Computation of the Solutions of Nonlinear Polynomial Systems

by

Evan Conway Sherbrooke

S.B. Mathematics, Massachusetts Institute of Technology
June 1990

Submitted to the Department of Ocean Engineering
in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN NAVAL ARCHITECTURE AND MARINE ENGINEERING

and

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

October 1993

Author .....
Department of Ocean Engineering
October 15, 1993

Certified by ... .
Nicholas M. Patrikalakis
Associate Professor of Ocean Engineering
Thesis Supervisor

Read by .....
John N. Tsitsiklis
Professor of Electrical Engineering
Thesis Reader

Accepted by .....
A. Douglas Carmichael
Chairman, Departmental Committee on Graduate Students
Department of Ocean Engineering

Accepted by .....
Frederic R. Morgenthaler

Chairman, Departmental Committee on Graduate Students
Department of Electrical Engineering and Computer Science

Computation of the Solutions of Nonlinear Polynomial Systems

by

Evan Conway Sherbrooke

Submitted to the Department of Ocean Engineering
on October 15, 1993, in partial fulfillment of the
requirements for the degrees of
MASTER OF SCIENCE IN NAVAL ARCHITECTURE AND MARINE ENGINEERING
and
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Abstract

A fundamental problem in computer aided design is the efficient computation of all roots of a system of nonlinear polynomial equations in n variables which lie within an n-dimensional box. We present two techniques designed to solve such problems, which rely on representation of polynomials in the multivariate Bernstein basis and subdivision. In order to isolate all of the roots within the given domain, each method uses a different scheme for constructing a series of bounding boxes; the first method projects control polyhedra onto a set of coordinate planes, and the second employs linear optimization. We also examine in detail the local convergence properties of the two methods, proving that the former is quadratically convergent for $n = 1$ and linearly convergent for $n > 1$, while the latter is quadratically convergent for all n. Worst-case complexity analysis, as well as analysis of actual running times are performed. The techniques are applied to the problem of determining stationary points of a scalar field. Classification of these stationary points is also described. In addition, the related problem of tracing sets of non-isolated roots is treated theoretically. Finally, several examples are presented which illustrate some of the applications of our techniques as well as their performance.

Thesis Supervisor: Nicholas M. Patrikalakis
Title: Associate Professor of Ocean Engineering

Thesis Reader: John N. Tsitsiklis
Title: Professor of Electrical Engineering

Acknowledgements

I would like to dedicate this thesis to my parents and brother, who have provided love and support throughout my life. My parents will be pleased to note that this thesis is costing them nothing. I would also like to acknowledge my father's assistance on the theory of Linear Programming, and his creative variation of the protestant work ethic which has always been an inspiration.

I must acknowledge my advisor, Professor N. M. Patrikalakis, for his guidance throughout the process, Professor J. N. Tsitsiklis for his patience in reading this thesis, and Professor A. W. Drake for making the dual degree process totally painless. Several co-workers have provided valuable assistance, including Mr. C.-Y. Hu, Dr. T. Maekawa, and Ms. J. Zhou. They have all helped to test or enhance the algorithms described herein, and their assistance is greatly appreciated.

This work was supported, in part, by the M.I.T. Sea Grant College Program; the Office of Naval Research; and the National Science Foundation under grant numbers NA90AA-D-SG-424; N00014-91-J-1014; DDM-9215411, and IRI-9224640; and by an NSF Graduate Fellowship.

Contents

Acknowledgements **3**

1 Introduction **8**

2 Literature Review **10**

3 Notation and Definitions **13**

4 Formulation **16**

5 The Projected-Polyhedron Intersection Algorithm **20**

5.1 Motivation for the Algorithm 20

5.2 Generating the Bounding Box 23

5.3 Analysis 26

5.3.1 The Bounding-Box Tree 27

5.3.2 Complexity Analysis 28

5.3.3 Establishing Bounds 29

5.3.4 Quadratic Convergence in One Dimension 33

5.3.5 Convergence in Higher Dimensions 34

6 The Linear Programming Root Isolation Algorithm **37**

6.1 Generating the Bounding Box 37

6.2 Analysis 41

6.2.1 Complexity Analysis 41

6.2.2 Quadratic Convergence in n Dimensions 42

7 Optimization Problems **45**

7.1 Classification of Isolated Stationary Points 46
7.2 Identification of Non-isolated Stationary Points 51
7.3 Remarks 53

8 Examples and Applications **55**
8.1 Numerical Examples 55
8.1.1 Intersecting Two Simple Planar Curves 55
8.1.2 The Wilkinson Polynomial 58
8.1.3 Significant Points of a Planar Algebraic Curve 58
8.1.4 Stationary Points of a Distance Function 60
8.1.5 A Stationary Point Problem at a Strict Tolerance 62
8.1.6 A Distance Problem with Tangency 62
8.2 Other Applications 63
8.2.1 Systems of Piecewise Polynomials 63
8.2.2 More Equations than Unknowns, or Vice-Versa 64
8.2.3 Finding Complex Roots 64
8.3 Choosing an Algorithm 64

9 Concluding Remarks **66**

A Supplemental Algorithms **68**
A.1 Converting from the Power to the Bernstein Basis 68
A.2 Subdivision of a Multivariate Bernstein Polynomial 69

List of Figures

4-1 The control polyhedron of $(x, y, x^2 + y^2 - 1)$ 18

5-1 Effect of projecting the polyhedra of $(x, y, x^2 + y^2 - 1)$ and $(x, y, \frac{x^2}{4} + 4y^2 - 1)$ 26

6-1 A comparison between a PP and an LP bounding box 38

8-1 The curve $f(u, v) = 0$ of section 8.1.3 60

A-1 Pseudocode for basis conversion 69

A-2 Pseudocode for multivariate subdivision 70

List of Tables

8.1 Bounding boxes in the Projected-Polyhedron algorithm for a simple two variable case 56
8.2 Bounding boxes in the Linear Programming algorithm for a simple two variable case 57
8.3 Roots of the Wilkinson Polynomial 59
8.4 Critical points of an algebraic curve 61
8.5 Critical points of a distance function 62
8.6 Comparison of CPU times of PP, LP, and hybrid algorithms in test cases . 65

Chapter 1

Introduction

A fundamental problem in computer-aided design and manufacturing is the efficient computation of all solutions to a system of nonlinear polynomial equations within some finite domain. This problem arises in many different applications, as varied as they are numerous. For example, in computer-aided design it is often necessary to identify all characteristic points of an intersection curve between two surfaces, in order to trace out all of the branches of the curve; if the surfaces are piecewise rational polynomial, we must solve a system of nonlinear polynomial equations in order to identify these points. In the area of feature recognition, computing the medial axis transform requires the determination of branch points, which can frequently be formulated as the solution set of a polynomial system. This problem also arises in planning robotic motions, in computing distance functions and extrema, in arranging distributive systems in complex mechanical platforms, and in many other areas of computer-aided engineering.

In this thesis, we describe two techniques for eliciting all real solutions to a system of n nonlinear polynomial equations in n unknowns over an n-dimensional rectangular domain. We note that such an algorithm can also be used to find complex roots; simply separate each equation into a real and imaginary part, effectively doubling the number of equations in the system.

The thesis is structured as follows. Chapter 2 briefly reviews some of the prior research related to this topic. The notation we will use is introduced in chapter 3, and chapter 4 gives the formulation of the problem. Chapter 5 describes and analyzes the *Projected-Polyhedron* (PP) technique. In chapter 6, we introduce a new technique based on *Linear Programming*

(LP), and show that this method exhibits quadratic convergence irrespective of n. Chapter 7 considers the application of our root-finding schemes to optimization problems. Since, as we shall see, non-isolated roots often appear in such problems, this chapter also explores strategies for dealing with non-isolated roots. Some examples comparing the performance of the PP and LP approaches follow in chapter 8, along with a brief survey of some of the applications of this technique. A few concluding remarks are given in chapter 9, along with recommendations for future research.

