We have τ_1 is totally finer than τ_2 which implies that $\xi_2 \subseteq \xi_1$ and $\eta_2 \subseteq \eta_1$. This means that ξ_1 and η_1 can be obtained from ξ_2 and η_2, respectively, by adding a finite number of points. So it suffices to prove, without loss of generality, that if we add one point $\{t^*\}$ to ξ_2 then we have $F_{\tau_1}(I_a^b) \geq F_{\tau_2}(I_a^b)$ and then use induction. Suppose that $\xi_2 : a_1 = t_0 < t_1 < \ldots < t_m$ and let $\xi_1 : a_1 = t_0 < \ldots < t_{k-1} < t^* < t_k < \ldots < t_m = b_1$, and $\eta_1 = \eta_2 : a_2 = s_0 < \ldots < s_n = b_2$. Now

$$\begin{aligned}
F_{\tau_2}(I_a^b) &= \sum_{i=1}^{m}\sum_{j=1}^{n} |F(D_{ij})|, \\
&= \sum_{\substack{i=1 \\ i\neq k}}^{m}\sum_{j=1}^{n} |F(D_{ij})| + \sum_{j=1}^{n} |F(D_{kj})|, \\
&\leq \sum_{\substack{i=1 \\ i\neq k}}^{m}\sum_{j=1}^{n} |F(D_{ij})| + \sum_{j=1}^{n} |f(t_k, s_j) - f(t^*, s_j) - f(t_k, s_{j-1}) + f(t^*, s_{j-1})| \\
&\quad + \sum_{j=1}^{n} |f(t^*, s_j) - f(t_{k-1}, s_j) - f(t^*, s_{j-1}) + f(t_{k-1}, s_{j-1})|, \\
&= F_{\tau_1}(I_a^b).
\end{aligned}$$

□

We now give the definition of functions of bounded variation of two variables and we will use the notion

$$\Delta_{1i} f := f(t_i, a_2) - f(t_{i-1}, a_2),$$

$$\Delta_{2j} f := f(a_1, s_j) - f(a_1, s_{j-1}).$$

Definition 3.3.3 (Bounded Variation). Let $f : I_a^b \to \mathbb{R}$ be a function and $\tau := (\xi, \eta)$ be a partition of I_a^b. We define the Vitali variation [21] of f to be

$$V_2(f, I_a^b) = \sup_{\tau} F_\tau(I_a^b),$$

and we define the Jordan variation of the functions $f_1 := f(., a_2)$ and $f_2 := f(a_1, .)$ of one variable to be

$$V(f_1; a_1, b_1) = \sup_{\xi} \sum_{i=1}^{m} |\Delta_{1i} f|,$$
$$V(f_2; a_2, b_2) = \sup_{\eta} \sum_{j=1}^{n} |\Delta_{2j} f|.$$

We finally define the total variation of f by

$$V(f, I_a^b) = V(f_1; a_1, b_1) + V(f_2; a_2, b_2) + V_2(f, I_a^b),$$

and we say that f is a function of bounded variation if $V(f, I_a^b) < \infty$. We denote the set of functions of bounded variation by $\mathbb{BV}(I_a^b)$.

We will make use of these concepts for the CEA in higher dimensions. Our goal is to prove that each function of bounded variation can be written as the difference of two increasing functions.

Definition 3.3.4. We say that $f : I_a^b \to \mathbb{R}$ is an increasing function if $f(x_1, x_2) \leq f(y_1, y_2)$ for all $x_1 \leq y_1$ and $x_2 \leq y_2$.

Now we define the positive and negative parts of $V(f, I_a^b)$ as follows

$$P(f, I_a^b) := \sup_{\tau} \sum_{i,j} F(D_{ij})^+ + \sup_{\xi} \sum_{i=1}^{m} (\Delta_{1i} f)^+ + \sup_{\eta} \sum_{j=1}^{n} (\Delta_{2j} f)^+,$$
$$N(f, I_a^b) := \sup_{\tau} \sum_{i,j} F(D_{ij})^- + \sup_{\xi} \sum_{i=1}^{m} (\Delta_{1i} f)^- + \sup_{\eta} \sum_{j=1}^{n} (\Delta_{2j} f)^-.$$

Replacing b by x and letting x vary we get P and N as functions of x. These functions have a special property that will be shown in the following lemma.

Lemma 3.3.1. *If $f \in \mathbb{BV}(I_a^b)$ and we consider $P(f, I_a^x)$ and $N(f, I_a^x)$ as functions of $x = (x_1, x_2)$, then P and N are increasing functions.*

Proof. Suppose that $x < y$ where $x, y \in \mathbb{R}^2$, then we have three cases:

1. $x_1 < y_1$ and $x_2 = y_2$.

2. $x_1 = y_1$ and $x_2 < y_2$.

3. $x_1 < y_1$ and $x_2 < y_2$.

We will prove the first case for the function $P(f, I_a^x)$. Let $\tau_1 = (\xi_1, \eta_1)$ be a partition

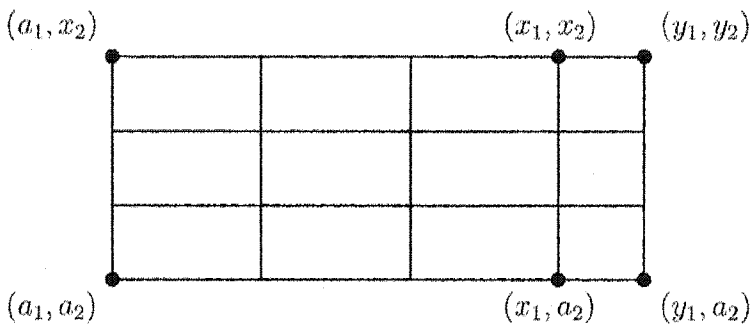


Figure 3.14: The function $V(f, I_a^x)$.

of $[a_1, x_1] \times [a_2, x_2]$ where $\xi_1 = \{t_i\}_{i=1}^{m_{\xi_1}}$ and $\eta_1 = \{s_j\}_{j=1}^{m_{\eta_1}}$, and $\tau_2 = (\xi_2, \eta_2)$ be a partition of $[a_1, y_1] \times [a_2, y_2]$ such that $\xi_2 := \xi_1 \cup \{y_1\}$ and $\eta_2 := \eta_1$ (see Figure 3.14). Then we have

$$\begin{aligned}
P_{\tau_1}(f, I_a^x) &= \sum_{i=1}^{m_{\xi_1}} \sum_{j=1}^{n_{\eta_1}} F(D_{ij})^+ + \sum_{i=1}^{m_{\xi_1}} (\Delta_{1i} f)^+ + \sum_{j=1}^{n_{\eta_1}} (\Delta_{2j} f)^+ \\
&\leq \sum_{i=1}^{m_{\xi_1}} \sum_{j=1}^{n_{\eta_1}} F(D_{ij})^+ + \sum_{i=1}^{m_{\xi_1}} (\Delta_{1i} f)^+ + \sum_{j=1}^{n_{\eta_1}} (\Delta_{2j} f)^+ + \sum_{j=1}^{n_{\eta_1}} F(D_{(m_{\xi_1}+1)j})^+ \\
&\quad + (\Delta_{1(m_{\xi_1}+1)} f)^+ \\
&= \sum_{i=1}^{m_{\xi_1}+1} \sum_{j=1}^{n_{\eta_1}} F(D_{ij})^+ + \sum_{i=1}^{m_{\xi_1}+1} (\Delta_{1i} f)^+ + \sum_{j=1}^{n_{\eta_1}} (\Delta_{2j} f)^+ \\
&= P_{\tau_2}(f, I_a^y)
\end{aligned}$$

Now we take the supremum on both sides of the inequality for all partitions τ_1 and we get

$$P(f, I_a^x) \leq P(f, I_a^y).$$

The second case can be proven in the same way we proved case 1. The last case follows immediately from case 1 and case 2. The same argument can be made to prove that $N(f, I_a^x)$ is an increasing function. □

We now generalize the Jordan decomposition theorem to functions in two variables and use the previous lemma to prove this theorem.

Theorem 3.3.1. *If f is a function of bounded variation on I_a^b then f can be written as the difference of two increasing functions*

$$f(x) = p(x) - n(x), \quad x \in I_a^b.$$

Proof. We have (by cancellation)

$$f(x_1, x_2) - f(x_1, a_2) - f(a_1, x_2) + f(a_1, a_2) = \sup_{\tau} \sum_{i,j} F(D_{ij}).$$

First, we consider the two functions of one variable $f(x_1, a_2)$ and $f(a_1, x_2)$. For the first function we have

$$f(x_1, a_2) - f(a_1, a_2) = \sup_{\xi} \sum_{i} \Delta_{1i} f.$$

If we let $p_1(x_1, a_2) = \sup_\xi \sum_i (\Delta_{1i} f)^+$ and $n_1(x_1, a_2) = \sup_\xi \sum_i (\Delta_{1i} f)^-$ then $f(x_1, a_2) = f(a_1, a_2) + p_1(x_1, a_2) - n_1(x_1, a_2)$ where p_1 and n_1 are increasing functions of one variable x_1. Similarly, if we let $p_2(a_1, x_2) = \sup_\eta \sum_j (\Delta_{2j} f)^+$ and $n_2(a_1, x_2) = \sup_\eta \sum_j (\Delta_{2j} f)^-$ then p_2 and n_2 are increasing functions and

$$f(a_1, x_2) = f(a_1, a_2) + p_2(a_1, x_2) - n_2(a_1, x_2).$$

Finally, $\sup_\tau \sum_{i,j} F(D_{ij})^+$ and $\sup_\tau \sum_{i,j} F(D_{ij})^-$ are increasing functions. Hence

$$f(x_1,x_2) = f(a_1,a_2) + [p_1(x_1,a_2) + p_2(a_1,x_2) + \sup_\tau \sum_{i,j} F(D_{ij})^+]$$
$$- [n_1(x_1,a_2) + n_2(a_1,x_2) + \sup_\tau \sum_{i,j} F(D_{ij})^-],$$

which means $f(x) = f(a_1,a_2) + P(f,I_a^x) - N(f,I_a^x)$. □

In the following section we exploit this proof of the theorem to write an algorithm to compute p and n. As we have noticed in one dimension, these two functions cannot be increasing on I_a^b. Therefore, we will use the same argument as in one dimension and show that they are in fact ϵ-increasing.

3.4 Decomposition Algorithm and Examples

In this section we will introduce *the decomposition algorithm for two variables* that approximates p and n with respect to the uniform partition $\tau := (\xi,\eta)$ of I_a^x where $\xi = \{a_1 + i(x_1 - a_1)/(m_1+1)\}_{i=0}^{m_1+1}$ and $\eta = \{a_2 + j(x_2 - a_2)/(m_2+1)\}_{j=0}^{m_2+1}$. For the algorithm, we choose m_i to be the number of points between a_i and x_i for $i = 1,2$. Then we approximate the functions $P_f[a,x]$ and $N_f[a,x]$. It is not hard to construct an example to show that p and n are not increasing even for smooth functions. Therefore, we will again need the definition of ϵ-increasing functions in two dimensions. We now write our algorithm and we will use different notion for a and x. We write, for simplicity, $\mathbf{a} = (a,b)$ and $\mathbf{x} = (x,y)$.

Algorithm 3.4.1 (Decomposition Algorithm). INPUT: $\mathbf{a} = (a,b)$, f and $\mathbf{x} = (x,y)$.

OUTPUT: p and n evaluated at x.

choose m_1 (number of points between a_1 and x_1).

choose m_2 (number of points between a_2 and x_2).

$dx := (x - a)/(m_1 + 1)$.

$dy := (y - b)/(m_2 + 1)$.

for $i = 0 : m_1 + 1$

$x_i := a + idx$.

for $j = 0 : m_2 + 1$

$y_j := b + jdy$.

$p = f(a, b)$.

$n = 0$.

for $i = 1 : m_1 + 1$

If $s = \Delta_{1i} f \geq 0$

$p = p + s$.

else

$n = n + s$.

for $j = 1 : m_2 + 1$

If $s = \Delta_{2j} f \geq 0$

$p = p + s$.

else

$n = n + s$.

for $i = 1 : m_1 + 1$

for $j = 1 : m_2 + 1$

If $s = F(D_{ij}) \geq 0$

$p = p + s$.

else

$n = n + s$.

We now define the concept of ϵ-increasing and the definition of Lipschitz condition in two dimensions.

Definition 3.4.1 (ϵ-increasing). We say that the function $f : I_a^b \longrightarrow \mathbb{R}$ is ϵ-increasing, where ϵ is a positive number, if $f(x) \leq f(y) + \epsilon$ for all $x < y$ and $x, y \in I_a^b$.

Definition 3.4.2 (Lipschitz Condition). [2] A function $f : I_a^b \longrightarrow \mathbb{R}$ is a Lipschitz function on I_a^b with a Lipschitz constant C if there is a constant C such that

$$|f(x) - f(y)| \leq C\|x - y\| \quad \forall x, y \in I_a^b,$$

for some norm. We denote all Lipschitz functions on I_a^b by $Lip(I_a^b)$.

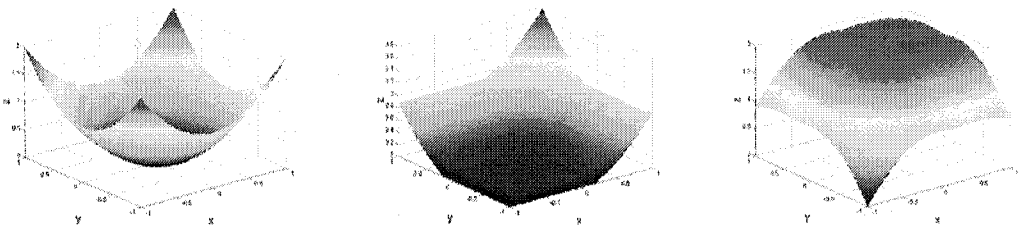
We have discussed in Proposition 3.1.2 that all functions in $Lip[a, b]$ are of bounded variation on $[a, b]$. In fact, this proposition can be generalized in two dimensions.

Proposition 3.4.1. *If $f : I_a^b \longrightarrow \mathbb{R}$ is a Lipschitz function then f is of bounded variation on I_a^b.*

Proof for this proposition is straightforward and is based on the Lipschitz condition.

We now state an analogous of Theorem 3.2.1 for the case where we have two variables. This theorem can be proven in a similar way we proved 3.2.1; nevertheless, we should consider three cases, where f oscillates in the x direction, in the y direction and finally in both directions. We will not write details for the proof of this theorem since it requires tedious computations and is similar in concept to the 1-dimensional case.

Theorem 3.4.1. *If $f : I_a^b \longrightarrow \mathbb{R}$ is a Lipschitz function with a Lipschitz constant C, then for all $\epsilon > 0$ there exists $m_1 := m_1(\epsilon)$ and $m_2 := m_2(\epsilon) \in \mathbb{N}$ such that p and n in Algorithm 3.4.1 are ϵ-increasing.*

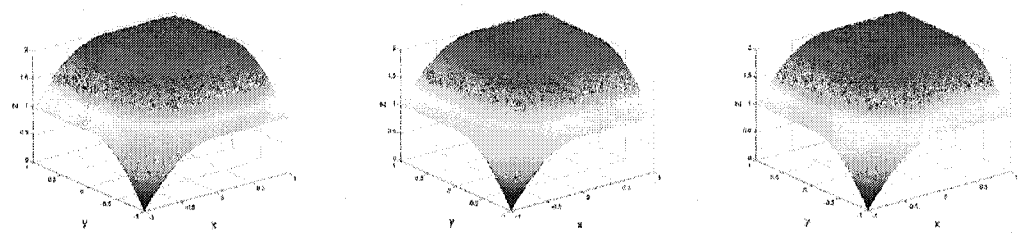


(a) The function f (b) The function p (c) The function n

Figure 3.15: The functions f, p and n in Example 3.4.1 where $m = 2$.

We conclude this section with some examples that illustrate the usage of Algorithm 3.4.1.

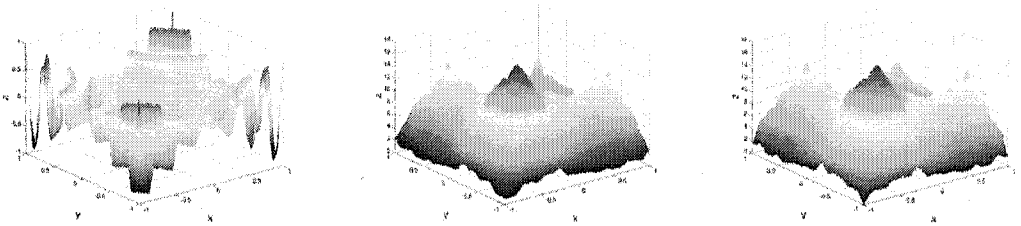
Example 3.4.1. Let $f : [-1, 1] \times [-1, 1] \to \mathbb{R}$ be defined by $f(x) = x^2 + y^2$. We can see immediately that this function is not increasing nor decreasing. We apply Algorithm 3.4.1 to this function and plot the functions f, p and n in Figure 3.15 where $m = m_1 := m_2 = 2$. Since our function is smooth and has only one local minimum, we do not expect that we need a very big value for m in order for p and n to be ϵ-increasing for $\epsilon = 10^{-2}$. We plot the function p for the case where $m = 4, 6$ and 8 in Figure 3.16.



(a) $m = 4$ (b) $m = 6$ (c) $m = 8$

Figure 3.16: The function p in Example 3.4.1

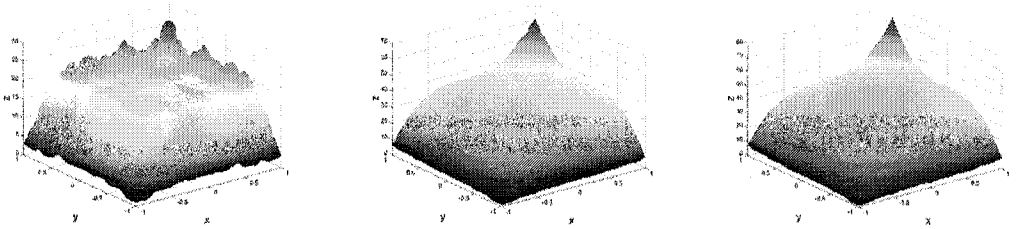
The function f in the previous example has only one minimum and therefore, a small value for m can be sufficient. In the following example we choose another function that involves the functions sine and cosine.



(a) The function f (b) The function p (c) The function n

Figure 3.17: The functions f, p and n in Example 3.4.2 where $m = 4$.

Example 3.4.2. Let $f : [-1,1] \times [-1,1] \longrightarrow \mathbb{R}$ be defined as $f(x) = xy\sin(20xy)$. This function is not monotone on $[-1,1]$; moreover, it oscillates several times on $[-1,1]$. We plot the functions f, p and n in Figure 3.17 where $m = 4$. Then we illustrate the behavior of p as m increases in Figure 3.18. It seems that p is increasing on $[-1,1] \times [-1,1]$; however, if we take a closer look at p in a neighborhood of $(0,0)$ for $m = 40$, we notice that p is not increasing on this region. Nevertheless, it is ϵ-increasing for some $\epsilon > 0$. We plot the function p in Figure 3.19.



(a) $m = 8$ (b) $m = 20$ (c) $m = 40$

Figure 3.18: The function p in Example 3.4.2

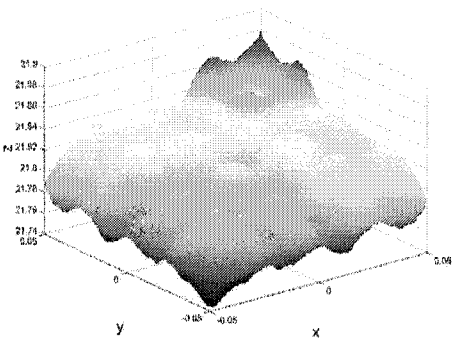


Figure 3.19: The function p in a neighborhood of (0,0) where $m = 40$.

CHAPTER 4

NELDER-MEAD ALGORITHM

The Nelder-Mead Algorithm (NMA) [50] is one of the direct search methods which are easy to program and do not require derivatives, [58, 63]. Since its publication in 1965, the Nelder-Mead algorithm has become one of the widely used methods for nonlinear unconstrained optimization. For example, the Matlab function `fmins` uses the Nelder-Mead algorithm. Moreover, this algorithm appears in *Numerical Recipes* [51].

The Nelder-Mead method attempts to find a local minimum for a scalar-valued nonlinear function f of n real variables using only functions valued without any derivative information (explicit or implicit). More introductory information about NMA can be found in [40, 46, 52]. Section 4.1 describes the way this algorithm works. Then Section 4.2 presents some convergence results for NMA. In Section 4.3, some examples are given where this algorithm converges to a local minimum. Finally, we conclude this chapter by other examples in Section 4.4 where this algorithm does not converge to a minimizer.

4.1 The Algorithm

The Nelder-Mead algorithm starts with a nondegenerate *simplex*, a geometric figure in n dimensions of nonzero volume that is the convex hull of $n+1$ vertices, in such a way that this simplex is "close" to the minimizer [36]. In [33], it was enough to chose one vertex, say x_1, which is an initial approximation of the minimum, and

then define the following auxiliary values:

$$p_n = \frac{\sqrt{n+1} - 1 + n}{n\sqrt{2}} M,$$
$$q_n = \frac{\sqrt{n+1} - 1}{n\sqrt{2}} M,$$

where M is a scaling factor. The $n+1$ vertices are given by:

$$x_1 = (a_1, a_2, \ldots, a_n)^T,$$
$$x_2 = (p_n + a_1, q_n + a_2, \ldots, q_n + a_n)^T,$$
$$x_3 = (q_n + a_1, p_n + a_2, \ldots, q_n + a_n)^T,$$
$$\vdots$$
$$x_{n+1} = (q_n + a_1, q_n + a_2, \ldots, p_n + a_n)^T.$$

In this algorithm the vertices $\{x_j\}_{j=1}^{n+1}$ are resorted according to the objective function values as follows:

$$f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1}). \tag{4.1}$$

The vertex x_1 is called the best vertex and x_{n+1} is the worst. The algorithm attempts to replace the worst vertex x_{n+1} with a new point of the form:

$$x(\mu) = (1 + \mu)\bar{x} - \mu x_{n+1}, \tag{4.2}$$

where $\bar{x}$ is defined by the barycenter of the face opposite x_{n+1}:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{4.3}$$

and the value of μ is selected from a sequence:

$$-1 < \mu_{ic} < 0 < \mu_{oc} < \mu_r < \mu_e.$$

A typical sequence [40] of candidate values for μ is

$$\{\mu_r, \mu_e, \mu_{oc}, \mu_{ic}\} = \{1, 2, 1/2, -1/2\}.$$

Each iteration of NMA begins with a simplex which is specified by its $n+1$ vertices and the associated function values. One or more test points are computed, along with their function values, and the iteration terminates with a new simplex such that the function values at its vertices satisfy some form of descent condition compared to the previous simplex. The next algorithm shows how the Nelder-Mead method can find a local minimum [36, 40].

Algorithm 4.1.1 (Nelder-Mead Algorithm).

Evaluate f at the vertices of S and sort the vertices of S so that (4.1) holds.

1. Compute $\bar{x}$ from (4.3), $x_r = x(\mu_r)$ from (4.2), and $f_r = f(x_r)$.

2. **Reflect:** If $f(x_1) \leq f_r < f(x_n)$, replace x_{n+1} with x_r and go to step 7.

3. **Expand:** If $f_r < f(x_1)$ then compute $f_e = f(x(\mu_e)) = f(x_e)$.
   If $f_e < f_r$, replace x_{n+1} with x_e; otherwise replace x_{n+1} with x_r.
   Go to step 7.

4. **Outside Contraction:** If $f(x_n) \leq f_r < f(x_{n+1})$, compute $f_c = f(x_{oc})$.
   If $f_c \leq f_r$ replace x_{n+1} with x_{oc} and go to step 7; otherwise go to step 6.

5. **Inside Contraction:** If $f_r \geq f(x_{n+1})$, compute $f_c = f(x_{ic})$.
   If $f_c < f(x_{n+1})$, replace x_{n+1} with x_{ic} and go to step 7;
   otherwise go to step 6.

6. **Shrink:** For $2 \leq i \leq n+1$: set $x_i = (x_i + x_1)/2$ and compute $f(x_i)$.

7. **Sort:** Sort the vertices of S so that (4.1) holds.

Repeat until a stopping criterion holds.

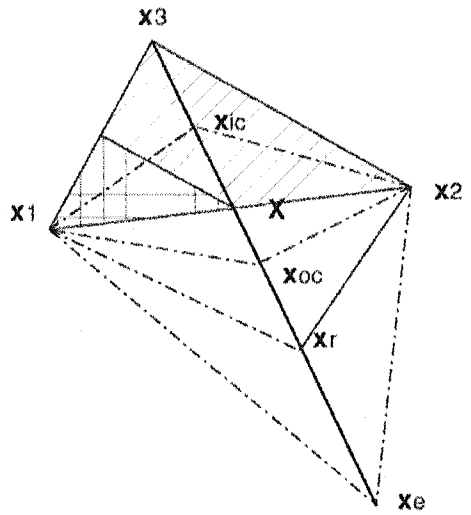


Figure 4.1: Reflection, expansion, and contraction for NMA

Several choices have been made as a stopping criterion for Algorithm 4.1.1. For example, [36] suggests either $f(x_{n+1}) - f(x_1) < \tau$ for a given $\tau > 0$ or a user-specified number of function evaluations has been expended. While [33] suggests

$$\left(\sum_{i=1}^{n+1} \frac{(f(x_i) - f(\bar{x}))^2}{2} \right)^{\frac{1}{2}} < \tau.$$

Figure 4.1 is an illustration of the options in two dimensions.

4.2 Convergence Property

There are only limited convergence results for the NMA in a restricted class of problems in one and two dimensions, which assume that f is strictly convex among other things. Let Δ be the simplex $x_1, \ldots, x_{n+1}$, then we define the diameter of Δ as

$$\text{diam}(\Delta) = \max_{i \neq j} ||x_i - x_j||_2.$$

Definition 4.2.1 (Strict Convexity). The function f is strictly convex on $\mathbb{R}^n$ if, for every pair of points y, z with $y \neq z$ and every λ satisfying $0 < \lambda < 1$,

$$f(\lambda y + (1 - \lambda)z) < \lambda f(y) + (1 - \lambda) f(z).$$

Let $\mathcal{CB}$ be the set of all functions that are strictly convex and bounded below. We now consider the case where $\{\mu_r, \mu_e, \mu_{oc}, \mu_{ic}\} = \{1, 2, 1/2, -1/2\}$.

In one dimensions [40], if $f \in \mathcal{CB}$ then NMA will converge to a local minimum $x_{\min}$. In other words both end points of the NM interval converge to $x_{\min}$. The convergence in this case is M-step linear, i.e., there are integers M and K such that

$$\text{diam}(\Delta_{k+M}) \leq \frac{1}{2}\text{diam}(\Delta_k) \quad \forall k \geq K.$$

In two dimension [40]; however, this is not the case . If $f : \mathbb{R}^2 \to \mathbb{R}$ is in $\mathcal{CB}$ and the NMA is applied to f beginning with an initial simplex Δ_0, then the simplices $\{\Delta_k\}$ will collapse to a point, i.e.,

$$\lim_{k\to\infty} \text{diam}(\Delta_k) = 0.$$

Even though NMA will converge to a point for all functions $f : \mathbb{R}^2 \to \mathbb{R}$ and $f \in \mathcal{CB}$, this point is not guaranteed to be a local minimum as we will discuss in Section 4.4.

4.3 Examples

We present two examples in $\mathbb{R}^2$. In the first example, NMA will converge to $x_{\min}$ even if the initial point x_1, is far from the minimizer. In the second example, x_1 must be chosen so that it is close to $x_{\min}$.

Example 4.3.1 (Rosenbrock's Function). Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the Rosenbrock's function which is defined by:

$$f(x, y) := 100(y - x^2)^2 + (1 - x)^2.$$

This function is a classic optimization problem. The global optimum is inside a long, narrow, parabolic shaped flat valley. The convergence to the global optimum,

iteration	x_1	x_2	x_3
0	(.5, .5)	(.69318, .55176)	(.55176, .69318)
1	(.64142, .35857)	(.69318, .55176)	(.5, .5)
2	(.64142, .35857)	(.70912, .44396)	(.69318, .55176)
⋮	⋮	⋮	⋮
47	(1.00004, 1.00008)	(.99998, .99995)	(.99996, .99994)

Table 4.1: NMA for Example 4.3.1 with $x_1 = (.5, .5)$

$x_{\min} := (1, 1)$, is difficult and hence this problem has been repeatedly used in assess the performance of optimization algorithms. We choose several values for x_1 and fix the tolerance to be 10^{-16}. The first choice for x_1 is (-1.2,1). With this point the algorithm will converge to the minimum in 79 iterations. The simplex at iteration 79 is $x_1 = (.99998, .99997), x_2 = (1.00001, 1.00002), x_3 = (1.00007, 1.00016)$. The second choice for x_1 is (.5,.5), then this algorithm will converge after 47 iterations. The simplex in this case is $x_1 = (1.00004, 1.00008), x_2 = (.99998, .99995), x_3 = (.99996, .99994)$. In fact, NMA converges to the minimum value even for $x_1 = (100, 100)$. Figure 4.2 illustrates the plot of the Rosenbrock's function, and Table 4.1 gives the values of some simplices for the case $x_1 = (.5, .5)$.

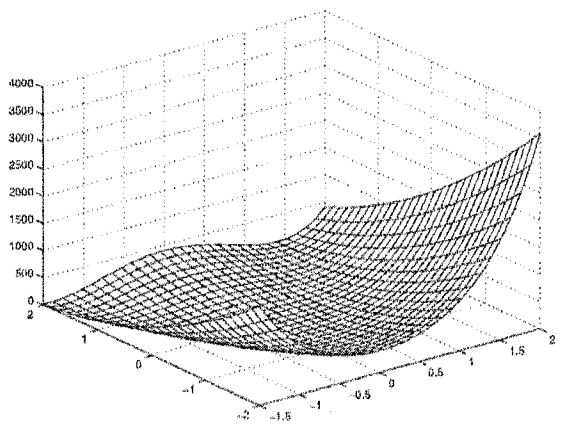


Figure 4.2: Rosenbrock's Function

Example 4.3.2. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be the function

$$f(x, y) := xe^{-x^2-y^2}.$$

This function has its minimum at the point $x_{\min} := (-\frac{1}{\sqrt{2}}, 0)$. We choose two values for x_1 and fix the tolerance to be 10^{-16}. If we choose x_1 to be (-1,1) then the algorithm will converge to the minimum in 35 iterations. The simplex at iteration 35 is $x_1 = (-0.70714, 0.00003), x_2 = (-0.70707, -0.00012), x_3 = (-0.70719, -0.00004)$. On the other hand, if we choose x_1 to be (5,4), then NMA will converge in the first step to a nonminimizer point. The simplex in this case is $x_1 = (5, 4), x_2 = (6.93185, 4.51763), x_3 = (5.51763, 5.93185)$, and

$$\left(\sum_{i=1}^{3} \frac{(f(x_i) - f(\bar{x}))^2}{2} \right)^{\frac{1}{2}} = 3 \times 10^{-35}.$$

The reason that NMA does not converge goes back to the shape of the function, this the function is "flat", i.e., the function values at the vertices of the first simplex are very close. Figure 4.3 illustrates the plot of the function f, and the iterations of NMA starting at $x_1 = (-3, 3)$.

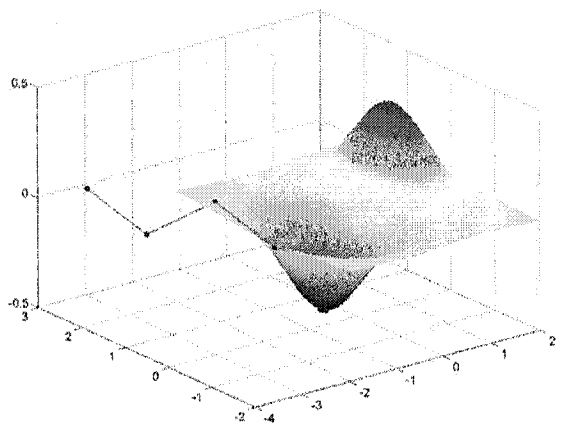


Figure 4.3: NMA iterations for the function in Example 4.3.2

4.4 Examples Where NMA Converges to a Nonstationary Point

In this section, we will consider the problem:

$$\min_{\mathbf{x} \in \mathbb{R}^2} f(\mathbf{x}) \tag{4.4}$$

i.e., $N = 2$, and give some examples where NMA converges to a nonminimizing point. The first example was given by Mckinnon [46], where a family of strictly convex functions in two dimensions have the property that all vertices in the Nelder-Mead method converge to a nonminimizing point. All functions cause the Nelder-Mead method to apply the inside contraction step repeatedly with the best vertex remaining fixed.

Example 4.4.1. Let f be defined as:

$$f(x,y) = \begin{cases} \theta\phi|x|^\tau + y + y^2, & x \leq 0, \\ \theta x^\tau + y + y^2, & x > 0, \end{cases}$$

and consider the parameter sets:

$$(\tau, \theta, \phi) = \begin{cases} (3, 6, 400), \\ (2, 6, 60), \\ (1, 15, 10). \end{cases}$$

The initial simplex is:

$$x_1 = (1,1)^T, x_2 = (\lambda_+, \lambda_-)^T, x_3 = (0,0)^T \text{ where } \lambda_\pm = (1 \pm \sqrt{33})/8.$$

With this data, the Nelder-Mead iteration will converge to the origin, which is not a critical point for f. In this case, the algorithm will repeat inside contractions that leave the best point (which is not a minimizer) unchanged. We compute some values for the case $(\tau, \theta, \phi) = (1, 15, 10)$ in Table 4.2 and we illustrate the behavior of NMA for the same case in Figure 4.4.

iteration	x_1	x_2	x_3
0	(0,0)	(.8430703309, -.5930703309)	(1,1)
1	(0,0)	(.7107675827, .3517324173)	(.8430703309, -.5930703309)
2	(0,0)	(.5992270611, -.2086020611)	(.7107675827, .3517324173)
3	(0,0)	(.5051905566, .1237156934)	(.5992270611, -.2086020611)
4	(0,0)	(.4259111697, -.0733721073)	(.5051905566, .1237156934)
5	(0,0)	(.3590730707, .0435148199)	(.4259111697, -.0733721073)

Table 4.2: NMA for Example 4.4.1 where $(\tau, \theta, \phi) = (1, 15, 10)$

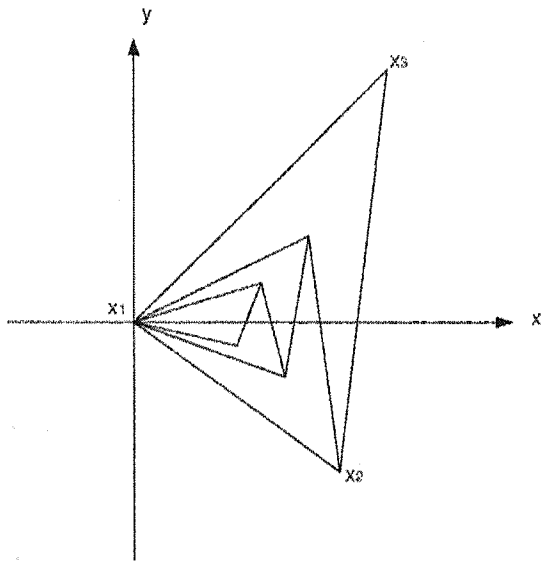


Figure 4.4: NMA for Example 4.4.1 where $(\tau, \theta, \phi) = (1, 15, 10)$.

However, if we start with a different set of vertices, e.g., $x_1 := (1, 1), x_2 := (1/4, -1/2)$ and $x_3 := (0, 0)$ then NMA will converge to the local minimum (0,-1/2).

It is also possible to construct examples of nonconvex differentiable functions for which the the Nelder-Mead method converges by repeated contractions to a simplex , none of whose vertices are a stationary point [46]. An example of this case follows:

Example 4.4.2. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined as:

$$f(x, y) = x^2 - y(y - 2),$$

with initial simplex $(1, 0), (0, -3), (0, 3)$.