An extended summary of the results of this thesis may be found in [42], with a more detailed version in [41]. A summary of this work dealing with computation of non-isolated roots and application of the algorithms to distance function computation may be found in [47]. Further applications are explored in [32].

Chapter 2

Literature Review

In recent CAD-related research, three classes of methods for the computation of solutions of nonlinear polynomial systems have been favored: algebraic techniques, homotopy, and subdivision. These methods may be classified as *global* because they are designed to compute all roots in some area of interest. There also exist a number of *local* numerical techniques which employ some variation of Newton-Raphson iteration or numerical optimization [7]. These methods are used in CAD applications requiring high accuracy because they are efficient (usually exhibiting quadratic convergence rates close to simple roots) and are straightforward to program. However, they typically require good initial approximations to roots; such approximations are usually obtained through some sort of global search like sampling, a process which cannot provide full assurance that all roots have been found. This lack of robustness makes the development of efficient and stable global techniques desirable; in what follows, we briefly review three classes of global methods.

The computation of all complex roots of nonlinear polynomial systems has typically been approached with *algebraic geometry techniques* like elimination or Groebner basis methods [4] [3]. These methods have many advantages; they are theoretically elegant, guaranteed to find all complex roots of a system irrespective of the dimensionality of the solution set, and well-suited for implementation in symbolic mathematical systems. However, they suffer from numerical instability, making implementation in floating point arithmetic difficult. Furthermore, they are inefficient in memory and processing time requirements, making them unattractive for systems only moderate in degree or dimensionality. There is also a philosophical problem with these techniques: they frequently give us much more information

than we need. As we shall see in chapter 8, many CAD applications only require those roots which lie in some real n-dimensional box.

The second category of methods is the class of *homotopy techniques* [46] [12]. These methods may be used to find all complex solutions of a nonlinear polynomial system if the number of roots is finite. Unfortunately, investigation of such methods indicates that they also tend to be numerically ill-conditioned. If we try to get around this problem by implementing the algorithm in exact rational arithmetic, we end up with enormous memory requirements because we have to solve large systems of complex initial value problems. And homotopy suffers from the same philosophical problem as algebraic geometry techniques — it simply provides more information than we need.

The third class, the category of *subdivision-based techniques*, is the class to which both the PP and LP methods described in this thesis belong. Lane and Riesenfeld [21] investigated the application of binary subdivision and the variation diminishing property of polynomials in the Bernstein basis to eliciting the real roots and extrema of a polynomial within an interval. Boehm [2] and Cohen et al. [5] extended this idea to general nonuniform subdivision of B-splines. Geisow [14] was among the first to develop subdivision algorithms to intersect planar algebraic curves expressed in the tensor-product and barycentric Bernstein bases and apply them to surface intersections. Formulation of a class of geometric problems in more than one dimension, requiring the solution of nonlinear piecewise polynomial systems expressed in terms of B-splines, was made by Dokken[8], who suggested a solution approach based on [5]. Subdivision techniques have also more recently been used in a wide variety of intersection problems for geometric modeling. Sederberg [39] developed an adaptive subdivision algorithm which can be used to intersect two planar algebraic curves expressed in the barycentric Bernstein basis. Patrikalakis, Prakash, and Kriezis [35] [33] [20] investigated the use of subdivision of algebraic curves in intersecting an implicit algebraic surface with a rational polynomial surface. Their method relies on the computation of real characteristic points of an algebraic curve represented in the Bernstein basis, which typically involves intersecting two or three algebraic curves by repeated adaptive subdivision and minimization. Minimization is used to increase the precision of the root quadratically; alternatively, Newton-Raphson iteration, which is also quadratically convergent, can be used to refine the root. Nishita et al. [28] developed an adaptive subdivision technique known as Bézier clipping to intersect rays with trimmed rational polynomial sur-

face patches, also recasting the problem as the intersection of two algebraic curves expressed in the Bernstein basis. Sederberg and Nishita [40] extended this method to intersect parametric curves with parametric surfaces. Vafiadou and Patrikalakis [44] employed an early two-dimensional form of the PP algorithm coupled with minimization to ray-trace offset surfaces. In this thesis, we will study the n-dimensional extension of the PP algorithm in detail, along with the related LP approach. It should be noted that subdivision techniques have a number of disadvantages. They are not as general as algebraic methods, since they are designed for zero-dimensional solution sets within an n-dimensional subset of Euclidean space. Furthermore, although the chances that all roots have been found increase as the resolution tolerance is lowered, there is no certainty that each root has been extracted. Lastly, subdivision techniques provide no explicit information about root multiplicities without additional computation. However, despite these drawbacks, their speed and stability make them attractive as root-finding schemes.

The subdivision methods reviewed above belong to the general class of *interval-based methods*, although they have a geometric character. Interval arithmetic techniques, primarily interval extension of Newton methods coupled with bisection to ensure convergence, have also been an active research topic in recent years [18] [27] [1]. Interval arithmetic techniques with a more geometric flavor may be found in [24] [22] [23] and in [17].

Chapter 3

Notation and Definitions

1. The lower-case letters i, j, k, l, m, and n denote nonnegative integers.

2. The lower-case letters f, g, and h denote real-valued functions.

3. Other lower-case letters, such as x and y, denote real numbers.

4. Lower-case letters in boldface, such as $\mathbf{x}$ and $\mathbf{y}$ will denote vectors of real numbers, or points. It should be clear from the context what the dimension of any given vector is. When we work with the components of the vector $\mathbf{x}$, we will assume that x_i is the ith component of $\mathbf{x}$.

5. Upper-case boldface letters such as $\mathbf{F}$ denote vector-valued functions. The ith component of $\mathbf{F}$ is denoted by $\mathbf{F}_i$.

6. Upper-case letters other than those between I and N inclusive (which are used for multi-indices, defined below) denote sets or matrices.

7. To avoid confusion, some variables may have superscripts instead of or in addition to subscripts. For example, the matrix $W^{(k)}$ is superscripted so that we can refer to the element in the ith row, jth column as $w_{ij}^{(k)}$, rather than as the ambiguous w_{ijk}.

8. $[a, b]$ denotes set of all real x such that $a \leq x \leq b$.

9. $\mathbf{R}$ denotes the real numbers; $\mathbf{R}^n$ denotes the set of points in n-dimensional Euclidean space.

Definition 3.1 *The* Euclidean norm $|\mathbf{x}|$ *of a vector* $\mathbf{x}$ *is defined to be the quantity* $\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$.

Definition 3.2 *A* multi-index I *is an ordered* n*-tuple of nonnegative integers* $(i_1, i_2, \ldots, i_n)$. *We say that* I *is* bounded *by the multi-index* $M = (m_1, m_2, \ldots, m_n)$ *if* $0 \leq i_1 \leq m_1, 0 \leq i_2 \leq m_2, \ldots, 0 \leq i_n \leq m_n$.

We will use multi-indices as a shorthand for subscripting multi-dimensional arrays. For example, if $I = (2, 3)$, then w_I is equivalent to w_{23}, the element of the matrix $\{w_{ij}\}$ at row 2, column 3.

We will use the notation $\sum_I^M$, where M is a bounding multi-index, to indicate that a sum is to be taken over all possible multi-indices I which are bounded by M. For example, if $M = (1, 1)$, then $\sum_I^M w_I \equiv w_{00} + w_{01} + w_{10} + w_{11}$.