Then NMA converges to the simplex (0,0), (0,3), (0,-3), where non of the vertices of this simplex is a minimizer. Table 4.3 gives some values for the first 5 iterations and we can see that x_1 and x_2 are fixed and x_3 takes the value $(1/2^k, 0)$ where k is the iteration number. Figure 4.5 explains the behavior of the NMA.

iteration	x_1	x_2	x_3
0	(0,-3)	(0,3)	(1,0)
1	(0,-3)	(0,3)	(1/2,0)
2	(0,-3)	(0,3)	(1/4,0)
3	(0,-3)	(0,3)	(1/8,0)
4	(0,-3)	(0,3)	(1/16,0)
5	(0,-3)	(0,3)	(1/32,0)

Table 4.3: NMA for Example 4.4.2

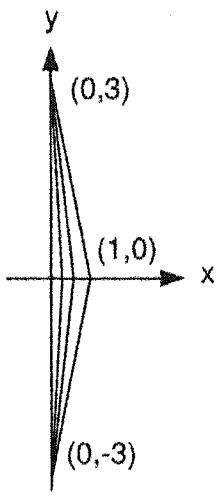


Figure 4.5: NMA for Example 4.4.2

These examples are presented to show that this algorithm can not always converge to a local minimizer; however, in both examples, the initial simplex was chosen carefully so that NMA does not converge. If we change the choice of the initial simplex, the NMA will converge in the first example to the global minimum and diverge in the second example, because the function in the second example does not have a local minimum. Therefore, these examples are very special cases and does not reflect the general behavior of the NMA.

CHAPTER 5

HOPF BIFURCATION

We consider the dynamical system $\dot{x} = f(x, \mu)$, where $x \in \mathbb{R}^n$, $\mu \in \mathbb{R}^m$ and $f : \mathbb{R}^{n+m} \to \mathbb{R}^n$ is continuously differentiable. The stability of the stationary solution $(x(\mu), \mu)$ of $f(x, \mu) = 0$ changes at a Hopf bifurcation point, which can be characterized by computing the eigenvalues of $f_x(x(\mu), \mu)$. In Section 5.1 some properties of dynamical systems are given to introduce the definition of Hopf bifurcation point. In Section 5.2 we describe how to locate Hopf bifurcation points. We conclude this chapter with an example which illustrates the computation of those points.

5.1 General Information about Hopf Bifurcation

Consider an autonomous set of ordinary differential equations of the form

$$\frac{dx_i}{dt} = f_i(x_1, \ldots, x_n, \mu_1, \ldots, \mu_m), \qquad i = 1, 2, \ldots, n,$$

which can be written as:

$$\dot{x} = f(x, \mu), \tag{5.1}$$

where $x \in \mathbb{R}^n$, $\mu \in \mathbb{R}^m$ and $f : \mathbb{R}^{n+m} \to \mathbb{R}^n$ is continuously differentiable. In order to define Hopf bifurcation, we need to define first the critical points of (5.1) and their stability.

Definition 5.1.1 (Critical Point). [30] The point (x_0, μ_0) is said to be a critical point of (5.1) if $f(x_0, \mu_0) = 0$; (x_0, μ_0) is also called a stationary point.

If special conditions are satisfied by f near the critical point (x_0, μ_0), then, locally, $f(x, \mu) = 0$ can be written as $f(x(\mu), \mu) = 0$ in a neighborhood of (x_0, μ_0). These conditions are explained in the following theorem:

Theorem 5.1.1 (Implicit Function Theorem). *[60] Suppose V is open in $\mathbb{R}^{n+m}$ and $F : V \to \mathbb{R}^n$ is continuously differentiable on V. Suppose further that $F(x_0, \mu_0) = 0$ for some $(x_0, \mu_0) \in V$, where $x_0 \in \mathbb{R}^n$ and $\mu_0 \in \mathbb{R}^m$. If*

$$\det\left(\frac{\partial(F_1, \ldots, F_n)}{\partial(x_1, \ldots, x_n)}(x_0, \mu_0)\right) \neq 0,$$

then there is an open set $W \subset \mathbb{R}^m$ containing μ_0 and a unique continuously differentiable function $x : W \to \mathbb{R}^n$ such that $x(\mu_0) = x_0$, and $F(x(\mu), \mu) = 0$ for all $\mu \in W$.

Let $(\bar{x}(t), \bar{\mu}(t))$ be any solution of (5.1), then $(\bar{x}(t), \bar{\mu}(t))$ is *stable* if solutions starting "close" to $(\bar{x}(t), \bar{\mu}(t))$ at a given time remain close to $(\bar{x}(t), \bar{\mu}(t))$ for all later times.

Definition 5.1.2 (Stability). [62] The solution $(\bar{x}(t), \bar{\mu}(t))$ is said to be *stable* if, given $\epsilon > 0$, there exists a $\delta = \delta(\epsilon) > 0$ such that, for any other solution, $(y(t), \nu(t))$, of (5.1) satisfying $\|(\bar{x}(t_0), \bar{\mu}(t_0)) - (y(t_0), \nu(t_0))\| < \delta$, then $\|(\bar{x}(t), \bar{\mu}(t)) - (y(t), \nu(t))\| < \epsilon$ for $t > t_0$, $t_0 \in \mathbb{R}$, where $(\bar{x}(t_0), \bar{\mu}(t_0)) = (x_0, \mu_0)$.

The stability of the stationary solution $(x(\mu), \mu)$ for a given μ is determined by the real parts of the eigenvalues of the Jacobian of the linearized system

$$\dot{x} = f_x(x_0, \mu_0)x = A(x_0, \mu_0)x. \tag{5.2}$$

The elements of the Jacobian matrix A depend on μ continuously, because f is continuously differentiable. Suppose that the eigenvalues of A are of the complex form $\lambda(\mu) = \alpha(\mu) + i\beta(\mu)$ and they are distinct, then a solution for (5.2) is of the form

$$e^{(\alpha+i\beta)t} = e^{\alpha t}(cos(\beta t) + isin(\beta t)).$$

In order to have stability, we assume that $\alpha = \mathrm{Re}(\lambda) < 0$ for all eigenvalues of A. Moreover, we assume that when μ is increasing, one or more eigenvalues cross the imaginary axis. In this case, the stability of the stationary solution can change. If for $\mu < \mu_0$ all eigenvalues of A were in the complex left half-plane, the stationary solution will have lost its stability for $\mu = \mu_0$, and for $\mu > \mu_0$ it will be unstable. This travelling of eigenvalues from one half-plane to another can be realized in one of two different ways. The first case appears when $\alpha(\mu_0) = \beta(\mu_0) = 0$. In this case the point (x_0, μ_0) is called a Takens-Bogdanov (TB) point, which is discussed in [39, 61]. The second case appears when $\alpha(\mu_0) = 0$ and $\beta(\mu_0) > 0$. In this case the point (x_0, μ_0) is called Hopf bifurcation point (see Figure 5.1.3). Now we give a brief discussion of Hopf bifurcation points.

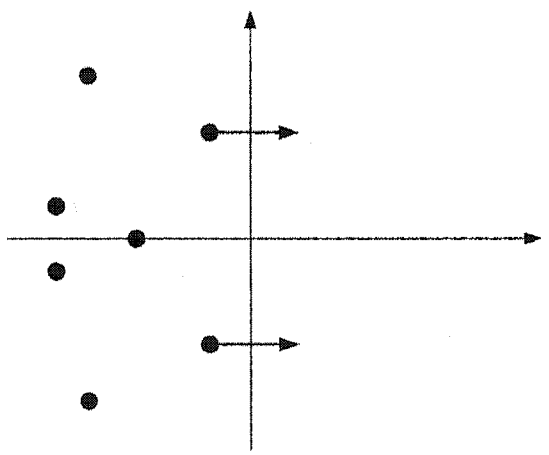


Figure 5.1: Hopf bifurcation point occurs.

Definition 5.1.3 (Hopf bifurcation point). [16, 23] Consider the dynamical system (5.1), and let $u = (x(\mu), \mu)$ be a solution curve of $f(x, \mu) = 0$. A point $u_0 = (x_0, \mu_0) = (x(\mu_0), \mu_0)$ is a simple Hopf bifurcation point if for small $|\mu - \mu_0|$ there is a complex conjugate pair of simple eigenvalues $\lambda(\mu) = \rho(\mu) \pm i\omega(\mu)$ of $f_x(x(\mu), \mu)$ crossing the imaginary axis for $\mu = \mu_0$, i.e., $\rho(\mu_0) = 0$, $\dot{\rho}(\mu_0) \neq 0$,

$\omega(\mu_0) \neq 0$, and all the remaining $n-2$ eigenvalues have negative real parts. The value $\nu_0 = (\omega(\mu_0))^2$ is called *the Hopf number.*

More information about Hopf bifurcation can be found in [30, 39, 47, 59, 62]. In the next section, we will discuss how to locate Hopf bifurcation points for the dynamical system (5.1).

5.2 Finding the Hopf Bifurcation Points

We briefly describe how to compute Hopf bifurcation points for the dynamical systems (5.1). Assume that $(x(\mu), \mu)$ is one parameter solution family of (5.1). In order to compute the Hopf bifurcation point, we introduce the one parameter matrix $A(\mu) = f_x(x(\mu), \mu)$ as in Section 5.1. From definition 5.1.3, Hopf bifurcation points occur when $A(\mu)$ has 2 pure imaginary eigenvalues and the remanning $n-2$ eigenvalues have negative real parts. Consequently, the numerical computation of Hopf bifurcation becomes the problem of determining μ so that $A(\mu)$ has a pair of pure imaginary eigenvalues [67].

Definition 5.2.1 (Hopf Matrix). We say that a matrix A is a Hopf matrix if A has two pure imaginary eigenvalues and the remanning $n-2$ eigenvalues have negative real parts.

The following example illustrates the main ideas of Hopf bifurcation points and will be used in the subsequent chapters.

Example 5.2.1. Let $f : \mathbb{R}^3 \times \mathbb{R}^2 \to \mathbb{R}^3$ be defined by :

$$f(x, \mu) = \begin{pmatrix} {\mu_1}^2 + \mu_2 - {\mu_2}^2 x_3 \\ x_1 + {\mu_1}^3 \mu_2 - \mu_1 x_3 \\ \mu_1 \mu_2 + x_2 - \mu_1 \mu_2 x_3 \end{pmatrix},$$

where $x \in \mathbb{R}^3$ and $\mu \in \mathbb{R}^2$. Then a stationary solution occurs at the point $(\bar{x}, \bar{\mu}) = (\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{\mu}_1, \bar{\mu}_2)$, where

$$\bar{x}_1 = \frac{\mu_1(\mu_1{}^2\mu_2{}^3 - \mu_1{}^2 - \mu_2)}{\mu_2{}^2}, \qquad \bar{x}_2 = \frac{\mu_1(-\mu_2{}^2 + \mu_1{}^2 + \mu_2)}{\mu_2},$$
$$\bar{x}_3 = \frac{\mu_1{}^2 + \mu_2}{\mu_2{}^2}, \qquad \bar{\mu}_i = \mu_i, \quad i = 1, 2.$$

and in this case,

$$A(\mu) := f_x(\bar{x}, \bar{\mu}) = \begin{pmatrix} 0 & 0 & -\mu_2{}^2 \\ 1 & 0 & -\mu_1 \\ 0 & 1 & -\mu_1\mu_2 \end{pmatrix}.$$

Suppose that $\mu_2 = 3$ and we want to find the value or values of μ_1 so that (x, μ) is a Hopf bifurcation point. We compute

$$A(\mu_1, 3) = \begin{pmatrix} 0 & 0 & -9 \\ 1 & 0 & -\mu_1 \\ 0 & 1 & -3\mu_1 \end{pmatrix},$$

and the characteristic polynomial of A is

$$p(\lambda) = \lambda^3 + 3\mu_1\lambda^2 + \mu_1\lambda + 9. \tag{5.3}$$

If the system $\dot{x} = f(x, \mu)$ has a Hopf bifurcation point, then A must have two pure imaginary eigenvalues $\pm i\omega$ and one real negative eigenvalue $s < 0$. Therefore,

$$\begin{aligned} p(\lambda) &= (\lambda + i\omega)(\lambda - i\omega)(\lambda - s) \\ &= (\lambda^2 + \omega^2)(\lambda - s) \\ &= \lambda^3 - s\lambda^2 + \omega^2\lambda - s\omega^2. \end{aligned} \tag{5.4}$$

Comparing (5.3) and (5.4) we find that the conditions for A to be a Hopf matrix are $3\mu_1 > 0$ and $3\mu_1^2 = 9$, that means $\mu_1 = \sqrt{3}$, and the Hopf number is $\nu = \sqrt{3}$.

In the previous example, we computed the Hopf bifurcation point by applying the definition 5.1.3. This can be done easily if A is a 3×3 matrix. However, if A is of higher dimension, definition 5.1.3 may not be easy to verify. For example, if A

is a 4×4 matrix with characteristic polynomial $p(\lambda) = \lambda^4 + c_3\lambda^3 + c_2\lambda^2 + c_1\lambda + c_0$, then A is a Hopf matrix if there exist $s, t, w^2 \in \mathbb{R}^+$ such that $p(\lambda) = \lambda^4 - (s + t)\lambda^3 + (w^2 + st)\lambda^2 - (s + t)w^2\lambda + w^2st$. Therefore, the conditions in this case are $c_1, c_3 < 0$, $c_0, c_2 > 0$, $s + t = -c_3$, $w^2 + st = c_2$, $(s + t)w^2 = -c_1$ and $w^2st = c_0$. This system is not easy to be dealt with and it will be more complicated if the size of A is greater than 4. Therefore, we need other numerical methods to find the Hopf bifurcation points. Several methods for the computation of Hopf bifurcation have been discussed in the literature, see for example [16, 49, 53, 54, 65, 67]. We will investigate two methods for the numerical of computation of Hopf bifurcation points in the next two chapters; polynomial resultants and bordered matrices.

CHAPTER 6

HOPF ALGORITHM USING POLYNOMIAL RESULTANTS

The polynomial resultants method uses the characteristic polynomial of the matrix $A(\mu)$ to compute Hopf bifurcation points. In Section 6.1, we define the Sylvester matrix and the resultant of two polynomials. Then we develop, in Section 6.2, the relationship between the characteristic polynomial of a matrix and the resultant. After that, Theorem 6.2.1 is used to detect Hopf bifurcation points. In Section 6.3, we describe the properties of this method and when it can be used. It turns out that this method is not practical for high dimensional matrices. However, this method can be used to find starting values for μ, the Hopf parameter, and ν, the Hopf number, in the next chapter. Finally, we discuss some test results in Section 6.4.

6.1 Sylvester Matrix

Consider the two polynomials

$$\begin{aligned} f(t) &= t^n + a_{n-1}t^{n-1} + \ldots + a_1 t + a_0, \qquad a_n \neq 0, \\ &= (t-\alpha_1)(t-\alpha_2)\ldots(t-\alpha_n), \\ g(t) &= t^m + b_{m-1}t^{m-1} + \ldots + b_1 t + b_0, \qquad b_m \neq 0, \\ &= (t-\beta_1)(t-\beta_2)\ldots(t-\beta_m). \end{aligned} \tag{6.1}$$

The Sylvester matrix of f and g, $S(f,g)$, is the $(m+n)\times(m+n)$ matrix defined by:

$$S(f,g) = \begin{bmatrix} 1 & a_{n-1} & \dots & a_1 & a_0 & 0 & 0 & 0 \\ 0 & 1 & a_{n-1} & \dots & a_1 & a_0 & 0 & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & \\ 0 & \dots & \dots & 0 & 1 & \dots & \dots & a_0 \\ 1 & b_{m-1} & \dots & b_1 & b_0 & 0 & 0 & 0 \\ 0 & 1 & b_{m-1} & \dots & b_1 & b_0 & 0 & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & \\ 0 & 0 & 0 & 0 & 1 & \dots & \dots & b_0 \end{bmatrix},$$

and the determinant of $S(f,g)$ is known as Sylvester's resultant which is denoted by $\mathcal{R}_s$ [56].

Definition 6.1.1 (Resultant). [10] Given two polynomials $f(t)$, $g(t) \in \mathbb{R}[t]$ with roots $\alpha_1, \dots, \alpha_m$ and $\beta_1, \dots, \beta_n$ respectively the resultant $R(f,g)$ of f and g is defined to be

$$R(f,g) = \prod_{i,j}(\alpha_i - \beta_j).$$

We will give a theorem to connect these two ideas in next section.

6.2 Detection and Computation of Hopf Points

We seek explicit criteria that specify whether an $n \times n$ matrix, A, has a pair of pure imaginary eigenvalues. We explain necessary conditions in terms of the corresponding characteristic polynomial for the matrix A. Let $p(\lambda) = c_0 + c_1\lambda + \dots + c_{n-1}\lambda^{n-1} + \lambda^n$. denote the characteristic polynomial of A. Then next proposition describes the conditions for p to have a pair of nonzero roots $\{\lambda, -\lambda\}$.

Proposition 6.2.1. *The characteristic polynomial p has the non-zero root pair $\{\lambda, -\lambda\}$ if and only if λ is a common root of the two polynomials $p(\lambda)+p(-\lambda)$ and $p(\lambda) - p(-\lambda)$.*

Proof. "$\Rightarrow$" Suppose that p has the roots λ and $-\lambda$ then $p(\lambda) \pm p(-\lambda) = 0 \pm 0 = 0$. "$\Leftarrow$" Adding $p(\lambda) + p(-\lambda)$ and $p(\lambda) - p(-\lambda)$ yields $p(\lambda) = 0$. While subtracting, yields $p(-\lambda) = 0$. □

We now make the substitution $z = \lambda^2$ to construct two new polynomials. If n is even , let

$$r_e(z) = c_0 + c_2 z + c_4 z^2 + \ldots + c_{n-2} z^{\frac{n-2}{2}} + z^{\frac{n}{2}}, \tag{6.2}$$
$$r_o(z) = c_1 + c_3 z + c_5 z^2 + \ldots + c_{n-1} z^{\frac{n-2}{2}},$$

while if n is odd, set

$$r_e(z) = c_0 + c_2 z + c_4 z^2 + \ldots + c_{n-3} z^{\frac{n-3}{2}} + c_{n-1} z^{\frac{n-1}{2}}, \tag{6.3}$$
$$r_o(z) = c_1 + c_3 z + c_5 z^2 + \ldots + c_{n-2} z^{\frac{n-3}{2}} + z^{\frac{n-1}{2}},$$

then p has a non-zero root pair $\{\lambda, -\lambda\}$ if there exists a z that satisfies :

$$\begin{pmatrix} r_e(z) \\ r_o(z) \end{pmatrix} = 0. \tag{6.4}$$

The polynomials $r_e(z)$ and $r_o(z)$ have a common root if and only if they share a common factor. Let the *Sylvester matrix* of the pair of equations (6.2) or (6.3) be the $(n-1) \times (n-1)$ matrix given by

$$\mathcal{S} = \begin{bmatrix} c_0 & c_2 & \cdots & c_{n-2} & 1 & 0 & 0 & \cdots & 0 \\ 0 & c_0 & c_2 & \cdots & c_{n-2} & 1 & 0 & \cdots & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & c_0 & c_2 & \cdots & c_{n-2} & 1 \\ c_1 & c_3 & \cdots & c_{n-1} & 0 & 0 & \cdots & \cdots & 0 \\ 0 & c_1 & c_3 & \cdots & c_{n-1} & 0 & \cdots & \cdots & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & c_1 & c_3 & \cdots & \cdots & c_{n-1} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \frac{n-2}{2} \text{ rows} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \frac{n}{2} \text{ rows} \end{matrix}$$

if n is even, while if n is odd, this matrix is defined to be

$$\mathcal{S} = \begin{bmatrix} c_0 & c_2 & \cdots & c_{n-3} & c_{n-1} & 0 & 0 & \cdots & 0 \\ 0 & c_0 & c_2 & \cdots & c_{n-3} & c_{n-1} & 0 & \cdots & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & c_0 & c_2 & \cdots & c_{n-3} & c_{n-1} \\ c_1 & c_3 & \cdots & c_{n-2} & 1 & 0 & \cdots & \cdots & 0 \\ 0 & c_1 & c_3 & \cdots & c_{n-2} & 1 & 0 & \cdots & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & & 0 \\ 0 & \cdots & \cdots & 0 & c_1 & c_3 & \cdots & c_{n-1} & 1 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \frac{n-1}{2} \text{ rows} \\ \left.\vphantom{\begin{matrix}1\\1\\1\\1\end{matrix}}\right\} \frac{n-1}{2} \text{ rows} \end{matrix}$$

The following proposition, which is proven in [10], describes the relationship between Sylvester's resultant and the resultant.

Proposition 6.2.2. *[10] Let f and g be defined as in (6.1) and let $S = S(f, g)$ be the Sylvester matrix. Then $R(f, g) = \det(S)$.*

Let $\mathcal{R}_s = \det(\mathcal{S})$ be the *Sylvester's resultant* of r_e and r_o .Then it follows immediately from proposition 6.2.2 that:

Proposition 6.2.3. *[29] The Sylvester matrix is singular (equivalently, $\mathcal{R}_s = 0$) if and only if the polynomials r_e and r_o share a common root.*

The next example describes how to compute the Sylvester matrix from the characteristic polynomial of a matrix.

Example 6.2.1. Let $p(\lambda)$ be the characteristic polynomial of A which is defined by:

$$\begin{aligned} p(\lambda) &= \lambda^5 - 2\lambda^4 + 10\lambda^3 - 19\lambda^2 + 9\lambda - 9 \\ &= (\lambda^2 + 9)(\lambda^3 - 2\lambda^2 + \lambda - 1). \end{aligned} \tag{6.5}$$

Then we define

$$r_e(z) = -9 - 19z - 2z^2,$$

$$r_o(z) = 9 + 10z + z^2.$$

and the Sylvester matrix for $r_e(z)$ and $r_o(z)$ is

$$\mathcal{S} = \begin{bmatrix} -9 & -19 & -2 & 0 \\ 0 & -9 & -19 & -2 \\ 9 & 10 & 1 & 0 \\ 0 & 9 & 10 & 1 \end{bmatrix}.$$

We then compute the determinant of S

$$\mathcal{R}_s = \det(S) = 0,$$

which means that $r_e(z)$ and $r_o(z)$ have a common root, say v. Therefore,

$$r_e(z) = -2(z - v)(z - \mu_1),$$
$$r_o(z) = (z - v)(z - \mu_2),$$

and $p(\lambda)$ can be written as :

$$\begin{aligned} p(\lambda) &= r_e(\lambda^2) + \lambda r_o(\lambda^2) \\ &= (\lambda^2 - v)(\lambda^3 - 2\lambda^2 - \mu_2\lambda + 2\mu_1). \end{aligned} \tag{6.6}$$

Comparing (6.5) and (6.6), we get $v = -9$, $\mu_1 = -\frac{1}{2}$ and $\mu_2 = -1$.

Suppose that $p(\lambda) = (\lambda^2 - \nu)q(\lambda)$, then in order for the matrix A to have a Hopf bifurcation point, we have to require $\nu < 0$. This condition can be verified using subresultants.

Definition 6.2.1 (Subresultants). [29] We define the subresultant matrices S_0 and S_1 to be the matrices obtained from $\mathcal{S}$ by deleting the rows 1 and $\frac{n}{2}$ and the columns 1 and $i + 2$, for $i = 0, 1$.

More information about resultants and subresultants can be found in [10, 11, 37, 41, 56]. Our work will be based on the following theorem which was proven in [29].

Theorem 6.2.1. *[29] Let $\mathcal{S}$ be the Sylvester matrix for the polynomials r_e and r_o in Equation (6.2) or (6.3). Then A has precisely one pair of pure imaginary eigenvalues if*

$$\mathcal{R}_s = 0 \text{ and } \det(\mathcal{S}_0) \cdot \det(\mathcal{S}_1) > 0.$$

If $\mathcal{R}_s \neq 0$ or $\det(\mathcal{S}_0) \cdot \det(\mathcal{S}_1) < 0$, then $p(\lambda)$ has no purely imaginary roots.

The following algorithm uses the Resultant Method to compute the Hopf bifurcation points of (5.1). Its details and Maple implementation are presented in the Appendix A.

n	$\mathcal{R}_s$	$\det(S_0)\cdot\det(S_1)$
2	c_1	c_0
3	$c_0 - c_1c_2$	c_1
4	$c_0c_3^2 - c_1c_2c_3 + c_1^2$	c_1c_3
5	$(c_2 - c_3c_4)(c_1c_2 - c_0c_3) + c_1c_4(c_1c_4 - 2c_0) + c_0^2$	$(c_2 - c_3c_4)\cdot(c_0 - c_1c_4)$

Table 6.1: Conditions for theorem 6.2.1 where $n = 2, 3, 4, 5$.

Algorithm 6.2.1 (Resultant Algorithm).

INPUT: A matrix A which depends on μ.

OUTPUT: A set $H = \{\mu : \mathcal{R}_s(\mu) = 0 \text{ and } \det(S_0(\mu))\cdot\det(S_1(\mu)) > 0\}$.

1. Compute the characteristic polynomial of A.
2. Find the Sylvester matrix of A.
3. Compute the subresultants S_0 and S_1.
4. Evaluate the determinant of S and solve $\mathcal{R}_s = 0$.
5. $H = \phi$.
6. For each root μ of $\mathcal{R}_s = 0$

   if $\det(S_0(\mu))\cdot\det(S_1(\mu)) > 0$

   $H = H \cup \{\mu\}$.

6.3 Comments about the Resultants Method

The Resultants method works well for small dimension matrices; however, it cannot be applied for high dimensional matrices [29] because the computation of the characteristic polynomial coefficients is numerically unstable [49]. On the other hand, this method can give an explicit description for the Hopf curve, i.e., a curve such that each point in this curve is a Hopf bifurcation point, (see example 6.4.1). Moreover, it can find all Hopf bifurcation points.

6.4 Examples

We illustrate how this method works by some examples where the Hopf data reported in the discussion which follows was computed using the Maple symbolic algebra package.

Example 6.4.1. We reconsider example 5.2.1, where

$$p(\lambda) = \lambda^3 + \mu_1\mu_2\lambda^2 + \mu_1\lambda + \mu_2^2.$$

In this case, we have $c_0 = \mu_2^2$, $c_1 = \mu_1$ and $c_2 = \mu_1\mu_2$. We now use the table (6.1) which gives the conditions

1. $\mu_2^2 = \mu_1^2\mu_2$.

2. $\mu_1 > 0$.

Therefore, the Hopf bifurcation curve of A is defined by $\mu_2 = \mu_1^2$, where $\mu_1 > 0$, and the Hopf number is $\nu = \mu_1$. So, we have

$$p(\lambda) = (\lambda^2 + \mu_1)(\lambda + \mu_1^3).$$

Even though the matrix A in Example 6.4.1 has an analytic description of the Hopf bifurcation curve, that is not always the case as we will see in the next examples. The following example shows a case where (5.2) does not have any Hopf bifurcation point.

Example 6.4.2. Let

$$A(\mu) := \begin{bmatrix} 1 & -2 & \mu_2 \\ 1 & 0 & \mu_1 \\ \mu_2 & \mu_1 & 1 \end{bmatrix}, \tag{6.7}$$

then the characteristic polynomial of A is

$$p(\lambda) = \lambda^3 - 2\lambda^2 - (1 + \mu_1^2 + \mu_2^2)\lambda + (2 - 2\mu_1\mu_2 + \mu_1^2).$$

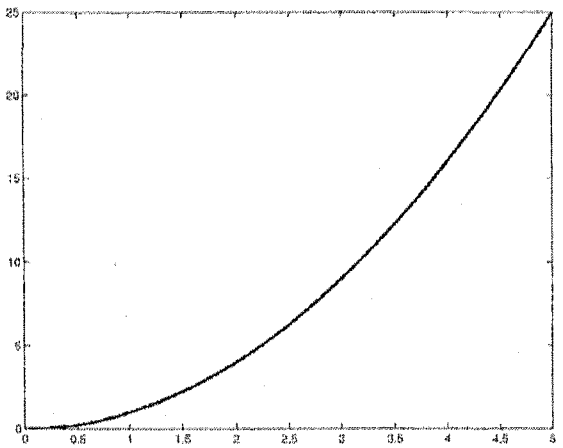


Figure 6.1: Hopf bifurcation curve for Example 6.4.1

We therefore have $c_0 = 2 - 2\mu_1\mu_2 + \mu_1^2$, $c_1 = -(1 + \mu_1^2 + \mu_2^2)$, $c_2 = -2$. In order to find Hopf bifurcation points, we use Table 6.1 to get the conditions

1. $2 - 2\mu_1\mu_2 + \mu_1^2 = 2(1 + \mu_1^2 + \mu_2^2)$.

2. $-(1 + \mu_1^2 + \mu_2^2) > 0$.

However, c_1 is strictly less than zero. Therefore, A does not have Hopf bifurcation points.

Next example was given in [8]. The system (5.2) has a Hopf bifurcation point, but not a Hopf bifurcation curve.

Example 6.4.3. Let

$$A(\mu) := \begin{bmatrix} -1 & 0 & -\frac{1}{2}\mu \\ \frac{1}{2}\mu & -1 & 0 \\ 0 & \frac{1}{2}\mu & -1 \end{bmatrix},$$

then the characteristic polynomial of A is

$$p(\lambda) = \lambda^3 + 3\lambda^2 + 3\lambda + (1 + \frac{1}{8}\mu^3),$$

and we have $c_0 = 1 + \frac{1}{8}\mu^3$, $c_1 = 3$, $c_2 = 3$. In order to find a Hopf bifurcation point, we use table 6.1 to get the conditions

1. $1 + \frac{1}{8}\mu^3 = 9 \Rightarrow \mu = 4$.

2. $3 > 0$.

Therefore, A has only one Hopf bifurcation point; namely at $\mu = 4$.

The following example was constructed in such a way that the system (5.2) has a Hopf bifurcation point at $(\mu_1, \mu_2, \mu_3) = (1, 1, 1)$

Example 6.4.4. Let

$$A(\mu) := \begin{pmatrix} 0 & 0 & 0 & -2\mu_1 \sin(\frac{\pi}{2}\mu_1\mu_2) - 6\mu_1\mu_2 \\ 1 & 0 & 0 & 2(\mu_1 + \mu_2)^2 \\ 0 & 1 & 0 & -2e^{(\mu_1\mu_2 - 1)}\mu_2 - 4\mu_1 \\ 0 & 0 & 1 & 2\sin(\frac{\pi}{2}\mu_2\mu_3) \end{pmatrix}. \tag{6.8}$$

Then

$$\begin{aligned} c_0 &= 2\mu_1 \sin(\frac{\pi}{2}\mu_1\mu_2) + 6\mu_1\mu_2, \\ c_2 &= -2(\mu_1 + \mu_2)^2, \\ c_3 &= 2e^{(\mu_1\mu_2 - 1)}\mu_2 + 4\mu_1, \\ c_4 &= -2\sin(\frac{\pi}{2}\mu_2\mu_3). \end{aligned}$$

We want to compute the Hopf bifurcation surface in a neighborhood of the point $(1, 1, 1)$. We use Table 6.1 to get the conditions

1. $1 - c_0c_3^2 - c_1c_2c_3 + c_1^2 = 0$.

2. $2 - c_1c_3 > 0$.

It is clear that this surface cannot be computed analytically because of the terms $\sin(\frac{\pi}{2}\mu_1\mu_2)$ and $\sin(\frac{\pi}{2}\mu_2\mu_3)$. Therefore, we use Maple to compute the surface numerically. Now we consider μ_3 as a function of μ_1 and μ_2 and vary μ_1 and μ_2. Then we solve condition (1) for μ_3 and check if condition (2) is satisfied. Some values of μ_3 are computed and listed in Table 6.2.

μ_1	μ_2	$\mu_3 := \mu_3(\mu_1, \mu_2)$
.9999999	.9999999	1.000000200
.9999999	1	1.000000100
1	1	1
1	1.0000001	1.000426957
1.0000001	1.0000001	1.000667492

Table 6.2: The values of μ_3 at a neighborhood of $(1, 1)$ for example 6.4.1

In the following chapter we describe another algorithm to find Hopf bifurcation points which is more efficient and can be applied for higher dimensional matrices. However, this algorithm requires starting values for μ, the Hopf parameter, and ν, the Hopf number. These values can be obtained from the resultants method if we have a discretization problem. We first choose a small value for n, the dimension of A, and then use the resultant method to locate the starting values μ_0 and ν_0. Then we increase the dimension n and and use the method in the next chapter with starting values μ_0 and ν_0.

CHAPTER 7

HOPF ALGORITHM USING BORDERED MATRICES

The bordered matrices method is widely used nowadays to compute Hopf bifurcation points. In this method, *bordered matrices* are used to classify the set of matrices with rank deficiency two, at which Hopf bifurcation point occurs. We first define the Hopf manifold in Section 7.1. Then we describe the relationship between bordered matrices and the Hopf manifold in Section 7.2. In Section 7.3, we use Theorem 7.3.1 to write an explicit algorithm to detect Hopf bifurcation points. We conclude this chapter with numerical examples to compute Hopf bifurcation points.

7.1 Hopf Manifold

We say that A is a Hopf matrix with Hopf number $\nu = \omega^2 > 0$, if A has geometrically simple pure imaginary eigenvalues $\pm i\omega$. Then we define the Hopf manifold to be:

Definition 7.1.1 (Hopf Manifold). [61] The Hopf manifold, $\mathcal{M}_{H,\nu}$, is defined to be:

$$\mathcal{M}_{H,\nu} := \{(A, \nu) \in \mathbb{R}^{n\times n} \times \mathbb{R} : A \text{ is a Hopf matrix with Hopf number } \nu\}.$$

If we denote $\mathcal{N}^{\nu}(A)$ to be the kernel of $A^2 + \nu I$ then $\mathcal{N}^{\nu}(A)$ is just the real eigenspace associated with the eigenvalues $\pm i\omega \in \sigma(A)$ [61]. It is possible to give

a characterization of $\mathcal{M}_{H,\nu}$ in terms of two scalar equations

$$\alpha(A,\nu) = 0, \qquad \beta(A,\nu) = 0, \tag{7.1}$$

which can be used for the computation of *curves* of Hopf matrices since $\mathcal{M}_{H,\nu}$ is a codimension-2 manifold [61].

7.2 Bordered Matrices

We consider the following linear bordered system:

$$\begin{pmatrix} B & R \\ L^T & 0 \end{pmatrix} \begin{pmatrix} U \\ T \end{pmatrix} = \begin{pmatrix} 0_{n,m} \\ I_m \end{pmatrix} \tag{7.2}$$

with $B \in \mathbb{R}^{n\times n}$, $R, L, U \in \mathbb{R}^{n\times m}$, and $T \in \mathbb{R}^{m\times m}$ and we denote the bordered matrix in (7.2) by $\mathcal{B}(B,R,L)$. We assume R and L have maximal rank m. Our goal is to classify matrices B with rank deficiency m. This can be done by taking $T = 0_m$. In this case, we have m^2 equations. The following proposition reduces the size of T for the special case $B = A^2 + \nu I$ and $m = 2$.