Definition 3.3 *The* i*th* Bernstein polynomial *of degree* m *is defined by*

$$b_{i,m}(u) = \binom{m}{i} u^i (1-u)^{m-i} \tag{3.1}$$

The Bernstein multinomial *determined by the multi-index* $I = (i_1, i_2, \ldots, i_n)$ *and the bounding multi-index* $M = (m_1, m_2, \ldots, m_n)$ *is defined by*

$$B_{I,M}(\mathbf{u}) = b_{i_1,m_1}(u_1) b_{i_2,m_2}(u_2) \ldots b_{i_n,m_n}(u_n) \tag{3.2}$$

where $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in \mathbf{R}^n$.

Definition 3.4 *Let* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *be an element of* $\mathbf{R}^n$, *and let* $f : \mathbf{R}^n \to \mathbf{R}$ *be a function of* $\mathbf{x}$. *The* graph *of* f *is the function* $\mathbf{G}_f : \mathbf{R}^n \to \mathbf{R}^{n+1}$ *defined by*

$$\mathbf{G}_f(\mathbf{x}) = (\mathbf{x}, f(\mathbf{x})). \tag{3.3}$$

Definition 3.5 *Let* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ *be an element of* $\mathbf{R}^n$. *Then the* i*th* projection map $\pi_i : \mathbf{R}^n \to \mathbf{R}$ *is defined by*

$$\pi_i(\mathbf{x}) = x_i. \tag{3.4}$$

Definition 3.6 *Suppose* P *is a set of* m *points of* $\mathbf{R}^n$. *Let* $C(P)$ *denote the set of all* $\mathbf{x}$ *such that*

$$\mathbf{x} = \sum_{i=1}^{m} c_i \mathbf{p}_i \tag{3.5}$$

where for each i, $\mathbf{p}_i \in P$ and $c_i \geq 0$, and $\sum_{i=1}^{m} c_i = 1$. Then $C(P)$ is called the convex hull of P.

Definition 3.7 *Suppose $f_1, f_2, \ldots, f_n$ are functions of the parameters $x_1, x_2, \ldots, x_n$. Then the* Jacobian matrix *of the f_i, $\frac{\partial\{f_1, f_2, \ldots, f_n\}}{\partial\{x_1, x_2, \ldots, x_n\}}$, is a square, $n \times n$ matrix whose element at row i, column j, is equal to $\frac{\partial f_i}{\partial x_j}$.*

Definition 3.8 *Suppose $f(x)$ and $g(x)$ are both real-valued functions of one variable. We say that $f(x) = \mathcal{O}(g(x))$ if there exists a $\delta > 0$ and a constant $\lambda > 0$ such that for $|x| < \delta, |f(x)| \leq \lambda |g(x)|$.*

Chapter 4

Formulation

Suppose we have a set of n functions of n variables $f_1, f_2, \ldots, f_n$, each of which is polynomial in the independent parameters $u_1, u_2, \ldots, u_n$. Let $m_i^{(k)}$ denote the degree in u_i of polynomial f_k, so that the multi-index $M^{(k)} = (m_1^{(k)}, m_2^{(k)}, \ldots, m_n^{(k)})$ describes all the degree information of f_k. Furthermore, suppose there is an n-dimensional rectangular subset of $\mathbf{R}^n$

$$S = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]. \tag{4.1}$$

We wish to find all points $\mathbf{u} = (u_1, u_2, \ldots, u_n) \in S$ such that

$$f_1(\mathbf{u}) = f_2(\mathbf{u}) = \ldots = f_n(\mathbf{u}) = \mathbf{0}. \tag{4.2}$$

By making the *affine parameter transformation* [9] $u_i = a_i + x_i(b_i - a_i)$ for each i between 1 and n inclusive, we simplify the problem to one of determining all $\mathbf{x} \in [0, 1]^n$ such that

$$f_1(\mathbf{x}) = f_2(\mathbf{x}) = \ldots = f_n(\mathbf{x}) = \mathbf{0}. \tag{4.3}$$

Since all of the f_k are polynomial in each of the n independent parameters, a simple *change of basis* [9] allows us to express them in the multivariate Bernstein basis. In other words, for each f_k there exists an n-dimensional array of real coefficients $w_{i_1 i_2 \ldots i_n}^{(k)}$ such that for each $k \in \{1, \ldots, n\}$

$$f_k(\mathbf{x}) = \sum_{I}^{M^{(k)}} w_I^{(k)} B_{I, M^{(k)}}(\mathbf{x}). \tag{4.4}$$

Representation of algebraic and piecewise algebraic surfaces (i.e., for $n = 3$) in terms

of tensor products of Bernstein polynomials or B-splines has been studied earlier by Patrikalakis and Kriezis [31]; equation (4.4) is simply an extension to n dimensions. Provided that conversion of the problem to the Bernstein basis is exact or sufficiently accurate, the use of the Bernstein basis in conjunction with subdivision is known to be numerically stable [10]. The conversion process itself may be numerically ill-conditioned [11]; therefore, we recommend that the problem be formulated in the Bernstein basis from the very beginning. If necessary, polynomials may be converted from the multivariate power basis to the multivariate Bernstein basis using the algorithm described in the appendix; alternatively, the problem may be reformulated using a bilinear transformation described in [25].

The form of (4.3) is not sufficient to apply the two techniques we consider in this thesis. Before we can use either method, we have to restate the problem as the intersection of the *graphs* of the f_k (each of which is a hypersurface in $\mathbf{R}^{n+1}$) and the hyperplane $x_{n+1} = 0$ of $\mathbf{R}^{n+1}$. This idea is designed to impart geometrical significance to the coefficients of the polynomials and to the solution process. A similar approach has been used extensively as the foundation of convex-hull based subdivision algorithms for intersection problems (see chapter 2). Let us take the same approach in rewriting equation (4.3). Out of each f_k we will build its graph $\mathbf{F}_k$ (see definition 3.4):

$$\begin{aligned} \mathbf{F}_k(\mathbf{x}) &= (x_1, x_2, \ldots, x_n, f_k(\mathbf{x})) \\ &= (\mathbf{x}, f_k(\mathbf{x})). \end{aligned} \tag{4.5}$$

Clearly, (4.3) is satisfied by the point $\mathbf{x}$ if and only if

$$\mathbf{F}_1(\mathbf{x}) = \mathbf{F}_2(\mathbf{x}) = \ldots = \mathbf{F}_n(\mathbf{x}) = (\mathbf{x}, 0). \tag{4.6}$$

As with the f_k, the $\mathbf{F}_k$ may be expressed in the multivariate Bernstein basis. It can be shown that ([9])

$$\sum_{i=0}^{m} \frac{i}{m} b_{i,m}(u) = u; \tag{4.7}$$

in other words, the monomial u can be expressed as the weighted sum of Bernstein polynomials with coefficients evenly spaced in the interval $[0, 1]$. Using equation (4.7) and the summation to unity property of the Bernstein basis (5.2), we obtain an equivalent expression