Proposition 7.2.1. *If $B := A^2 + \nu I \in \mathbb{R}^{n\times n}$ is singular and $\nu = \omega^2 > 0$, then B has rank deficiency at least two.*

Proof. Since B is singular, $\det(B)$ equals 0, and we have

$$\begin{aligned} \det(A^2 + \nu I) = 0 &\Rightarrow \det((A + i\omega I)(A - i\omega I)) = 0 \\ &\Rightarrow \det(A + i\omega I)\det(A - i\omega I)) = 0, \end{aligned}$$

so either $\det(A + i\omega I) = 0$ or $\det(A - i\omega I) = 0$. If $\det(A + i\omega I) = 0$, then $-i\omega$ is an eigenvalue of A, but since A is real $i\omega$ is also an eigenvalue of A. The same argument can be made if $\det(A - i\omega I) = 0$. So, there exist two linearly independent vectors x and $y \in \mathbb{C}^n$ such that $(A + i\omega I)x = 0$ and $(A - i\omega I)y = 0$. This implies that $Bx = 0$ and $By = 0$. Consequently, B has rank deficiency at least two. □

In the bordered matrix system (7.2), we let $m = 2$ and $B = A^2 + \nu I$ to find Hopf bifurcation points. It seems that we need to have $T = 0_2 \in \mathbb{R}^{2\times 2}$ in order for B to have rank deficiency two. However, it is enough to take $T = [0, 0]^T$ since $\nu > 0$ and B is singular, which implies that B has at least rank deficiency two, by Proposition 7.2.1. Moreover, we take $[1, 0]^T$ or $[0, 1]^T$ instead of I_2

7.3 Conditions for Hopf Bifurcation

For the bordered matrices in $\mathcal{B}(B, R, L)$, a special choice for R and L, which depends on A, was done by Werner [61]

$$\mathcal{B}(A, \nu) := \begin{pmatrix} A^2 + \nu I & Ar & r \\ l^T A & 0 & 0 \\ l^T & 0 & 0 \end{pmatrix}, \tag{7.3}$$

where r and $l \in \mathbb{R}^n$. Then we consider the system:

$$\begin{pmatrix} A^2 + \nu I & Ar & r \\ l^T A & 0 & 0 \\ l^T & 0 & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0_n \\ 1 \\ 0 \end{pmatrix}, \tag{7.4}$$

where $u_1 \in \mathbb{R}^n$, $\alpha, \beta \in \mathbb{R}$. Observe that the normalizing conditions

$$l^T u_1 = 0, \qquad l^T A u_1 = 1,$$

from our bordered system force u_1 not to be an eigenvector of A. Because if u_1 is an eigenvector of A, then we have

$$Au_1 = \lambda u_1 \Rightarrow l^T A u_1 = \lambda l^T u_1$$

$$\Rightarrow 1 = 0 \quad \text{(contradiction)}.$$

We now define acute pairs of subspaces, which will be used in the subsequent discussion.

Definition 7.3.1. [61] The pair (U, V) of two m-dimensional subspaces U and V of $\mathbb{R}^n$ are acute if:

$$U \cap V^{\perp} = \{0\}.$$

In other words, $uv \neq 0$, $\forall\ u \in U$, $v \in V$.

The following lemmas were given without proof in [61]. These lemmas are used to prove Theorem 7.3.1.

Lemma 7.3.1. *(H. B. Keller)[27]. If B is regular, then the regularity of $\mathcal{B}(B,R,L)$ is equivalent to the regularity of the Schur complement $L^TB^{-1}R \in \mathbb{R}_{m\times m}$. Moreover,*

$$T = T(B) = -(L^TB^{-1}R)^{-1}.$$

If rank B is less than $n-m$, then $\mathcal{B}(B,R,L)$ is singular and if rank $B = n-m$, then $\mathcal{B}(B,R,L)$ is regular if and only if the pairs $(Im(L),\mathcal{N}(B))$ and $(Im(R),\mathcal{N}(B^T))$ of m-dimensional spaces are acute.

Lemma 7.3.2. *If A is a Hopf matrix with Hopf number $\nu > 0$, then* $\dim\mathcal{N}^\nu(A) = 2$ *(or equivalently the rank deficiency of $A^2+\nu I$ is two). The converse is also true.*

Proof. " $\Rightarrow$ " Suppose that A is a Hopf matrix with Hopf number $\nu = \omega^2 > 0$. Then A has simple imaginary eigenvalues $\pm i\omega$, i.e. there exist two linearly independent vectors $x, y \in \mathbb{R}^n$ such that $Ax = i\omega x$ and $Ay = -i\omega y$. Multiplying by A, we get $A^2x = -\omega^2 x$ and $A^2y = -\omega^2 y$. That implies $x, y \in \mathcal{N}^\nu(A)$. Therefore $\dim\mathcal{N}^\nu(A) \geq 2$. But $\pm i\omega$ are simple eigenvalues of A, so $\dim\mathcal{N}^\nu(A) = 2$.
" $\Leftarrow$ " Suppose that $\dim\mathcal{N}^\nu(A) = 2$, and let $\mathcal{N}^\nu(A) = \text{span}\{x, y\}$. Then $\det(A^2 + \nu I) = 0$. Which is equivalent to

$$\det((A + i\omega I)(A + i\omega I)) = 0.$$

By the same argument we used to prove Proposition 7.2.1, we conclude that $\pm i\omega$ are eigenvalues of A. And since $\dim\mathcal{N}^\nu(A) = 2$, $\pm i\omega$ are simple eigenvalues. Therefore A is a Hopf matrix. □

Lemma 7.3.3. *A is a Hopf matrix with Hopf number $\nu > 0$ iff there exists a nonzero u such that u and Au are linearly independent and span $\mathcal{N}^\nu(A)$.*

Proof. " $\Rightarrow$ " Suppose that A is a Hopf matrix with Hopf number $\nu = \omega^2 > 0$. Then from Lemma 7.3.2, $\dim \mathcal{N}^\nu(A) = 2$. Let $u \in \mathcal{N}^\nu(A)$, then $(A^2 + \nu I)u = 0$. By the way of contradiction, we suppose that u and Au are linearly dependent, then there exists a $\lambda \in \mathbb{R}$ such that $Au = \lambda u$. Now

$$(A^2 + \nu I)u = \lambda^2 u + \nu u,$$

this implies that $\nu = -\lambda^2 < 0$, which is a contradiction.
" $\Leftarrow$ " If $\mathcal{N}^\nu(A) = \text{span}\{u, Au\}$, where u and Au are linearly independent, then it follows immediately from Lemma 7.3.2 that A is a Hopf matrix □

Theorem 7.3.1. *[61] Let $\mathcal{B}(A, \nu)$ be regular. Then*

$$(A, \nu) \in \mathcal{M}_{H,\nu} \Leftrightarrow \alpha(A, \nu) = 0, \quad \beta(A, \nu) = 0. \tag{7.5}$$

Proof. " $\Rightarrow$ " : Because of Lemma 7.3.2 we have $\dim \mathcal{N}^\nu(A) = 2$ and by Lemma 7.3.1, ($\mathcal{B}$ is regular), there exists no nontrivial $\tilde{u} \in \mathcal{N}^\nu(A)$ satisfying

$$l^T \tilde{u} = 0, \qquad l^T A\tilde{u} = 0.$$

Hence there is a unique $u_1 \in \mathcal{N}^\nu(A)$ with $l^T A u_1 = 1$, $l^T u_1 = 0$ and we have $\alpha(A, \nu) = 0$, $\beta(A, \nu) = 0$.
" $\Leftarrow$ ": u_1 and Au_1 are linearly independent (this follows from $l_1^T u_1 = 0$ and $l_1^T A u_1 = 1$). Both are in $\mathcal{N}^\nu(A)$. By Lemma 7.3.1 and the regularity of $\mathcal{B}_H$, $\dim \mathcal{N}^\nu(A) = 2$, and $\mathcal{N}^\nu(A)$ is spanned by u_1 and Au_1 . Now apply Lemma 7.3.2. □

If we interchange 1 and 0, we obtain an analogous description of $\mathcal{M}_{H,\nu}$ by $\gamma(A, \nu) = 0$, $\delta(A, \nu) = 0$ with γ, δ replacing α, β. The following theorem gives the conditions for $\mathcal{B}(A, \nu)$ to be regular and was proved in [61].

Theorem 7.3.2. *Let A be a Hopf matrix with Hopf number ν (or let $\dim \mathcal{N}^\nu(A) = 2$) Then $\mathcal{B}(A, \nu)$ is regular if and only if*

$$l^T \mathcal{N}^\nu(A) \neq \{0\} \text{ and } r^T \mathcal{N}^\nu(A^T) \neq \{0\}. \tag{7.6}$$

The two scalar equations

$$\alpha(A,\nu) = 0, \qquad \beta(A,\nu) = 0$$

are equivalent to $(A,\nu) \in \mathcal{M}_{H,\nu}$, *provided that (7.6) are fulfilled.*

7.4 Computation of Hopf Points.

To detect Hopf points we use the test function $\tau(A) := \alpha(A,\nu(A))$ during a path-following algorithm applied to A. We first solve $\beta(A,\nu) = 0$ for $\nu = \nu(A)$. To perform this step, we fix a *reference number* ν_0 and try to solve the scalar equation $\beta(A,\nu) = 0$ for given A by a secant-like method with starting value $\nu = \nu_0$. Then we set $\tau(A) := \alpha(A,\nu(A))$ with the approximate solution $\nu = \nu(A)$ of $\beta(A,\nu) = 0$. Hopf bifurcation will then be detected by a sign change of $\tau(A)$.

The following theorem, on which we base our numerical approach, is due to Werner [61]. Let us introduce the notation

$$\bar{A} := A(\bar{u}) \qquad \bar{\nu} := \bar{\omega}^2$$

Theorem 7.4.1. *[23] Let* $\bar{u} = (x(\bar{\mu}),\bar{\mu})$ *be a simple Hopf bifurcation point. Let* $l, r \in \mathbb{R}^N$ *be such that span* $\{\bar{A}^T l, l\}$ *is acute to kernel(* $\bar{A}^2 + \bar{\nu}I$*), and span* $\{\bar{A}r, r\}$ *is acute to kernel(* $\bar{A}^2 + \bar{\nu}I)^T$*. Then the following linear system is nonsingular for* $u = \bar{u}$ *and* $\nu = \bar{\nu}$

$$\begin{pmatrix} A^2(u) + \nu I & A(u)r & r \\ l^T A(u) & 0 & 0 \\ l^T & 0 & 0 \end{pmatrix} \begin{pmatrix} u_1(u,\nu) \\ \alpha(u,\nu) \\ \beta(u,\nu) \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}. \tag{7.7}$$

Note that for $\nu = \bar{\nu}$ *and* $u = \bar{u}$ *we have* $\alpha(\bar{u},\bar{\nu}) = \beta(\bar{u},\bar{\nu}) = 0$, *and* $u_1(\bar{u},\bar{\nu})$ *is a particular vector in the two-dimensional kernel of* $\bar{A}^2 + \bar{\nu}I$. *Furthermore,* $\beta(u,\nu) = 0$ *implicitly defines* $\nu(u)$, *i.e.,* $\partial_2\beta(\bar{u},\bar{\nu}) \neq 0$, *and* $\tau(u) := \alpha(u,\nu(u))$ *changes sign at* $u = \bar{u}$. *Hence,* $\tau(u)$ *can be regarded as a test function for Hopf bifurcation.*

The following algorithm uses the bordered method to compute the Hopf bifurcation points of (5.1). Its details and Matlab implementation can be found in the Appendix A.

Algorithm 7.4.1 (Bordered Matrices Algorithm).

INPUT: A matrix A that depends on μ.

OUTPUT: (μ, ν) such that $\alpha(A, \nu) = 0$, and $\beta(A, \nu) = 0$.

1. Chose r and l to be in generic position (i.e., randomly). By the theorem of Sard, the matrix in (7.7) will be regular for almost all l and r in the Lebesgue sense.

2. Get an approximation for ν_0, and μ_0 (this can be done using the method described in Chapter 6).

3. Define $M := \mathcal{B}_H(A, \nu)$ as in (7.3).

4. Define β and α to be the solution of (7.4).

for fixed μ

using secant like method, find ν such that $\beta(A(\mu), \nu) = 0$,

so we defined $\nu = \nu(\mu)$.

for fixed ν

using a secant like method, find μ such that $\alpha(A(\mu), \nu(\mu)) = 0$.

7.5 Examples

We will choose two examples from the previous chapter so that we can compare our results. We use a Matlab package to compute the Hopf bifurcation points.

μ_1	$\bar{\nu}$	$\bar{\mu_2}$	ν	$\mu_2 := \mu_2(\mu_1)$
1	.5	.3	0.99999999999995	0.99999999999992
1.5	.5	.3	1.49999999999999	2.24999999999998
2	.5	.3	1.99999999998700	3.99999999992167
2.5	.5	.3	2.49999999999784	6.24999999998033

Table 7.1: Some values for μ_2 are computed for Example 7.5.1

Example 7.5.1. We reconsider example 5.2.1, where

$$A(\mu) := \begin{pmatrix} 0 & 0 & -{\mu_2}^2 \\ 1 & 0 & -\mu_1 \\ 0 & 1 & -\mu_1\mu_2 \end{pmatrix}.$$

We vary μ_1 and compute $\mu_2 := \mu_2(\mu_1)$ using Algorithm 7.4.1. As starting values, $\bar{\nu}$ was chosen to be .5, where $\bar{\mu_2} = 0.3$. It turns out that this choice works for $\mu_1 = 1, 2, ..., 12$. After that another choice must be made. The numerical results are listed in Table 7.1.

Example 7.5.2. We reconsider example 6.4.4 where,

$$A(\mu) := \begin{pmatrix} 0 & 0 & 0 & -2\mu_1 \sin(\frac{\pi}{2}\mu_1\mu_2) - 6\mu_1\mu_2 \\ 1 & 0 & 0 & 2(\mu_1+\mu_2)^2 \\ 0 & 1 & 0 & -2e^{(\mu_1\mu_2-1)}\mu_2 - 4\mu_1 \\ 0 & 0 & 1 & 2\sin(\frac{\pi}{2}\mu_2\mu_3) \end{pmatrix}.$$

We vary μ_1, μ_2 and compute $\mu_3 := \mu_3(\mu_1, \mu_2)$ in a neighborhood of (1,1) using algorithm 7.4.1. As starting values, $\bar{\nu}$ was chosen to be 3, where $\bar{\mu_3} = 1.5$. Since we are in a small neighborhood of (1,1), this choice can be fixed. The numerical results for this example are listed in Table 7.2.

μ_1	μ_2	$\bar{\nu}$	$\bar{\mu_3}$	ν	$\mu_3 := \mu_3(\mu_1, \mu_2)$
1	1	3	1.5	4.00000000000000	1.00000000497444
1	1.0000001	3	1.5	4.00000050000005	1.00014225251169
1.0000001	1	3	1.5	4.00000079999995	1.00028470497100
1.0000001	1.0000001	3	1.5	4.00000130000003	1.00031820980166

Table 7.2: Some values for μ_3 are computed in a neighborhood of (1,1) for Example 7.5.2

One can immediately see that both algorithms from this chapter and the previous one gave the same results. These algorithms will be used again in matrices with higher dimension resulting from a stability analysis of the electroconvection of nematic liquid crystals (see chapter 8).

CHAPTER 8

APPLICATION TO NEMATIC LIQUID CRYSTALS

Electroconvection in Nematic Liquid Crystals (NLC) is a paradigm for pattern formation in anisotropic systems, exhibiting a rich variety of dynamical structures. When an electric potential is applied across a thin NLC layer confined between two electrode plates, an electrohydrodynamic instability can occur above a critical field strength. The neutral stability of the basic state occurs on a neutral stability surface in (p, q, R)-space, where R is the bifurcation parameter, and p, q are the wave numbers in the x and y direction, respectively. The parameter values of physical interest are (p_c, q_c, R_c) - the coordinates of the global R-minimum on the Oscillatory Neutral Stability Surface (ONSS), for which the first physical instability, when R increases, is a Hopf Bifurcation. The mathematical model for the nematic electroconvection is described in Section 8.1. In Section 8.2 we apply the bordered matrices algorithm described previously to numerically compute the ONNS using Hopf Algorithm 7.4.1 described in Chapter 7 and then compute the global minimum using the NMA, Algorithm 4.1.1, and the CEA, 2.2.1.