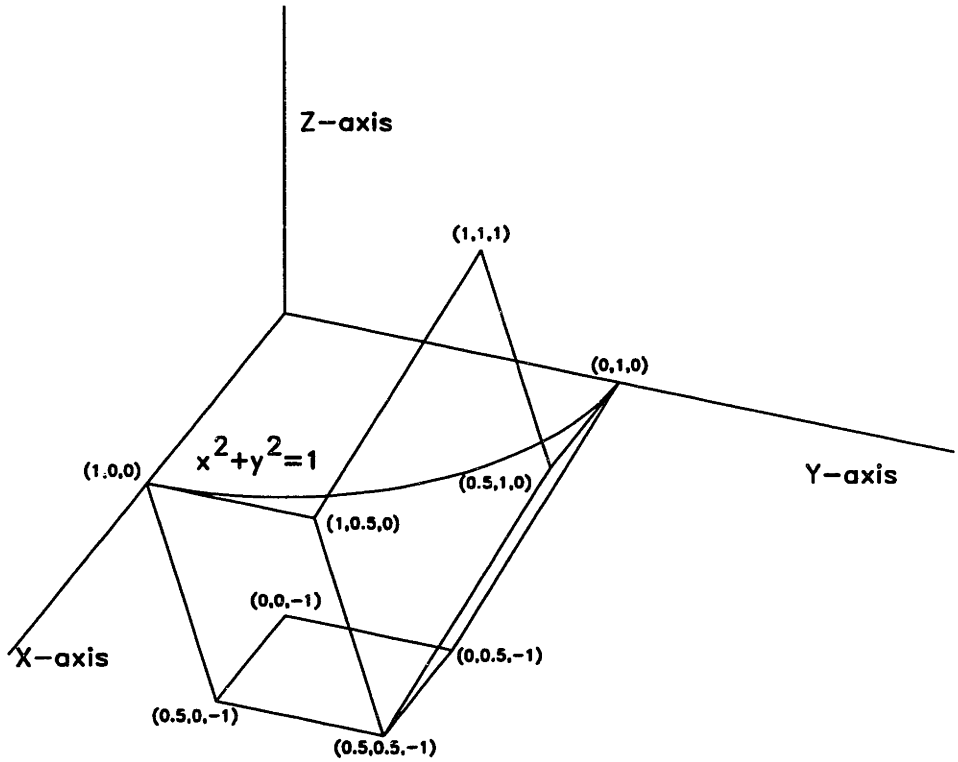

Figure 4-1: The control polyhedron of $(x, y, x^2 + y^2 - 1)$

for each of the x_j in equation (4.5):

$$x_j = \sum_{I}^{M^{(k)}} \frac{i_j}{m_j^{(k)}} B_{I,M^{(k)}}(\mathbf{x}). \tag{4.8}$$

Substituting (4.8) into (4.5) gives a more useful representation for the $\mathbf{F}_k$:

$$\mathbf{F}_k(\mathbf{x}) = \sum_{I}^{M^{(k)}} \mathbf{v}_I^{(k)} B_{I,M^{(k)}}(\mathbf{x}) \tag{4.9}$$

where

$$\mathbf{v}_I^{(k)} = (\frac{i_1}{m_1^{(k)}}, \frac{i_2}{m_2^{(k)}}, \ldots, \frac{i_n}{m_n^{(k)}}, w_I^{(k)}). \tag{4.10}$$

The $\mathbf{v}_I^{(k)}$ are called the *control points* of $\mathbf{F}_k$. In figure 4-1, we show the control points of the graph of $f(x, y) = x^2 + y^2 - 1$, whose zero set over $0 \leq x \leq 1, 0 \leq y \leq 1$ is a quarter circle in the plane. Although at first glance this seems like an unnecessarily complicated way of posing the problem, using the parametric hypersurfaces $\mathbf{F}_k$ instead of the real-valued f_k permits use of the powerful *convex-hull property* of the multivariate Bernstein basis, which

we will investigate below. As we shall see, the two techniques make use of this property in different ways, resulting in different convergence characteristics.

Chapter 5

The Projected-Polyhedron Intersection Algorithm

In this chapter, we introduce the first of our iterative root-finding algorithms for n-dimensional problems. Although occasionally slower than the second technique, it is easy to visualize and simple in that it only requires two straightforward algorithms in order to implement it: one for subdividing multivariate polynomials in Bernstein form, and one for finding the convex hull of a two-dimensional set of points. This algorithm is an extension of earlier adaptive subdivision algorithms for $n = 2$ used in intersecting two planar algebraic curves (reviewed in chapter 2).

5.1 Motivation for the Algorithm

Having rephrased the problem in the form of (4.6), we can now use properties of the Bernstein basis in order to eliminate parts of the domain $[0,1]^n$ which do not contain any roots. The two properties we will use are familiar ones which are immediate upon inspection of equation (3.1). The first is *positivity*:

$$b_{i,m}(u) \geq 0 \text{ for } u \in [0,1]. \tag{5.1}$$

The second is that the Bernstein polynomials form a *partition of unity* [9]:

$$\sum_{i=0}^{m} b_{i,m}(u) = 1. \tag{5.2}$$

Since the Bernstein multinomials are merely products of Bernstein polynomials, these two properties lead immediately to corresponding properties of Bernstein multinomials:

$$B_{I,M}(\mathbf{u}) \geq 0 \text{ for } \mathbf{u} \in [0,1]^n \tag{5.3}$$

and

$$\sum_{I}^{M^{(k)}} B_{I,M}(\mathbf{u}) = 1. \tag{5.4}$$

These two equations in turn lead to the important *convex hull property* of the multivariate Bernstein basis:

Theorem 5.1 *Let $V^{(k)}$ be the set of control points of $\mathbf{F}_k$ (that is, the $\mathbf{v}_I^{(k)}$ of (4.10)). Let $C(V^{(k)})$ be the* convex hull *of definition 3.6. Let $x \in [0,1]^n$. Then for each k between 1 and n inclusive,*

$$\mathbf{F}_k(\mathbf{x}) \in C(V^{(k)}). \tag{5.5}$$

Proof: Recall from (4.10) that $\mathbf{F}_k(\mathbf{x})$ is a linear combination of the control points $\mathbf{v}_I^{(k)}$, with coefficients being $B_{I,M^{(k)}}(\mathbf{x})$. By (5.3) and (5.4), these coefficients are positive and sum to 1, and therefore satisfy the requirements of the c_i in (3.5). Thus, $\mathbf{F}_k(\mathbf{x})$ is an element of the convex hull $C(V^{(k)})$.□

Because every point on the hypersurface $\mathbf{F}_k$ lies within the convex hull of its control points, equation (4.6) immediately implies the following theorem:

Theorem 5.2 *Let $\mathbf{x} \in [0,1]^n$ be a point satisfying (4.3), i.e. a root common to all of the f_k. Let X_{n+1} be the set of points of $\mathbf{R}^{n+1}$ which lie on the hyperplane $x_{n+1} = 0$. Then*

$$(\mathbf{x}, 0) \in \bigcap_{k=1}^{n} C(V^{(k)}) \cap X_{n+1} \tag{5.6}$$

Proof: We are given that $\mathbf{x}$ satisfies (4.3). Therefore, for each k between 1 and n, $\mathbf{x}$ satisfies $\mathbf{F}_k(\mathbf{x}) = (\mathbf{x}, 0)$. By theorem 5.1, the point $(\mathbf{x}, 0) \in C(V^{(k)})$, and since any point of the form $(\mathbf{x}, 0)$ is automatically an element of X_{n+1}, (5.6) follows.□

An simple but important corollary is worth stating:

Corollary 5.3 (Contrapositive) *Suppose $(\mathbf{x}, 0)$ does not satisfy (5.6). Then $\mathbf{x}$ is not a root common to all of the f_k.*

With the help of this corollary, we can devise the following general root-finding approach.

1. Let $B_0 = [0,1]^n$. This signifies that initially our region of search is the box $[0,1]^n$. As the algorithm proceeds, B_0 will keep track of which sub-box of $[0,1]^n$ we are considering.

2. Form the convex hulls of all the $\mathbf{F}_k$ and intersect them with one another and with the hyperplane $x_{n+1} = 0$. Call the intersection set A. If A is sufficiently small, stop.

3. Because A is the result of an intersection with the hyperplane $x_{n+1} = 0$, it can be expressed in the form $A = A' \times \{0\}$ for some set $A' \in [0,1]^n$. By corollary 5.3, any roots in $[0,1]^n$ must lie inside A'. We will call an element of A' a *feasible point.*

4. Find a box $B = [a_1, b_1] \times [a_2, b_2], \times \ldots \times [a_n, b_n] \in [0,1]^n$ which encloses A'. If this box B is not significantly smaller than $[0,1]^n$, split B into some number of equally sized, smaller boxes. Continue with the next step once for each one of these sub-boxes.