8.1 Basic Equations and Linear Stability Analysis

The bifurcation analysis on the weak electrolyte model (WEM) has been discussed in [13]. We consider a thin layer of nematic liquid crystals sandwiched between two horizontal electrode plates, subjected to a vertical electric field. The weak electrolyte model (WEM) describes the electroconvection process that takes

place in the nematic and is described by the Navier-Stokes Equations. In nondimensionalized units, the WEM consists the following set of equations.

$$(\partial_t + \mathbf{v}\cdot\nabla)\mathbf{n} = \omega \times \mathbf{n} + \mathbf{d}(\lambda \mathbf{A}\mathbf{n} - \mathbf{h}), \tag{8.1}$$

$$P_1(\partial_t + \mathbf{v}\cdot\nabla)\mathbf{v} = -\nabla p - \nabla\cdot(\mathbf{T} + \Pi) + \pi^2\rho\mathbf{E}, \tag{8.2}$$

$$\nabla\cdot\mathbf{v} = 0, \tag{8.3}$$

$$P_2(\partial_t + \mathbf{v}\cdot\nabla)\rho = -\nabla\cdot(\mu\mathbf{E}\sigma), \tag{8.4}$$

$$(\partial_t + \mathbf{v}\cdot\nabla)\sigma = -\alpha^2\pi^2\nabla\cdot(\mu\mathbf{E}\rho) - \frac{r}{2}(\sigma^2 - 1 - P_1\alpha\pi^2\rho^2), \tag{8.5}$$

together with Poisson's law,

$$\rho = \nabla\cdot(\epsilon E),$$

where the dielectric tensor ϵ and the conductivity tensor μ are given by $\epsilon_{ij} = \delta_{ij} + \epsilon_a n_i n_j$ and $\mu_{ij} = \delta_{ij} + \sigma_a n_i n_j$, respectively, $\mathbf{n}$ is the director field, $\mathbf{v}$ is the fluid velocity, p is the pressure, $\mathbf{E} = (\sqrt{2R}/\pi)\mathbf{e}_3 - \nabla\phi$ is the electric field with ϕ the potential, $\omega = \frac{1}{2}(\nabla\times\mathbf{v})$ is the vorticity, ρ is the charge density, σ is the local conductivity and ε_a and σ_a are the dielectric, respectively the conductivity, coefficients. Furthermore, P_1 is the ratio of the viscous relaxation time to the director relaxation time, P_2 is the ratio of the charge relaxation time to the director relaxation time, and $\mathbf{h}$ is the molecular field which is given by

$$\mathbf{h} = 2(\frac{\partial f}{\partial \mathbf{n}} - \nabla\cdot\frac{\partial f}{\partial\nabla\mathbf{n}}) - \varepsilon_a\pi^2(\mathbf{n}\cdot\mathbf{E})\mathbf{E},$$

and is derived from the elastic energy density due to splay, twist and bend deformations,

$$2f = (\nabla\cdot\mathbf{n})^2 + K_2[\mathbf{n}\times(\nabla\times\mathbf{n})]^2 + K_3[\mathbf{n}\cdot(\nabla\times\mathbf{n})]^2.$$

The parameter ε_a is the dielectric coefficient, and K_2 and K_3 are the ratios of the twist and bend elastic coefficients to the splay elastic coefficient. The tensors $\mathbf{d}$,

$\mathbf{A}$, and $\mathbf{T}$ are the projection tensor $d_{ij} = \delta_{ij} - n_i n_j$ that guarantees $|\mathbf{n}| = 1$, the shear flow tensor $A_{ij} = \frac{1}{2}(v_{i,j} + v_{j,i})$, and the viscous stress tensor

$$-T_{ij} = \alpha_1 n_i n_j n_k n_l A_{kl} + \alpha_2 n_j m_i + \alpha_3 n_i m_j + \alpha_4 A_{ij} + \alpha_5 n_j n_k A_{ki} + \alpha_6 n_i n_k A_{kj},$$

where $\mathbf{m} = \mathbf{d}(\lambda \mathbf{A}\mathbf{n} - \mathbf{h})$. The tensor Π is the nonlinear Ericksen stress tensor with components $\Pi_{ij} = \frac{\partial f}{\partial n_{k,j}} n_{k,i}$. In our dimensionless units, the Leslie coefficients $\alpha_1, \ldots, \alpha_6$ and the Onsager coefficient λ satisfy the Onsager and scaling relations $\alpha_1 + \alpha_3 = \alpha_6 - \alpha_5$, $\alpha_3 - \alpha_2 = 1$ and $\lambda = \alpha_5 + \alpha_6$. These parameters are usually expressed in terms of four independent Miesowicz coefficients $\eta_0, \eta_1, \eta_2, \eta_3$,

$$\lambda = \eta_1 - \eta_2, \alpha_1 = \eta_0 - 2\eta_1 - 2\eta_2 + 2\eta_3 + 1, \alpha_2 = -(1+\lambda)/2, \alpha_3 = (1-\lambda)/2,$$

$$\alpha_4 = 2\eta_3, \alpha_5 = 2\eta_1 - 2\eta_3 - (1+\lambda)/2, \alpha_6 = 2\eta_2 - 2\eta_3 - (1-\lambda)/2.$$

We assume an infinitely extended NLC layer in both horizontal directions (x, y) and use vertical boundary conditionsl

$$\frac{\partial \sigma}{\partial z}, n_2, n_3, \phi, \mathbf{v} = 0 \text{ at } z = \pm\frac{\pi}{2}.$$

The WEM equations (8.1)–(8.5) depend on the basic parameter R, proportional to the strength of the electric field, four time scale ratios P_1, P_2, α and r, and the following scaled material parameters: ε_a, σ_a, K_2, K_3, and the Miesowicz coefficients $\eta_0, \eta_1, \eta_2, \eta_3$.

In [13], P_1 and P_2 are set to zero in the system of dynamical equations (8.1)–(8.5), because P_1, P_2 are very small compared to the other parameters. In this limit $\mathbf{v}$ and ϕ are slaved by σ and $\mathbf{n}$ through the equations (8.2) and (8.4) [13]. In addition to the main parameter R, we are left with ten independent parameters $\eta_0, \eta_1, \eta_2, \eta_3, \varepsilon_a, \sigma_a, \alpha, K_2, K_3$.

The basic (non-convecting) state of (8.1)–(8.5) is given by $\sigma = 1$, $\mathbf{n} = (1, 0, 0)$, $\mathbf{v} = \mathbf{0}$, $p =$ constant and $\phi = 0$. The stability of this state is governed by linearized

equations for perturbational fields $\delta\sigma$, δn_2, δn_3, $\delta\phi$, δv_j, δp. Owing to the translation invariance w.r.t. (x, y), the perturbational fields are represented by horizontal Fourier modes,

$$(\delta\sigma, \delta n_2, \delta n_3, \delta\phi, \delta v_j, \delta p) = e^{i(px+qy)}(\Sigma, N_2, N_3, \Phi, V_j, P),$$

where Σ, N_2 etc. depend on (t, z, p, q) and the parameters. The V_j and P are represented by poloidal and toroidal stream functions F and G leaving us with a system of linear equations for $(\Sigma, N_2, N_3, \Phi, V_j, F, G)$. Symbolically, these equations can be written as

$$\partial_t \mathbf{U} = \mathbf{L_U}(\mathbf{U}, \Phi, p, q, R), \tag{8.6}$$

$$L_\Phi(\Phi, \mathbf{U}, p, q, R) = 0, \tag{8.7}$$

$$\mathbf{L_F}(\mathbf{F}, \Phi, \mathbf{U}, p, q, R) = \mathbf{0}, \tag{8.8}$$

where $\mathbf{U} = \mathbf{U}(\Sigma, N_2, N_3)$, $\mathbf{F} = (F, G)$, and $\mathbf{L_U}, L_\Phi, \mathbf{L_F}$ are linear differential operators w.r.t. z that depend on (p, q, R) and the other parameters [13]. Formally, we view (8.6)–(8.8) as a linear dynamical system for $\mathbf{U}$,

$$\partial_t \mathbf{U} = \mathbf{L}(\mathbf{U}, p, q, R).$$

where $\mathbf{L}$ is defined by substituting the solution $\Phi[\mathbf{U}, p, q, R]$, $\mathbf{F}[\mathbf{U}, p, q, R]$ of (8.7) and (8.8) into $\mathbf{L_U}$.

The neutral stability of the basic state occurs on a neutral stability surface in (p, q, R)-space, on which $\mathbf{L}$ has either a zero eigenvalue (stationary neutral stability surface) giving rise to a stationary bifurcation, or an imaginary eigenvalue (oscillatory neutral stability surface - ONSS) giving rise to a Hopf bifurcation. The parameter values of physical interest are (p_c, q_c, R_c) - the coordinates of the global R-minimum on the ONSS, for which the first instability is a Hopf Bifurcation, when R increases.

Owing to the symmetry $(z, n_3, v_3, \phi) \longrightarrow -(z, n_3, v_3, \phi)$, $\mathbf{L}$ has odd and even invariant subspaces spanned by modes of the form $\mathbf{U}_m$ and $\mathbf{V}_m$, respectively, where

$$\mathbf{U}_m = (a_1 \sin(2m-1)z, a_2 \sin 2mz, a_3 \cos(2m-1)z) \quad (m \geq 1),$$

$$\mathbf{V}_m = (a_1 \cos 2mz, a_2 \cos(2m+1)z, a_3 \sin 2mz) \qquad (m \geq 0).$$

For the parameter range considered here the instability occurs in the odd subspace. In this subspace, $\mathbf{L}$ is represented by an infinite matrix $\mathbf{M}$ composed of 3×3 blocks $M(m,n)$ defined by

$$M_{ij}(m,n) = \frac{2}{\pi} \int_{-\pi/2}^{\pi/2} \mathbf{L}(\mathbf{U}_{mi}) \cdot \mathbf{U}_{ni} dz, \quad 1 \leq i,j \leq 3,$$

where $\mathbf{U}_{mi}$ is the $\mathbf{U}_m$ mode with $a_j = \delta_{ij}$. Their computation is shown in [13].

For computing (p_c, q_c, R_c) numerically, we use a $3N \times 3N$ truncation of $\mathbf{M}$ by restricting m and n to $1 \leq m,n \leq N$. Numerical convergence with an accuracy up to five significant figures usually was observed for $N \leq 9$, in [13].

8.2 Numerical Results

We consider the NLC-matrix M of size $3N \times 3N$ resulting from the linear stability analysis, that depends on three parameters p, q and R, where R is the main bifurcation parameter, proportional to the strength of the electric field, and p, q are wave numbers in x, y direction; respectively. This matrix was computed by Oprea and Dangelmayr [13], and they kindly provided me the software for this matrix. From this matrix, we compute the surface $R = R(p,q)$ and find the global minimum value $R_c = R(p_c, q_c)$ for this surface. We use the bordered matrices algorithm to compute the surface $R = R(p,q)$. The independent parameters and the starting values of p and q are taken from [13]. The values $a = 0.8, b = 0.07, c = 0.1, \eta = -.9, \epsilon = 0.02, r = 0.5, \sigma = 1, \alpha = 0.3, k_2 = .7$ and $k_3 = 0.7$ are fixed. Let $p = 1.030151161$, $q = 0.7096795031$ be the starting values for the algorithm and we

will choose $N = 5$ then we use the bordered matrices method to compute R and we get $R = 7.11280244928449$. In this case, the 15×15 matrix $\mathbf{M}$ has 15 eigenvalues, 13 of which have negative real parts and the remaining two have purely imaginary parts that are ± 2.555917241105; therefore, we have $\nu = 6.53271294337779$. We use the values R and ν to be the starting values for computing the surface $R = R(p, q)$. We first find the local minimum $\tilde{R}$ of the surface $R = R(p, q)$ in a small neighborhood (p, q), say Λ_1 . Then we will take a larger domain to show that $\tilde{R}$ is in fact the global minimum of the surface $R = R(p, q)$. We will consider the cell $\Lambda_1 := [p - 0.2, q - 0.2] \times [p + 0.2, q + 0.2]$ to be our domain. Figure 8.1 plots the surface $R = R(p, q)$.

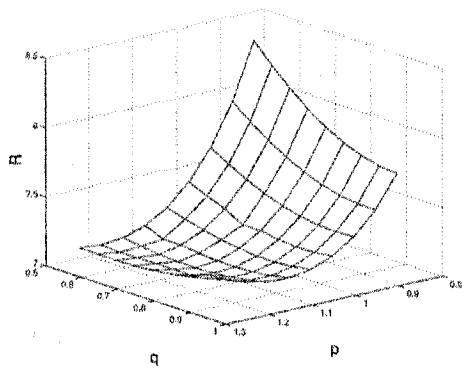
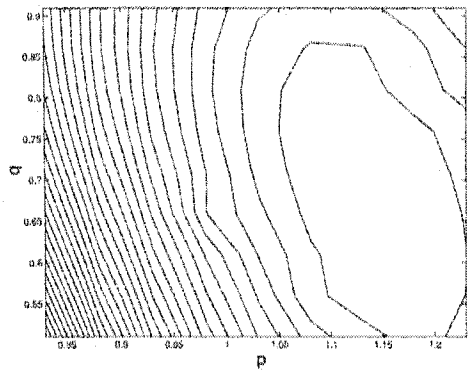


Figure 8.1: The surface $R = R(p, q)$.

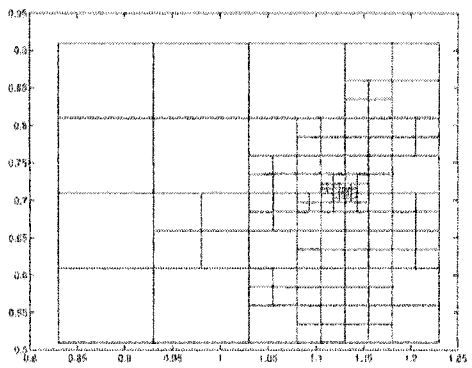
Our goal is to find the global minima of the surface $R = R(p, q)$. In order to find the global minima, we use Cell Exclusion Algorithm in the region Λ_1. We choose Tol = .001 and we get the global minimum is $R = 7.04302807$ at $(p, q) = (1.130151, 0.7065545)$. We plot the level sets of the surface and the refinements of the domain in Figure 8.2.

We now compute the global minima for the cases where $N = 7, 9$ and 10. The values of p, q and R are listed in Table 8.1.

Now we define a larger domain to insure that $\tilde{R}$ is in fact the global minimum. Moreover, we choose $M_k = \min\{\tilde{R}, R(m_k)\}$ for $k = 1, 2, \ldots$ so that CEA works



(a) The level sets of the function $R = R(p, q)$.



(b) Refinement of the domain.

Figure 8.2: Refinement of the domain Λ for $N = 5$.

N	p	q	R
7	1.130146	0.7065543	7.04433204968870
9	1.1301459	0.70655429	7.04502799195270
10	1.1301459	0.70655429	7.04593846328180

Table 8.1: Approximation of the global minimum of of the surface $R = R(p, q)$.

faster. We consider the domain $\Lambda_2 = [10^{-16}, 10] \times [10^{-16}, 10]$ and then we apply the CEA to Λ_2. We noticed that the algorithm immediately discarded the boxes $[5, 10] \times [10^{-16}, 10]$ and then $[10^{-16}, 5] \times [5, 10]$. After that CEA kept refining until we were left with a box that was contained in Λ_1. At this point we stopped the algorithm and concluded that $\tilde{R}$ is in fact the global minimum.

We will choose different values of the parameters and compute the global minimum for each case. All the parameters are chosen from [13]. We fix the values $a = 0.8, b = 0.07, c = 0.1, \eta = -1, \epsilon = -0.1, r = 0.5, \sigma = 1, \alpha = 0.3$. Then we choose $k_2 = .5$, $k_3 = .6766$ and $p = 1.413$. In this case we do not have a value for q; therefore, we choose $q = 1.2123$ as a starting value. We then compute the global minimum in the cell $\Lambda := [p - 0.2, q - 0.2] \times [p + 0.2, q + 0.2]$. The global minimum in this case is located on the boundary of the cell at the point $(p, q) = (1.411, 1.0123)$. Therefore, we expect the global minimum to be in another cell. We choose the next cell to be $\Lambda := [p - 0.2, q - 0.6] \times [p + 0.2, q - 0.2]$. In

this case, we find that the global minimum is also on the boundary of this cell; consequently, we continue the same procedure and we get the global minimum at the point $(p_c, q_c) = (1.4105, .00001)$ in the case where $N = 5$. We now compute the global minima for the cases where $N = 7, 9$ and 10. The values of p, q and R are listed in Table 8.2.

N	p	q	R
5	1.41048	.00001	6.67866662421935
7	1.41047	.000098	6.67890829710754
9	1.410465	.000098	6.67895519682318
10	1.410468	.000097	6.67896339958781

Table 8.2: The global minimum of of the surface $R = R(p, q)$.

We now change the values k_2 and k_3 and choose another starting point (p, q) and then compute the global minimum. Let $k_2 = 1.5, k_3 = 1.869$, $p = 1.2069$ and $q = .7654$. We compute the global minima for the cases where $N = 5, 7, 9$ and 10. The values of p, q and R are listed in Table 8.3.

N	p	q	R
5	1.2059	.00001	12.47467521704980
7	1.2059	.000099	12.47501607580518
9	1.2060	.00001	12.475700230495
10	1.2060	.000098	12.47598087723542

Table 8.3: Approximation of the global minimum of of the surface R.

We now choose the values $a = 0.98, b = 0.23, c = 0.38, \eta = -1, \epsilon = -0.1, r = 0.8, \sigma = .6, \alpha = 0.2$ $k_2 = 1.1$, $k_3 = .362$. Then we choose $p = 1.55376099544189$ and $q = .70309834593154$. We compute the global minima for the cases where $N = 5, 7, 9$ and 10. The values of p, q and R are listed in Table 8.4.

Finally, we choose the values $a = 0.8, b = 0.044375, c = 0.1, \eta = -0.85, \epsilon = 0.02, r = 0.8, \sigma = .8, \alpha = 0.3$ $k_2 = 1.3$, $k_3 = .8$. Then we choose $p = 1.08$ and $q = .79$. We compute the global minima for the cases where $N = 5, 7, 9$ and 10. The values of p, q and R are listed in Table 8.5.

N	p	q	R
5	1.553761	.703100	24.94507618845582
7	1.553759	.703099	24.94698883685102
9	1.553758	.703099	24.94750460660934
10	1.553757	.703098	24.94762161873586

Table 8.4: The global minimum of of the surface R.

N	p	q	R
5	1.005	0.865	10.94861882211243
7	1.0051	0.865	10.956020898222485
9	1.0051	0.8649	10.96629253770384
10	1.0049	0.8651	10.96938463281803

Table 8.5: An approximation of the global minimum.

We now give a summary of the way we computed the above results. The computer that was used to do all the computations is an Intel Pentium III with Mobile CPU 1000 MHz. We first choose N to be 5 and then use bordered matrices to define the function $R = R(p, q)$. The computer spends almost 12 minutes to compute the value first value R_1 for a given point (p_1, q_1). After that, the value R_1 is used as a starting value for the next point (p_2, q_2). This idea saves time for the computation of R_2, and in fact the computer takes almost 8 minutes to find R_2. This method gives a "black box" for the function $R = R(p, q)$. In order to find the global minimum, first define a domain Λ and then apply the CEA to Λ. We stop the CEA when the size of remaining cells is small and we are close to a minimum. Then we safely switch to the NMA because it is cheaper and faster. This procedure takes about 24 hours. After that we increase the value of N to 7 and do the same procedure but with (p_c, q_c) as a starting point, where (p_c, q_c) is the global minimum obtained from the case where $N = 5$. As soon as the CEA start refining close to (p_c, q_c), we switch to the NMA with (p_c, q_c) as a starting point. The numerical results confirm the convergence of this approach for increasing N.

Chapter 9

CONCLUSIONS AND OUTLOOK

Cellular Exclusion Algorithms (CEA) are a relatively new idea which show great promise as a versatile tool for handling the computation of global optima of very general functions which may even be defined as a result of some complicated process. This dissertation helps to bear out this promise. We have developed a practical computational cell elimination test using an extended analogue of the Jordan decomposition of functions of bounded variation. The extension involves the ϵ-increasing concept which is introduced and studied here.

A MATLAB implementation of a CEA using this exclusion test has been created. Numerical tests of the CEA have been made in which global minima were computed for the oscillatory neutral stability surface (ONSS) arising in the modelling of electroconvection in nematic liquid crystals. These results nourish the hope for extensions in the following directions:

1. We wish to develop the CEA for somewhat higher dimensions than we have done so far.

2. In the dissertation we usually first applied the CEA to find a first approximation of a global minimum and then converted over to a usually faster Nelder-Mead Algorithm (NMA). It would be very useful to find criteria for a good point at which this conversion from the CEA to the NMA could be made.

3. For the verification of the ϵ-increasing property, we need to have an appropriate mesh size. We would like to have a practical criterion for verifying the ϵ-increasing property.

4. We plan to apply a combination of CEA and NMA to further investigations of the similar minimization problems arising in the study of the Ekhaus stability of bifurcating waves in nematic electroconvection.

Bibliography

[1] C. Adams and J. Clarkson. Properties of functions $f(x, y)$ of bounded variation. *Trans. Amer. Math. Soc.*, 36 no. 4 (36), 711–730, 1934.

[2] C. Aliprantis and O. Burkinshaw. Principles of real analysis. *Academic Press, Inc., San Diego, CA*, 1998.

[3] E. Allgower, M. Erdmann and K. Georg. On the complexity of exclusion algorithms for optimization. *Journal of Complexity*, 18:573–588, 2002.

[4] E. Allgower and K. Georg. Numerical path following. *Handbook of numerical analysis*, Vol. 5, pages 3–207, North-Holland, 1997.

[5] J. Banaś and W. El-Sayed. Functions of generalized bounded variation. *Zeszyty Nauk. Politech. Rzeszowskiej Mat. Fiz.*, no. 12, 91–110, 1991.

[6] V. Chistyakov. Superposition operators in the algebra of functions of two variables with finite total variation. *Monatsh. Math.* 137(2):99–114, 2002.

[7] E. Chong and S. Zak. An introduction to optimization. *Wiley-Interscience Series in disceret mathmatics and optimization*, 2001.

[8] K. Chu, W. Govaerts and A. Spence. Matrices with rank deficiency two in eigenvalue problems and dynamical systems. *SIAM J. Numer. Anal.*, 31, no. 2, 524–539, 1994.

[9] J. Clarkson and C. Adams. On definitions of bounded variation for functions of two variables. *Trans. Amer. Math. Soc.*, no. 4 (35) 824–854, 1933.

[10] H. Cohen. A course in computational algebraic number theory. *Graduate Texts in Mathematics, Springer-Verlag, Berlin*, 1993.

[11] D. Cox, J. Little and D. O'Shea. Ideals, varieties, and algorithms. An introduction to computational algebraic geometry and commutative algebra. *Springer-Verlag, New York*, 1992.

[12] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM J. Numer. Anal.*, 34, no. 3, 922–938, 1997.

[13] G. Dangelmayer and I. Oprea. A bifurcation study of wave patterns for electroconvection in nematic liquid crystals. *Mol. Crystal Liqu. Crystal,* 2003.

[14] J. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM J. Optim. 1* no. 4, p:448–474, 1991.

[15] J. DePree and C. Swartz. Introduction to real analysis. em John Wiley and Sons, Inc., New York, 1988.

[16] Z. Du and B. Hassard. Precise computation of Hopf bifurcation and two applications. *Dynamics of Continuous, Discrete and Impulsive Systems. Series A. Mathematical Analysis,* 8(4):495–518, 2001.

[17] M. Erdmann. On the implementation and analysis of cellular exclusion algorithms. PhD thesis, Colorado State University, 2001.

[18] G. Ewing. Calculus of variations with applications. *Norton , New York,* 1969.

[19] H. Federer. Geometric measure theory. *Springer, New York,* 1969.

[20] G. Folland. Real Analysis: Modern Techniques and Their Applications, *Wiley-Interscience, New York.,* 1984.

[21] S. Galkina. Estimates for the Fourier-Haar coefficients of functions of two variables with bounded variation. *Izv. Vyssh. Uchebn. Zaved. Mat.* no. 2(45), 69–72, 2001.

[22] K. Georg. Improving the efficiency of exclusion algorithms. *Advances in Geometry,* 1:193–210, 2001.

[23] K. Georg. Matrix-free numerical continuation and bifurcation. *Numerical Functional Analysis and Optimization,* 22:303–320, 2001.

[24] A. Georgescu and I. Oprea. Bifurcation theory from the application point of view. *Timisoara University,* 1994.

[25] E.Godlewski and P. Raviart. Hyperbolic Systems of Conservation Laws, *Ellipses, Paris, France.,*1991.

[26] E. Godlewski and P. Raviart. Numerical Approximation of Hyperbolic Systems of Conservation Laws, *Springer, New York.,* 1996.

[27] W. Govaerts. Numerical methods for bifurcations of dynamical equilibria. *Society for Industrial and Applied Mathematics (SIAM),* Philadelphia, PA, 2000.

[28] J. Guckenheimer and M. Myers. Computing Hopf bifurcations II. Three examples from neurophysiology. *SIAM J. Sci. Comput,* 17:1275–1301, 1996.

[29] J. Guckenheimer, M. Myers and B. Sturmfels. Computing Hopf bifurcations I. *SIAM J. Numer. Anal.* 34:1–21, 1997.

[30] J. Hale and H. Kocak. Dynamics and bifurcations. *Texts in Applied Mathematics, 3. Springer-Verlag, New York,* 1991.

[31] T. Hildebrandt. Introduction to the theory of integration. *Pure and Applied Mathematics, Vol. XIII Academic Press, New York-London,* 1963.

[32] R. Horn and C. Johnson. Matrix analysis. *Cambridge University Press, Cambridge,* 1990.

[33] S. Jacoby, J. Kowalik and J. Pizzo. Iterative methods for nonlinear optimization problems. *Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J.*, 1972.

[34] K. Du and R. Kearfott. The cluster problem in multivariate global optimization. *J. Global Optim.,* 5, no. 3, 253–265, 1994.

[35] R. Kearfott. Rigorous global search: continuous problems. *Nonconvex Optimization and its Applications,* 13. Kluwer Academic Publishers, Dordrecht, 1996.

[36] C. Kelley. Iterative methods for optimization. *Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics* , Philadelphia, 1999.

[37] D. Knuth. The art of computer programming. Vol. 2. Seminumerical algorithms. *Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley Publishing Co., Reading, Mass.,* 1981.

[38] E. Kreyszig. Introductory functional analysis with applications. *Wiley Classics Library. John Wiley and Sons, Inc., New York,* 1989.

[39] M. Kubiček, and M. Marek. Computational methods in bifurcation theory and dissipative structures. *Springer Series in Computational Physics. Springer-Verlag* , New York, 1983.

[40] J. Lagarias, J. Reeds, M. Wright and P. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. em SIAM J. Optim., 9(1):112–147, 1999.

[41] S. Lang. Algebra. *Graduate Texts in Mathematics, Springer-Verlag, New York,* 2002.

[42] E. Lee and L. Markus. Foundations of Optimal Control Theory, *Wiley, New York.*, 1967.

[43] A. Leonov. Remarks on the total variation of functions of several variables and on a multidimensional analogue of Helly's choice principle. *Math. Notes,* 63, no. 1-2, 61–71, 1998.

[44] A. Leonov. On the use of functions of several variables with bounded variation for piecewise-uniform regularization of ill-posed problems. *Dokl. Akad. Nauk 351,* no. 5, 592–595, 1996.

[45] J. Malek, J. Necas, M. Rokyta and M. Ruzicka. Weak and Measure-Valued Solutions to Evolutionary PDEs, *Chapman and Hall, London, UK.,* 1996.

[46] K. McKinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point.*SIAM J. Optim.*, 9(1):148–158, 1999.

[47] J. Moiolaand G. Chen. Hopf bifurcation analysis. A frequency domain approach. *World Scientific Series on Nonlinear Science.* Series A: Vol 21, World Scientific Publishing Co., Inc., River Edge, NJ, 1996.

[48] R. Moore. Methods and applications of interval analysis. *SIAM Studies in Applied Mathematics, 2,* 1979.

[49] G. Moore , T. Garratt and A. Spence. The numerical detection of Hopf bifurcation points. *Continuation and bifurcations: numerical techniques and applications Adv. Sci. Inst. Ser. C Math. Phys. Sci.*, 313:227–246, 1990.

[50] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal,* Vol. 7: 308–313, 1996.

[51] W. Press, S. Teukolsky, W. Vetterling and B. Flannery. Numerical recipes in C++. The art of scientific computing. Second edition. *Cambridge University Press, Cambridge,* 2002.

[52] C. Price, I. Coope and D. Byatt. A convergent variant of the Nelder-Mead algorithm. *J. Optim. Theory Appl.*, 113(1): 5–19, 2002.

[53] D. Roose. An algorithm for the computation of Hopf bifurcation points in comparison with other methods. *Proceedings of the international conference on computational and applied mathematics. J. Comput. Appl. Math.*, 12/13: 517–529, 1985.

[54] D. Roose and V. Hlavacek A direct method for the computation of Hopf bifurcation points. *SIAM J. Appl. Math.* 45(6) 879–894, 1985.

[55] H. Royden. Real analysis. *The Macmillan Co., New York Collier-Macmillan Ltd., Canada,* 1963.

[56] T. Sederberg and J. Zheng. Algebraic methods for computer aided geometric design. *Handbook of computer aided geometric design,* 363–387, North-Holland, Amsterdam, 2002.

[57] L. Trefethen and D. Bau. Numerical linear algebra. *Society for Industrial and Applied Mathematics , Philadelphia,* 1997.

[58] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optim.* 7(1), 1–25, 1997.

[59] F. Verhulst. Nonlinear differential equations and dynamical systems, *Springer-Verlag, Berlin,* 1996.

[60] W. Wade. An introduction to analysis. *Prentice-Hall. Inc, New Jersey,* 1995.

[61] B. Werner. Computation of Hopf bifurcation with bordered matrices. *SIAM J. Numer. Anal.*, 33:435–455, 1996.

[62] S. Wiggins. Introduction to applied nonlinear dynamical systems and chaos. *Texts in Applied Mathematics, 2. Springer-Verlag, New York,* 1990.

[63] M. Wright. Direct search methods: once scorned, now respectable. *Numerical analysis 1995, Pitman Res. Notes Math. Ser.* 344, 191–208, 1996.

[64] X. Xiao Ping and Z. Bo. Properties of set-valued functions of bounded variation in a Banach space. *J. Harbin Inst. Tech.* no. 3(107), 102–105, 1991.

[65] H. Xu, V. Janovský and B. Werner. Numerical computation of degenerate Hopf bifurcation points. *ZAMM Z. Angew. Math. Mech.* 78, no. 12, 807–821, 1998.

[66] Z. Xu, J. Zhang and W Wang. A cell exclusion algorithm for determining all the solutions of a nonlinear system of equations. *Appl. Math. Comput.*, 80, no. 2-3, 181–208, 1996.

[67] R. Ye. A new approach for the computation of Hopf bifurcation points. *Appl. Math. Mech.* 21(11):1300–1307, 2000.

[68] P. Zingano and S. Steinberg. On the Hardy-Littlewood theorem for functions of bounded variation. *SIAM J. Math. Anal.* no. 5(33), 1199–1210, 2002.

Appendix A

Decomposition Algorithm and Cell Exclusion Algorithm

A.1 Decomposition Algorithm

In this appendix we present computer codes to compute the functions p and n for Decomposition Algorithms 3.2.1 and 3.4.1. The first code is for the case of one dimension. The second is for the case of two dimensions. Both codes are written using Matlab implementation.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This Algorithm computes the  functions p and n such that %%%
%%% p and n are increasing and f:[a,b]-->R such that         %%%
%%% f = p-n                                                   %%%
%%% INPUT: f, a, x, and m                                     %%%
%%% OUTPUT: p and n                                           %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  [p,n] = bv1(a,x,m)

p=f1(a); % initial value of p
n=0; % initial value of n
dx=(x-a)/(m+1); % the mesh size
for i=1:m+1
    s=f(a+i*dx)-f1(a+(i-1)*dx);
    if  s>0
        p=p+s;
    elseif s<0
        n=n-s;
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This Algorithm computes the  functions p and n such that %%%
%%% p and n are increasing and f:[a1,b1] x [a2,b2]-->R such  %%%
%%% that  f = p-n                                            %%%
%%% INPUT: f, a, x, and m                                    %%%
%%% OUTPUT: p and n                                          %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  [pf,nf] = bv(a,x,y,m)

p=-f(a(1),a(2)); % initial value of p
n=0; % initial value of p
dx=(x-a(1))/m
dy=(y-a(2))/m;

for i=1:m
    for j=1:mm
        s=f(a(1)+i*dx,a(2)+j*dy)...
            +f(a(1)+(i-1)*dx,a(2)+(j-1)*dy)...
            -f(a(1)+(i-1)*dx,a(2)+j*dy)...
            -f(a(1)+i*dx,a(2)+(j-1)*dy);
        if  s>0
            p=p+s;
        elseif s<0
            n=n-s;
        end
    end
end

% to compute f(x,a(2))
for i=1:m
    s=f(a(1)+i*dx,a(2))-f(a(1)+...
        (i-1)*dx,a(2));
    if  s>0
        pfx=pfx+s;
    elseif s<0
        nfx=nfx-s;
    end
end

% to compute f(a(1),y)
for i=1:mm
    s=f(a(1),a(2)+i*dy)-f(a(1),...
         a(2)+(i-1)*dy);
```

```
    if  s>0
         pfy=pfy+s;
    elseif s<0
        nfy=nfy-s;
    end
end

nf=n+nfx+nfy;
pf=p+pfx+pfy;
```

A.2 Cell Exclusion Algorithm

In this appendix we present computer codes to compute the the global minimum of a function f for Cell Exclusion Algorithm 2.2.1. The first code is for the case of one dimension. The second is for the case of two dimensions. Both codes are written using Matlab implementation.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This Algorithm computes the global minima of a function %%%
%%% f:[a,b]-->R such that f is monotonically decomposable   %%%
%%% f = g - h                                               %%%
%%% INPUT: f, a, b                                          %%%
%%% OUTPUT: the global minima                               %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function cell_ex(a,b,Tol,choose)

% Tol = tolerance
% choose = 1 or 2
% if choose = 1, g and h are computed by D. A.
% if choose = 2 g and h are computed analytically
% Initialization
i=1;
j=1;
Check=1; % # of M. C. Checks
s=a/2+b/2;% the midpoint
global_min=f(s); % Approximation of the global minima
cellab(i,j,1:2)=[a,b]; % to keep track on the cells at level i
NumCell(i)=1; % number of cells at level i
IntLength=b-a; % the length of the interval ||d||

while  IntLength > Tol % While Loop if ||d|| < Tol
    LastIndex=1;
    NumCell(i+1)=0;
    for k=1 : NumCell(i)
        % compute g and h
        if choose == 1 % use D. A. to compute g and h
           [gb,hb]=bv1(cellab(i,k,1),cellab(i,k,2),100);
           ss=f(cellab(i,k,1)) - hb; % the condition g - h
        elseif choose==2
           ss=g(cellab(i,k,1))-h(cellab(i,k,2));
        end

        if global_min >= ss
```

```
            NumCell(i+1)=NumCell(i+1)+2;
            cellab(i+1,LastIndex,1:2)=[cellab(i,k,1),...
                    (cellab(i,k,1)+cellab(i,k,2))/2];
            s= (3*cellab(i,k,1)+cellab(i,k,2))/4;
            M=f(s);
            if M < global_min;
                s1=s;
                global_min = M;
            end
            Check=Check+1;
            LastIndex=LastIndex+1;
            cellab(i+1,LastIndex,1:2)=[(cellab(i,k,1)...
                    +cellab(i,k,2))/2,cellab(i,k,2)];
            s= (cellab(i,k,1)+3*cellab(i,k,2))/4;
            M=f(s);
            if M<global_min
                s1=s;
                global_min=M;
            end
            Check=Check+1;
            LastIndex=LastIndex+1;
        end
    end
    IntLength=IntLength/2;
    i=i+1;
end
fprintf('%g  & %3.6f & %g & %g  '...
    ,i-1,global_min,LastIndex-1,Check)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This Algorithm computes the functions p and n such that %%%
%%% p and n are increasing and f:[a,b] x [a,b]-->R such     %%%
%%% that  f = p-n                                           %%%
%%% INPUT: f, a, x, and m                                   %%%
%%% OUTPUT: p and n                                         %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function cellex(a,b,Tol,choose)

% Initialization
i=1;
j=1;
Check=1;
s1=a/2+b/2;
global_min=f(s1(1),s1(2));

cellab(i,j,1:4)=[a,b];
NumCell(i)=1;
IntLength=norm(b-a,inf);

while  IntLength > Tol
    LastIndex=1;
    NumCell(i+1)=0;
    for k=1 : NumCell(i)
        if choose==1
            [gb,hb]=bv(cellab(i,k,1),cellab(i,k,2),...
                cellab(i,k,3),cellab(i,k,4),5);
            ss=f(cellab(i,k,1),cellab(i,k,2))-hb;
        elseif choose==2
            ss=g(cellab(i,k,1),cellab(i,k,2))...
                    -h(cellab(i,k,3),cellab(i,k,4));
        end

        if global_min >= ss
            NumCell(i+1)=NumCell(i+1)+4;
            Mid=([cellab(i,k,1),cellab(i,k,2)]+...
                [cellab(i,k,3),cellab(i,k,4)])/2;
            cellab(i+1,LastIndex,1:4)=[cellab(i,k,1),...
                    cellab(i,k,2),Mid];
            s= ([cellab(i,k,1),cellab(i,k,2)]+Mid)/2;

            M=f(s(1),s(2));
            if M < global_min
```

```
            s1=s;
            global_min=M;
        end
        LastIndex=LastIndex+1;
        cellab(i+1,LastIndex,1:4)=[Mid(1),cellab(i,k,2),...
              cellab(i,k,3),Mid(2)];
        s= ([Mid(1),cellab(i,k,2)]+...
              [cellab(i,k,3),Mid(2)])/2;

        M=f(s(1),s(2));
        if M<global_min
            s1=s;
            global_min=M;
        end

        LastIndex=LastIndex+1;
        cellab(i+1,LastIndex,1:4)=[cellab(i,k,1),Mid(2),...
              Mid(1),cellab(i,k,4)];
        s= ([cellab(i,k,1),Mid(2)]+...
              [Mid(1),cellab(i,k,3)])/2;

        M=f(s(1),s(2));
        if M < global_min
            s1=s;
            global_min=M;
        end

        LastIndex=LastIndex+1;
        cellab(i+1,LastIndex,1:4)=[Mid,...
              cellab(i,k,3),cellab(i,k,4)];
        s= (Mid+...
              [cellab(i,k,3),cellab(i,k,4)])/2;

        M=f(s(1),s(2));
        if M<global_min
            s1=s;
            global_min=M;
        end
        LastIndex=LastIndex+1;
        Check=Check+4;
    end
end
IntLength=IntLength/2
i=i+1;
```

```
end
fprintf('%g & %3.6f&%g&%g'...
    ,i-1,global_min,LastIndex-1,Check)
```

Appendix B

NELDER-MEAD ALGORITHM

In this appendix we present two computer codes that compute a local minimum of a function using the Nelder-Mead algorithm that was discussed in Chapter 4. This algorithm can find a local minima of a function if the initial point is close to the minimum. It only requires the value of the function at a point. The idea behind this algorithm is to compute the value of the function at three points and then to drop the one that has maximum function value and replace it with a "suitable" point. Sometimes it keeps the point that has minimum function value and replaces the other two. The first code was written in Maple, and the second one is a Matlab code. Both codes have the same inputs and outputs.