5. For each k, define a new function f'_k by

$$f'_k(\mathbf{x}) = f_k(a_1 + (b_1 - a_1)x_1, a_2 + (b_2 - a_2)x_2, \ldots, a_n + (b_n - a_n)x_n). \tag{5.7}$$

   This new function may be computed by applying De Casteljau subdivision according to the box B on f_k. (See the appendix for an algorithm which implements this procedure.) Notice that f'_k maps $[0,1]^n$ to the same set that f_k maps B to.

6. Update B_0. If $B_0 = [a_{0,1}, b_{0,1}] \times [a_{0,2}, b_{0,2}] \times \ldots \times [a_{0,n}, b_{0,n}]$, then the new $a_{0,k}$ and $b_{0,k}$ are given by $a_{0,k} \leftarrow a_{0,k} + (b_{0,k} - a_{0,k})a_k$ and $b_{0,k} \leftarrow a_{0,k} + (b_{0,k} - a_{0,k})b_k$, where k ranges from 1 to n.

7. Construct the control points of the graphs $\mathbf{F}_k$ from the f_k, and go back to step 2.

Although the above procedure is sound, we need to resolve two problems before implementation. First, forming the convex hull of a set of m points in n dimensions is generally an $\mathcal{O}(m^{\lfloor \frac{n}{2} \rfloor + 1})$ procedure [36]. And second, even when these convex hulls are generated, the time-consuming process of intersecting them remains, along with a large space requirement in which to store the result. Fortunately, it turns out that it is possible to obtain a good bounding box B *without actually executing step 2.* The two algorithms we introduce in

this thesis take full advantage of this fact, although they use different methods to generate B. Let us now examine the way that we generate the box B in the Projected-Polyhedron algorithm.

5.2 Generating the Bounding Box

We saw that the major difficulty of our computational procedure was generating and intersecting convex hulls from a set of points. The Projected-Polyhedron method generates a bounding-box by considering n two-dimensional problems on each iteration (one for each dimension of $[0,1]^n$) instead of one $n+1$-dimensional problem. The philosophy behind this approach is to generate each of the n sides of the box independently by *projecting* control points onto different planes.

Suppose we want to find $[a_j, b_j]$, the jth side of the bounding box B. Define a function $\boldsymbol{\pi}_j : \mathbf{R}^{n+1} \to \mathbf{R}^2$ by the equation

$$\boldsymbol{\pi}_j(\mathbf{x}) = (\pi_j(\mathbf{x}), \pi_{n+1}(\mathbf{x})) \tag{5.8}$$

where π_j and π_{n+1} are the projection maps of definition 3.5. Using $\boldsymbol{\pi}_j$, we will map the control points of each of the $\mathbf{F}_k$ into a Cartesian plane (which for the sake of simplicity we will call the (x,y) plane). We will then generate the (two-dimensional) convex hulls of these projected points and intersect them with one another and with the x-axis. Let us call the set resulting from this intersection B_j. Since B_j is a subset of the x-axis, it can be expressed as $B'_j \times \{0\}$ for some $B'_j \in [0,1]$. As the following theorem demonstrates, the set of feasible points A' is a subset of the box $B = B'_1 \times B'_2 \times \ldots \times B'_n$:

Theorem 5.4 *Let A' be the set of feasible points, as before. Let $\mathbf{v}_{j,I}^{(k)} = \boldsymbol{\pi}_j(\mathbf{v}_I^{(k)})$ for some specific j and k, and for each I bounded by $M^{(k)}$. Let $V_j^{(k)}$ be the set of these two-dimensional points $\mathbf{v}_{j,I}^{(k)}$, and let $C(V_j^{(k)})$ be its convex hull. Let X_2 be the x-axis (the set of points in the (x,y)-plane satisfying $y=0$). Finally, let*

$$B_j = \bigcap_{k=0}^{n} C(V_j^{(k)}) \cap X_2. \tag{5.9}$$

Express B_j in the form $B'_j \times \{0\}$. Then if $B = B'_1 \times B'_2 \times \ldots \times B'_n$, $A' \subset B$.

Proof: Let $\mathbf{x} \in A'$. Then for some choice of coefficients $\{c_I^{(k)}\}$ such that each $c_I^{(k)} \geq 0$ and $\sum_I^{M^{(k)}} c_I^{(k)} = 1$, we must have

$$(\mathbf{x}, 0) = \sum_I^{M^{(k)}} c_I^{(k)} \mathbf{v}_I^{(k)} \tag{5.10}$$

for all k between 1 and n. Taking $\boldsymbol{\pi}_j$ of both sides, we obtain

$$(x_j, 0) = \sum_I^{M^{(k)}} c_I^{(k)} \mathbf{v}_{j,I}^{(k)}. \tag{5.11}$$

Therefore $(x_j, 0) \in C(V_j^{(k)})$ for all k. Since also $(x_j, 0) \in X_2$, $(x_j, 0) \in B_j$ by (5.9), and thus $x_j \in B_j'$. Because this must hold for all j, we conclude that $\mathbf{x} \in B$. Since $\mathbf{x}$ was arbitrary, $A' \subset B$.□

Keep in mind that this inclusion is in general proper; that is, usually $A' \neq B$. In fact, A' tends to be *much* smaller than B in more than one dimension, resulting in a lower order of convergence than a tighter box around A' would achieve. Nevertheless, because generating the box is fairly simple, it is a reasonably good technique for most inputs.

Generation of the box is essentially performed according to the theorem. Suppose we are considering the jth direction. After we generate the $\mathbf{v}_{j,I}^{(k)}$ for each k, we construct their convex hulls with a simple algorithm related to Graham's scan [6]. We then intersect these convex polygons with the interval $[0, 1]$, which can be accomplished simply by marching along the vertices of each polygon and recording any intersections between their joining line segments and the x-axis as we go along. Note that since these polygons are convex, the intersection will always be either empty, a point, or a closed interval on the x-axis. This computation is also the essential element of the Bézier clipping method developed for $n = 2$ [28]. Once we have obtained these intervals (escaping from the procedure if any of them are empty) we simply intersect them. If the intersection B_j' is nonempty, it can therefore be expressed as an interval $[a_j, b_j]$. Performing this procedure for each j will give us a new B which is either empty or equal to $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$.

Figure 5-1 shows the effect of projection in the case of an intersecting circle and ellipse. Notice that the projection of their convex hulls cover the entire interval $[0, 1]$ on the x-axis, giving us no useful information; however, parts of the y-axis may be eliminated immediately, telling us that there is no root of the system whose y-coordinate is less than 0.1875 or greater

than 0.625 (see also section 8.1.1).

Now that we have a method for generating a bounding box, it is straightforward to write the algorithm PROJECTED-POLYHEDRON based on this method of box generation. The algorithm may be summarized in the following steps:

1. Start with an initial box of search

2. Perform the same transformation that we used in converting from equations (4.2) to (4.3), i.e. an affine parameter transformation that changes our box of search to $[0,1]^n$. The transformation of the n functions f_k is performed using multivariate De Casteljau subdivision (see the appendix).

3. Using the method described above, find a sub-box of $[0,1]^n$ which contains all the roots.

4. Using the reverse of the transformation we used in step 2, determine what this sub-box corresponds to in $\mathbf{R}^n$. If the actual box in $\mathbf{R}^n$ is sufficiently small, conclude that there is a root inside and return the midpoint of the box as an approximation to it.

5. If any dimensions of the sub-box in step 3 are not much smaller than 1 unit in length (i.e., the box has not decreased much in size along one or more sides), split the box at the midpoint of each dimension which is causing trouble. Continue on the next iteration with several independent sub-problems. (For example, suppose we have 4 equations in 4 unknowns, and thus a 4-dimensional box, and suppose also that two of the sides have not decreased considerably in length in step 3. Then, due to even splitting of the box at the problem areas, we will have $2^2 = 4$ boxes on the next iteration.)

6. Go back to step 2, once for each new box.

Step 2 is needed in order to use the convex hull property as described above. Step 5 is typically invoked when the box contains more than one root. The presence of more than one root in the same box obviously prevents the box from decreasing in size past a certain point, so splitting is necessary in order to isolate the roots from one another. On a typical iteration, however, step 5 will not be invoked, and step 6 will only send one box back to step 2 for processing.

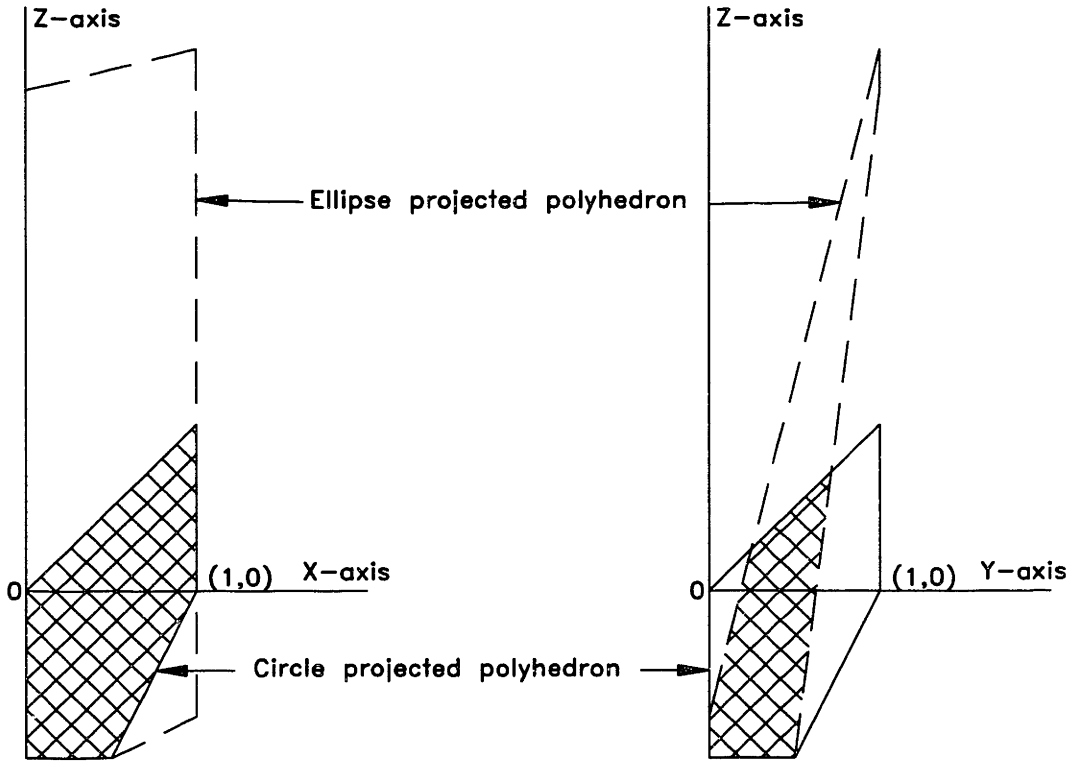

Figure 5-1: Effect of projecting the polyhedra of $(x, y, x^2 + y^2 - 1)$ and $(x, y, \frac{x^2}{4} + 4y^2 - 1)$

Obviously, some *tolerances* are needed. The tolerance in step 4 is simply whatever tolerance the root is required to satisfy. In step 5, we have to decide when a side is decreasing in size too slowly, and we use a threshold which we will call CRIT. Our experiments have suggested that we should require the sides to grow smaller by a factor of about CRIT$= 0.8$ on each iteration, although a slightly larger factor of CRIT$= 0.85$ or 0.9 is better for higher-dimensional problems since the boxes tend to decrease in size more slowly anyway. We will always assume that CRIT> 0.5.

5.3 Analysis

In this section, we will first analyze the working of the algorithm by introducing a *bounding box tree*. Next, we will briefly analyze the time complexity of the algorithm. Finally, by analyzing performance on a local scale, we will show that the algorithm achieves quadratic convergence for $n = 1$ and at best linear convergence for $n > 1$.

5.3.1 The Bounding-Box Tree

Let us first prove that the algorithm actually terminates. In order to do this, we first define a few terms relating to the algorithm:

Definition 5.5

1. *An* iteration *s is one execution of steps 2–5.*
2. *The* bounding box *B of iteration s is the actual box in $\mathbf{R}^n$ before we transform it to $[0,1]^n$ in step 2.*
3. *The* children *of a bounding box B are the boxes we are left with at step 6, and B is said to be the* parent *of these children. If we determine in step 4 that we are sufficiently close to a root, then B will have no children. Typically, B will only have one child; if splitting is necessary in step 5, then B will have a number of children equal to a power of 2.*

With the parent-child relationship defined among bounding boxes, we can construct a *bounding-box tree* where the nodes are bounding boxes and the links represent this relationship.

Lemma 5.6 *Let B_1 be a given node of the bounding-box tree. Suppose B_2 is one of its children. Then each side of B_2 is no larger than the corresponding side of B_1 times* CRIT.

Proof: By the definition of an iteration, between B_1 and B_2, each side of B_2 must either be smaller than the corresponding side of B_1 by a factor of CRIT or else splitting was performed, in which case the side of B_2 is no greater than half as long as the side of B_1. Since CRIT> 0.5, we are done.□

Corollary 5.7 *Suppose we are given some number $\varepsilon > 0$. Then there exists a number N such that after N or fewer steps, the algorithm has terminated.*

Proof: Since CRIT < 1, there exists some integer $p > 0$ such that CRIT$^p < \varepsilon$. Now each node has at most 2^n children for an n-dimensional problem. Furthermore, the depth of the bounding-box tree is no greater than p since any box with sides less than ε has no children, and by lemma 5.6 every time we proceed down a level of the tree, every side of a child's

box is at least CRIT times as small as its parent. The number of nodes in this tree must therefore be bounded above by 2^{np}. Choosing $N = 2^{np}$ completes the proof.□

By itself, Corollary 5.7 is not particularly reassuring. We would like all of the boxes of the first type (that is, boxes under tolerance) to contain one and only one root, but the corollary gives no assurance of this by itself. However, as we shall see, for small n the local convergence properties of the algorithm minimize this problem; in most cases, for reasonably small ε, all of the boxes under tolerance will contain one and only one root. Of course, the actual number of iterations needed to complete execution will usually be far smaller than 2^{np}; this value is merely an upper bound useful for showing that the algorithm terminates.

5.3.2 Complexity Analysis

It is difficult *a priori* to predict how many iterations will actually be needed to complete the algorithm. However, we can analyze the amount of time required to execute each *iteration* of the algorithm individually (see definition 5.5). In the analysis, we will assume that each equation is degree m in each variable, or of total degree mn; although it is easy to extend the analysis to the general case that all the degrees are different, the resulting formulas are quite long.

The generation of a convex hull of $(m+1)^n$ two-dimensional points takes at most $\mathcal{O}((m+1)^n \log (m+1)^n)$ time [6]. However, by exploiting the special structure of these points, we can reduce this cost. There are several points with identical abscissae; however, at most two of them can belong to the boundary of the convex hull (specifically, the points with the minimum and maximum ordinates). Therefore, we can discard all other points from consideration, leaving us with $2(m+1)$ points (a minimum and maximum point for each abscissa). These points may now be traversed in counterclockwise order as in Graham's scan [6]. The entire execution time of this procedure is $\mathcal{O}((m+1)^n)$. The time needed to traverse this convex hull in order to intersect it with the x-axis is no greater, so we conclude that the total execution time to perform the projection in one direction is $\mathcal{O}(m^n)$. Because there are a total of n convex hulls which need to be intersected in a given projection plane, the total cost of a projection becomes $\mathcal{O}(nm^n)$. Now, there are n different projection planes, one for each side of the box, so the total time needed to execute step 3 is $\mathcal{O}(n^2 m^n)$.