Procedure: We start our procedure by defining the initial simplex which depends on x_1. After that, we compute the value of the function at each vertex of the simplex. Then we decide which step should we take; reflection, expansion, contraction or shrinking and consider the new simplex. We do this until stopping criterion is satisfied.

INPUT: The function f, a starting point x_1, scaling value M and tolerence.

OUTPUT: Number of iterations and the vertices of the last simplex.

B.1 Maple Code

```
> ##################################################################
> ###  This program uses NM Method to find a local minimum       ###
> ###   of a function f : R^2 --> R.                              ###
> ###  INPUT: f, x1, M and tol.                                   ###
> ###  OUTPUT: number of iterations, x1, x2 and x3.               ###
> ##################################################################

> restart:
            ### Defining f, and the simplex x1, x2, x3. ###
> f:=(x,y)->100*(y-x^2)^2+(1-x)^2:
> x1:=(-1,1):
> M:=.0001:
> tol:=10^(-16):
> p1:=(sqrt(3)+1)/2*sqrt(2)*M:
> q1:=(sqrt(3)-1)/2*sqrt(2)*M:
> x2:=(p1+x1[1], q1+x1[2]):
> x3:=(q1+x1[1], p1+x1[2]):
            ### The midpoint x0 and the stopping criterion ###
> x0:=(x2+x3)/2:
> cri:= evalf(((f(x1)-f(x0))^2+(f(x2)-f(x0))^2+(f(x3)-f(x0))^2)/2):

            ### Starting the loop until cri <= tol or the   ###
            ### maximum number of iterations (100) achieved ###
> for  i from 0 while (cri> tol and i<101) do
>
>    s1:=evalf(f(x1)):
>    s2:=evalf(f(x2)):
>    s3:=evalf(f(x3)):
>
>          ### Ordering the vertices ###
>    if ( s1>=s2 and s1>=s3)  then
>        xh:=x1:
>        if s2>=s3 then
>           xs:=x2:
>           xl:=x3:
>        else
>           xs:=x3:
>           xl:=x2:
>        end if
>    elif (s2>=s1 and s2>=s3 ) then
>        xh:=x2:
>        if s1>=s3 then
```

```
>            xs:=x1:
>            xl:=x3:
>         else
>            xs:=x3:
>            xl:=x1:
>         end if
>     else
>         xh:=x3:
>         if s2>=s1 then
>            xs:=x2:
>            xl:=x1:
>         else
>            xs:=x1:
>            xl:=x2:
>         end if
>     end if:
>
>     x0:=(xs+xl)/2:
>     xr:=2*x0-xh:
>     sr:=evalf(f(xr)):
>     ss:=evalf(f(xs)):
>     sl:=evalf(f(xl)):
>
>     if sr<sl then
>          xe:=2*xr-x0:
>          se:=evalf(f(xe)):
>          if se<sl then
>              xh:=xe:    ### Expansion
>          else
>              xh:=xr:    ### Reflection
>          end if
>     else
>          if ss>=sr then
>              xh:=xr:    ### Reflection
>          else
>             sh:=evalf(f(xh)):
>             if sr<sh then
>                xh:=xr:    ### Reflection
>             end if:
>             xc:=(xh+x0)/2:
>             sc:=evalf(f(xc)):
>             if sc>=sh then
>                xs:=(xs+xl)/2:    ### Shrinking
>                xh:=(xh+xs)/2:
```

```
>            else
>               xh:=xc:   ### Contraction
>            end if
>        end if
>    end if:
>    x1:=xh:
>    x2:=xs:
>    x3:=xl:
>    s1:=evalf(f(x1)):
>    s2:=evalf(f(x2)):
>    s3:=evalf(f(x3)):
>
>    fmin:=min(s1,s2,s3):
>     if s1=fmin then
>         l:=(x1[1], x1[2], fmin):
>     elif s2=fmin then
>         l:=(x2[1], x2[2], fmin):
>     else s3=fmin:
>         l:=(x3[1], x3[2], fmin):
>     end if:
>
> end do: ### End loop
>
           ### Final print of the vertex where the  ###
           ### minimum value of f occurs            ###
> x1_value:=evalf(l[1]);
> x2_value:=evalf(l[2]);
> fmin_value:=l[3];
> number_of_iteration:=i;
```

B.2 Matlab Code

```
function [l] = nm(x1)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This program uses the Nelder-Mead Algorithm to find a     %%%
%%% local minimum for a 2-variable function z = f(x,y).       %%%
%%% The idea for this algorithm is to compare the values of   %%%
%%% 3 points and drops the one that has maximum value and     %%%
%%% replaces it with a point that has value less than the     %%%
%%% dropped one.                                              %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% INPUT: the initial point
%%% OUTPUT: the minimum point (x,y,z)


M=1; % to scale the choice of p and q.
tol=10^-12; % stopping tolerance
p=(sqrt(3)+1)/2*sqrt(2)*M;
q=(sqrt(3)-1)/2*sqrt(2)*M;
x2=[p+x1(1) q+x1(2)]; % the other 2 points are defined.
x3=[q+x1(1) p+x1(2)];
x0=(x2+x3)/2; % the midpoint
i=0;

while % starting loop
(((f(x1)-f(x0))^2+(f(x2)-f(x0))^2+(f(x3)-f(x0))^2)/2 > tol &
i<100)
    i=i+1; % to compute the number of iteration

    %%% we define xh, xs and xl such that
    %%% f(xh) >= f(xs) >= f(xl)
    if (f(x1)>=f(x2) & f(x1)>=f(x3) )
        xh=x1;
        if f(x2)>=f(x3)
            xs=x2;
            xl=x3;
        else
            xs=x3;
            xl=x2;
        end
    elseif (f(x2)>=f(x1) & f(x2)>=f(x3) )
        xh=x2;
```

```
    if f(x1)>=f(x3)
        xs=x1;
        xl=x3;
    else
        xs=x3;
        xl=x1;
    end
else
    xh=x3;
    if f(x2)>=f(x1)
        xs=x2;
        xl=x1;
    else
        xs=x1;
        xl=x2;
    end
end

%%% Replace the worst vertix.
x0=(xs+xl)/2;
xr=2*x0-xh;
if f(xr)<f(xl)
    xe=2*xr-x0;
    if(f(xe)<f(xl))
        xh=xe;   %%% Expansion
    else
        xh=xr;   %%% Reflection
    end
else
    if f(xs)>=f(xr);
        xh=xr;   %%% Reflection
    else
        if f(xr)<f(xh)
            xh=xr;   %%% Reflection
        end
        xc=(xh+x0)/2;
        if f(xc)>=f(xh)
            xs=(xs+xl)/2;   %%% Shrinking
            xh=(xh+xs)/2;
        else
            xh=xc;   %%% Contraction
        end
    end
end
```

```
    %%% the minimum point (x,y,z)
    x1=xh;
    x2=xs;
    x3=xl;
    fmin=min([f(x1),f(x2),f(x3)]);
    if f(x1)==fmin;
        l=[x1(1) x1(2) fmin];
    elseif f(x2)==fmin;
        l=[x2(1) x2(2) fmin];
    else f(x3)==fmin;
        l=[x3(1) x3(2) fmin];
    end
end

%%% print the result
fprintf('number of itertion is: %d', i)
if i>=100
    fprintf('\nAlgorithm reaches its maximum number of iteration 100')
end

%%% Defining f
function [x] = f(x)
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

APPENDIX C

COMPUTER CODE FOR HOPF BIFURCATION

In this appendix we present two computer codes to compute Hopf bifurcation point. The first one is a Maple code which uses the Resultant Method. The second is a Matlab code that uses the Bordered Matrices Method. For each algorithm we describe how the algorithm works, what the input and output are, and finally we present the code.

C.1 Resultants Algorithm

Included here is the Maple code designed to compute a Hopf bifurcation point for a given matrix A. This code uses the Resultant Method as described in Chapter 6.

A procedure is defined for a given matrix A which depends continuously on p, q and R. It locates the Hopf point by computing R for given values of p and q, where R is the minimum positive value for which the matrix has a Hopf point.

Procedure: We first define the matrix A which depends on p, q and R, then we substitute $p = p_{initial}$ and $q = q_{initial}$. After that we compute the matrix $(\lambda I - A)$, then we compute the coefficients of λ that depend on R. Secondly, we define the Sylvester Matrix S and its subresultants S_0 and S_1. Finally, we make the test $\det(S) = 0$ and $\det(S_0) \cdot \det(S_1) > 0$, to find all R values that satisfy these conditions and take R to be the minimum positive value.

INPUT: Hopf parameter values $p_{initial}$ and $q_{initial}$. N is the size of the matrix.

OUTPUT: R will be considered as a function of p and q.

```
> ##################################################################
> ###  This program uses Resultant Method to find the Hopf       ###
> ###  bifurcation point for a matrix A(p,q,R).                   ###
> ###  INPUT: p, q and N.                                        ###
> ###  OUTPUT: R.                                                ###
> ##################################################################
> restart:
```

```
> R_value:=proc(p_initial,q_initial,N)

>   local   A,A_R,i,j,d,re,ro,S,S0,S1,det_S,det_S0,det_S1,dim,
>           first_condition,solution,test,hopf_test,index_of_i,
>           first_min,mini,ind:
>   with(linalg):
>   A:=matrix(N,N,[....]): ### A:=A(p,q,R)
>   ### Evaluate the matrix at fixed parameters
>   A_R:=evalf((subs(p=p_initial,q=q_initial,evalm(A)))):

>   ### Compute the coefficients lambda for the matrix
>   ### (lambda I - A_R)
>   for i from 1 to  N do
>     A_R[i,i]:=A_R[i,i]-lambda
>   od:
>   evalm(simplify(A_R)):
>   d:=(-1^N)*simplify(evalf(det(A_R))):
>   for i from 0 to N do
>     c[i]:=coeff(d,lambda,i);
>   od:

>   ##################################################################
>   ### Compute the Sylvester Matrix and its Subresultants   ###
>   ##################################################################
>   re:=0:ro:=0:
>           #######
>   ###  If N is even    ###
>           #######
>   if N mod 2 = 0 then
>     for i from 0 to (N/2)-1 do
>       re:= re + c[N-2*i]*z^i:
>       ro:= ro + c[N-2*i-1]*z^i:
>     od:
>     re:=re + c[0]*z^(N/2):
>     c[N]:=1:
>     S:=sylvester(re,ro,z):
>     if N=2 then
>       S0:=c0:S1:=1:
>     else
>       S0:=matrix(N-3,N-3):
>       for i from 1 to N-3  do
>         for j from 1 to N-3 do
>           if i < (N-2)/2 then
>             S0[i,j]:=S[i+1,j+2]:
```

```
>           else
>             S0[i,j]:=S[i+2,j+2]:
>           fi:
>         od:
>       od:
>       S1:=matrix(N-3,N-3):
>       for i from 1 to N-3  do
>         for j from 1 to N-3 do
>           if j<2 then
>             if i < (N-2)/2 then
>               S1[i,j]:=S[i+1,j+1]:
>             else
>               S1[i,j]:=S[i+2,j+1];
>             fi:
>           else
>             if i < (N-2)/2 then
>               S1[i,j]:=S[i+1,j+2]:
>             else S1[i,j]:=S[i+2,j+2];
>             fi:
>           fi:
>         od:
>       od:
>     fi:
>           #######
>   ###  If N is odd    ###
>           #######
>   else
>     for i from 0 to (N-1)/2 do
>       re:= re + c[N-2*i-1]*z^i:
>       ro:= ro + c[N-2*i]*z^i:
>     od:
>     c[N]:=1:
>     S:=sylvester(re,ro,z):
>     if Ndim=3 then
>       S0:=c1:S1:=1:
>     else
>     S0:=matrix(N-3,N-3):
>     for i from 1 to N-3  do
>       for j from 1 to N-3 do
>         if i < (N-1)/2 then
>           S0[i,j]:=S[i+1,j+2]:
>         else
>           S0[i,j]:=S[i+2,j+2];
>         fi:
```

```
>       od:
>     od:
>     S1:=matrix(N-3,N-3):
>     for i from 1 to N-3  do
>       for j from 1 to N-3 do
>         if j<2 then
>           if i < (N-1)/2 then
>             S1[i,j]:=S[i+1,j+1]:
>           else
>             S1[i,j]:=S[i+2,j+1];
>           fi:
>           else
>             if i < (N-1)/2 then
>               S1[i,j]:=S[i+1,j+2]:
>             else
>               S1[i,j]:=S[i+2,j+2];
>             fi:
>           fi:
>         od:
>       od:
>     fi:
>   fi:

>   ##################################################################
>   ###       Conditions for Hopf bifurcation point          ###
>   ##################################################################
>   det_S:=det(S):
>   if N=1 then
>     D_S0:=1:
>     D_S1:=cx[1]:
>   else
>     det_S0:=det(S0):
>     det_S1:=det(S1):
>   fi:
>   dim:=degree(det_S,R);
>   first_condition:=simplify(evalf(det_S)):
>   solution:=[solve(hopf)]:
>   test:=vector(dim):
>   ### We check for all Rs that are positive real numbers
>   for i from 1 to dim do
>     if (Im(solution[i]) = 0 and Re(solution[i])>0) then
>       test[i]:=Re(solution[i]);
>     else
>       test[i]:=0;
```

```
>     fi:
>   od;
>   ### For each positive R we check  if  | S0 * S1 | > 0.
>   for i from 1 to dim do
>     hopf_test:=subs(R=test[i],det_S0*det_S1):
>     if (hopf_test> 0 and test[i]<>0) then
>       test[i]:=solution[i];
>     else
>       test[i]:=0
>     fi:
>   od:
>   ### Now we define R to be the minimum of all Rs.
>   index_of_i:=1;
>   first_min:=test[1];
>   for i from 1 by 1 while test[i] = 0 do
>     first_min:=test[i+1]:
>     index_of_i:=i+1:
>   od:
>   mini:=first_min;
>   ind:=index_of_i+1 ;
>   for i from ind to N do
>     if mini > test[i] and test[i]<>0 then
>       mini :=test[i];
>     fi;
>   od:
>   mini:
> end proc:
```

C.2 Bordered Matrices Algorithm

The following is a Matlab code designed to compute a Hopf bifurcation point for a given matrix A. This code uses Bordered Matrices as described in Chapter 7.

In this program, the matrix A depends continuously on one parameter R, where all other parameters are assigned to real values. It locates the Hopf point by solving the system 7.4.

Procedure: We first define the matrix A which depends on R. After that we compute the value of ν for fixed R_0 such that $\beta(\nu, R_0) = 0$. Then we compute the value of R such that $\alpha(\nu(R), R) = 0$.

INPUT: Initial values ν_0 and R_0.

OUTPUT: New values ν and R such that 7.4 holds.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% This Algorithm solves for a Hopf bifurcation point                        %%%
%%% using the Werner method.                                                  %%%
%%% INPUT: Initial values nu_0 and R_0                                         %%%
%%% OUTPUT: nu and R such that beta(nu,R)=alpha(nu,R)=0                        %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% A is defined as a function of R
function A = A_matrix(R)
%%% matrix is defined to be the Bordered Matrix
function M = matrix(R,nu)
A =A_matrix(R);
n=length(A);
l=zeros(1,n);
l(1,1)=1;
r=zeros(n,1);
r(2,1)=1;
M=[A^2+nu*eye(n),A*r, r ;
   l*A          , 0 , 0 ;
   l            , 0 , 0 ];
```

```
%%% beta function produses the value of nu such that
%%% beta(nu,R_0) is a solution of the Bordered System
%%% where R_0 is fixed
function nu_value=beta(nu,R_0)
n=length(A);
b=zeros(n+2,1);
b(n+1,1)=1;
x=matrix(R_0,nu)\b;
nu_value=x(n+2);

%%% nur function solves for beta(nu,R_0)=0
function beta_0=nur(nu,R_0)
x=nu;
a=x;
fa = beta(a,R_0);
b=a-.1;
fb = beta(b,R_0);
%%% Secant-Like method is used to solve for beta(nu,R_0)=0
p0=a;
p1=b;
q0=fa;
q1=fb;
while abs(q1)>10^(-12)
    p=p1-q1*(p1-p0)/(q1-q0);
    p0=p1;
    q0=q1;
    p1=p;
    q1=beta(p,R);
end
beta_0=p1;

%%% alpha function produses the value of R such that
%%% alpha(nu(R),R) is a solution of the Bordered System
function R_value=alpha(nu,R)
n=lingth(A);
b=zeros(n+2,1);
b(n+1,1)=1;
x=matrix(R,nur(nu,R))\b;
nuvalue=x(n+1);

%%% tau function solves for alpha(nu(R),R)=0
function [R_value,nu_value] = tau(nu,R)
a=R;
x=a;
```

```
fa=alpha(nu,a);
b=a-.1;
fb=alpha(nu,b);
%%% Secant-Like method is used to solve for alpha(nu(R),R)=0
p0=a;
p1=b;
q0=fa;
q1=b;
while abs(q1)>10^(-12)
    p=p1-q1*(p1-p0)/(q1-q0);
    p0=p1;
    q0=q1;
    p1=p;
    q1=alpha(nu,p);
end
R_value = p1;
nu_value=nur(nu,p1);
```