For a moment, assume that no splitting is necessary. A glance at the pseudocode in

the appendix will show that to subdivide one of the f_k in all n directions takes $\mathcal{O}(nm^{n+1})$ time; since there are n f_k which are subdivided, the total time needed to perform the transformation is $\mathcal{O}(n^2m^{n+1})$. Now, we don't know *a priori* how many children will result from a step of the algorithm, so the time will vary from step to step. But since we are really interested in an *average* cost per iteration, we can get a useful estimate by assessing the cost of subdivision on the children. With this useful device, we get an average cost per iteration of $\mathcal{O}(n^2m^n + n^2m^{n+1}) = \mathcal{O}(n^2m^{n+1})$. Although at first it may seem unpleasant that the algorithm is exponential in n, keep in mind that the input itself (the coefficients of the polynomials) is itself exponential in n, so merely printing out the control points would take exponential time. Furthermore, for most geometric modeling applications, it is quite rare to see an n higher than 6. It is often more helpful to consider that for a given number of dimensions, the algorithm is polynomial in m. This analysis is incomplete because the number of iterations needed may be quite different for systems with the same n and m. However, the local convergence properties can give us a better idea of how much time is required. In order to analyze these local convergence properties, we will first examine the relationship between the level of subdivision of a multivariate polynomial and the closeness of its control points.

5.3.3 Establishing Bounds

Theorem 5.8 *Let* $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ *and* $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. *Let* $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n) = (b_1 - a_1, b_2 - a_2, \ldots, b_n - a_n)$. *Suppose* $f : [0,1]^n \to \mathbf{R}$ *is the function which results from subdividing another function* $f_0 : [0,1]^n \to \mathbf{R}$ *over the box* $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$; *that is,*

$$f(x_1, x_2, \ldots, x_n) = f_0(a_1 + x_1\varepsilon_1, a_2 + x_2\varepsilon_2, \ldots, a_n + x_n\varepsilon_n). \tag{5.12}$$

Suppose further that for a given bounding multi-index $M = (m_1, m_2, \ldots, m_n)$, $f(\mathbf{x}) = \sum_I^M p_I B_{I,M}(\mathbf{x})$. *Then there exists a positive constant* κ *independent of* $\boldsymbol{\varepsilon}$ *such that for each* I *bounded by* M, *there exists a point* $\mathbf{x} \in [0,1]^n$ *for which* $|f(\mathbf{x}) - p_I| < \kappa|\boldsymbol{\varepsilon}|^2$.

Informally, this theorem simply says that if the sides of our bounding box are on the order of $|\boldsymbol{\varepsilon}|$, then the distance from f to any of its control points is on the order of $|\boldsymbol{\varepsilon}|^2$. In order to prove theorem 5.8, we will break the problem up into several steps. First, we take a Taylor expansion of f around the point $\mathbf{x} = \mathbf{0}$ in order to generate a linear approximation

$g : [0,1]^n \to \mathbf{R}$:

$$g(\mathbf{x}) = f(\mathbf{0}) + \sum_{j=1}^{n} x_j \frac{\partial f}{\partial x_j}(\mathbf{0}) \tag{5.13}$$

Since g is linear in each coordinate direction, it can be written as the sum of Bernstein multinomials of degree 1 in each variable. Suppose we now apply *degree elevation* [9] in each of the n directions, so that coordinate direction j has degree m_j instead of 1. Then we can rewrite g as

$$g(\mathbf{x}) = \sum_{I}^{M} q_I B_{I,M}(\mathbf{x}) \tag{5.14}$$

An important property of g is that because g is linear, all of its control points q_I lie on the image set of g [9]. Another important characteristic of g is that because it is a first-order approximation to f, the difference $h \equiv f - g$ is a second-order quantity. That is, there exists some positive constant κ_0 independent of ε such that for all $\mathbf{x} \in [0,1]^n$,

$$|h(\mathbf{x})| \leq \kappa_0 |\varepsilon|^2 \tag{5.15}$$

Since $h = f - g$, we can also write

$$h(\mathbf{x}) = \sum_{I}^{M} \delta_I B_{I,M}(\mathbf{x}) \tag{5.16}$$

where $\delta_I \equiv p_I - q_I$. We would like now to establish bounds on the δ_I, which we do by first proving bounds on the derivatives of h evaluated at $\mathbf{0}$:

Lemma 5.9 *Let $I = (i_1, i_2, \ldots, i_n)$, and let the notation $\frac{\partial^I}{\partial \mathbf{x}^I}$ be equivalent to $\frac{\partial^{i_1+i_2+\ldots+i_n}}{\partial x_1^{i_1} \partial x_2^{i_2} \ldots \partial x_n^{i_n}}$. Let h be as above. Then there exists a positive constant κ_1 independent of ε such that $|\frac{\partial^I h}{\partial \mathbf{x}^I}(\mathbf{0})| \leq \kappa_1 |\varepsilon|^2$ for all I bounded by M.*

Proof: There are three cases we need to consider:

Case I: If $I = (0, 0, \ldots, 0)$, then $\frac{\partial^I h}{\partial \mathbf{x}^I} = h$. Since $g(\mathbf{0}) = f(\mathbf{0})$, $h(\mathbf{0}) = 0 < \kappa_1 |\varepsilon|^2$ for any $\kappa_1 > 0$.

Case II: If $\frac{\partial^I h}{\partial \mathbf{x}^I} = \frac{\partial h}{\partial x_j}$ for some j (that is, $i_j = 1$ for some j and the other $n-1$ elements of I are 0), then $\frac{\partial^I h}{\partial \mathbf{x}^I}(\mathbf{0}) = 0 < \kappa_1 |\varepsilon|^2$ because $\frac{\partial f}{\partial x_j}(\mathbf{0}) = \frac{\partial g}{\partial x_j}(\mathbf{0})$.

Case III: For any I bounded by M which does not satisfy either case above, $\frac{\partial^I h}{\partial \mathbf{x}^I} = \frac{\partial^I f}{\partial \mathbf{x}^I}$ because derivatives of g of order greater than 1 are identically 0. Now recall that f is

a function which results from subdividing another function f_0 over the box $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$. Therefore, we can make use of equation (5.12) and the chain rule to obtain

$$\frac{\partial^I f}{\partial \mathbf{x}^I}(\mathbf{0}) = \frac{\partial^I f_0}{\partial \mathbf{x}^I}(\mathbf{a}) \prod_{j=1}^{n} \varepsilon_j^{i_j}. \tag{5.17}$$

Because I does not satisfy either of the first two cases, we must have $i_1 + i_2 + \ldots + i_n \geq 2$. Therefore,

$$\prod_{j=1}^{n} \varepsilon_j^{i_j} \leq |\boldsymbol{\varepsilon}|^2 \tag{5.18}$$

Furthermore, since f_0 is polynomial, all of its derivatives are bounded over $[0,1]^n$. Thus there exists some positive constant κ_1 such that for all $\mathbf{x} \in [0,1]^n$ and all I bounded by M,

$$|\frac{\partial^I f_0}{\partial \mathbf{x}^I}(\mathbf{x})| \leq \kappa_1. \tag{5.19}$$

Substituting (5.19) and (5.18) into (5.17) finally gives

$$|\frac{\partial^I h}{\partial \mathbf{x}^I}(\mathbf{0})| \leq \kappa_1 |\boldsymbol{\varepsilon}|^2. \tag{5.20}$$

Since these three cases cover all permissible I, we are done.□

The next step is to show that each of the δ_I can be expressed as a linear combination of derivatives of h evaluated at $\mathbf{0}$. In order to do this, we recall the following result from [10]:

$$x_j^k = \sum_{i=k}^{n} \frac{\binom{i}{k}}{\binom{n}{k}} b_{i,n}(x_j). \tag{5.21}$$

Now h can be as easily expressed in the power multinomial basis as the Bernstein multinomial basis; that is, there exist coefficients d_I such that

$$\sum_{I}^{M} \delta_I B_{I,M}(\mathbf{x}) = \sum_{I}^{M} d_I \prod_{j=1}^{n} x_j^{i_j}. \tag{5.22}$$

Substituting (5.21) for each x_j, we obtain

$$h(\mathbf{x}) = \sum_{I}^{M} d_I \prod_{j=1}^{n} [\sum_{i=i_j}^{m_j} \frac{\binom{i}{i_j}}{\binom{m_j}{i_j}} b_{i,m_j}(x_j)]. \tag{5.23}$$

If we define $\binom{n}{k} \equiv 0$ for $n < k$, (5.23) becomes

$$h(\mathbf{x}) = \sum_{I}^{M} d_I \prod_{j=1}^{n} [\sum_{i=0}^{m_j} \frac{\binom{i}{i_j}}{\binom{m_j}{i_j}} b_{i,m_j}(x_j)]. \tag{5.24}$$

Now we can easily equate like terms to obtain the following relationship between δ_I and d_I:

$$\delta_I = \sum_{J}^{M} \frac{\binom{j_1}{i_1}\binom{j_2}{i_2}\cdots\binom{j_n}{i_n}}{\binom{m_1}{i_1}\binom{m_2}{i_2}\cdots\binom{m_n}{i_n}} d_J. \tag{5.25}$$

This equation may not appear extremely relevant at first sight, but it becomes more useful when we recall the simple relationship between the d_J and the derivatives of h at $\mathbf{0}$:

$$d_J = \frac{1}{j_1! j_2! \ldots j_n!} \frac{\partial^J h}{\partial \mathbf{x}^J}(\mathbf{0}) \tag{5.26}$$

This relationship can be readily verified by differentiating the right-hand side of (5.22), but it is a familiar result of multivariable calculus. By plugging (5.26) into (5.25) we obtain

$$\delta_I = \sum_{J}^{M} \frac{\binom{j_1}{i_1}\binom{j_2}{i_2}\cdots\binom{j_n}{i_n}}{\binom{m_1}{i_1}\binom{m_2}{i_2}\cdots\binom{m_n}{i_n}} \frac{1}{j_1! j_2! \ldots j_n!} \frac{\partial^J h}{\partial \mathbf{x}^J}(\mathbf{0}). \tag{5.27}$$

With this relationship established, we can prove the following lemma:

Lemma 5.10 *Let δ_I be as before. Then there exists some positive constant κ_2 independent of ϵ such that for all I bounded by M, $|\delta_I| \leq \kappa_2 |\epsilon|^2$.*

Proof: Consider the coefficient of $\frac{\partial^J h}{\partial \mathbf{x}^J}(\mathbf{0})$ in equation (5.27). Since J is bounded by M, $m_k \geq j_k$ for $k \in \{1, 2, \ldots, n\}$, and so $\binom{m_k}{i_k} \geq \binom{j_k}{i_k}$. Therefore, each coefficient is a positive number less than or equal to 1, giving

$$|\delta_I| \leq \sum_{J}^{M} |\frac{\partial^J h}{\partial \mathbf{x}^J}(\mathbf{0})|. \tag{5.28}$$

Using the result of lemma 5.9, we obtain

$$|\delta_I| \leq m_1 m_2 \ldots m_n \kappa_1 |\epsilon|^2 \tag{5.29}$$

Putting $\kappa_2 \equiv m_1 m_2 \ldots m_n \kappa_1$ completes the proof.□

Now we are ready to put all the pieces together and prove theorem 5.8.

Proof of Theorem: Pick a multi-index I bounded by M. Now, q_I lies on the image set of g, i.e. there exists a point $\mathbf{x} \in [0,1]^n$ such that

$$g(\mathbf{x}) = q_I \tag{5.30}$$

From lemma 5.10 we know that

$$|p_I - q_I| = |g(\mathbf{x}) - p_I| \leq \kappa_2 |\varepsilon|^2 \tag{5.31}$$

But equation (5.15) gives us a bound on the quantity $|f - g|$. Applying the triangle inequality, we get

$$|f(\mathbf{x}) - p_I| \leq |f(\mathbf{x}) - g(\mathbf{x})| + |g(\mathbf{x}) - p_I| \leq (\kappa_0 + \kappa_2)|\varepsilon|^2. \tag{5.32}$$

Setting $\kappa \equiv \kappa_0 + \kappa_2$ completes the proof.□

5.3.4 Quadratic Convergence in One Dimension

Theorem 5.11 *Suppose we are given an interval $[a,b]$ which contains one and only one root x_0 of a polynomial $f_0 : [0,1] \to \mathbf{R}$. Further assume that x_0 is a simple root. Let $\varepsilon \equiv b - a$, and let f denote f_0 subdivided over $[a,b]$, i.e. $f(x) = f_0(a + x\varepsilon)$. Let ε' denote the size of the new interval which results after executing steps 2–4 of* PROJECTED-POLYHEDRON *with f, a, and b as the governing parameters. Then there exists a positive constant λ independent of ε and a $\delta > 0$ such that if $\varepsilon \leq \delta$, then $\varepsilon' \leq \lambda\varepsilon^2$.*

Proof: Now x_0 is a simple root of f_0, so $f_0'(x_0) \neq 0$. By continuity, there exists a δ such that $f_0'(x) \neq 0$ for $x \in [x_0 - \delta, x_0 + \delta]$. By assumption, $\varepsilon \leq \delta$, so $[a,b] \subset [x_0 - \delta, x_0 + \delta]$.

Let $f(x) = \sum_{i=0}^{m} p_i b_{i,m}(x)$. Let g be the linear approximation to f at 0, i.e. $g(x) = f(0) + xf'(0) = \sum_{i=0}^{m} q_i b_{i,m}(x)$. As before, let us construct the graphs of f and g:

$$\mathbf{F}(x) = (x, f(x)), \quad \mathbf{G}(x) = (x, g(x)) \tag{5.33}$$

From lemma 5.10, we know that for each i, $|q_i - p_i| \leq \kappa_2\varepsilon^2$. Therefore, for each i, the control points $(\frac{i}{m}, p_i)$ of $\mathbf{F}$ and $(\frac{i}{m}, q_i)$ of $\mathbf{G}$ are separated by a vertical distance less than or equal to $\kappa_2\varepsilon^2$. Since $\mathbf{G}$ is a line, all of its control points lie on the line; thus, each control point of $\mathbf{F}$ must lie on or between the following two lines which are respectively above and

below **G**:

$$\mathbf{L}_1(t) = (0, f(0) + \kappa_2\varepsilon^2) + (t, tf'(0)) \tag{5.34}$$

$$\mathbf{L}_2(t) = (0, f(0) - \kappa_2\varepsilon^2) + (t, tf'(0)) \tag{5.35}$$

The region enclosed by these two lines and the vertical lines $x = 0$ and $x = 1$ is convex; therefore, the convex hull of the control points of **F** is entirely contained within this region.

The intersection of L_1 with the horizontal axis is at $t = -\frac{f(0)+\kappa_2\varepsilon^2}{f'(0)}$, and the intersection of L_2 with the horizontal axis is at $t = -\frac{f(0)-\kappa_2\varepsilon^2}{f'(0)}$. The size of the intersection interval is therefore

$$\Delta t = |\frac{2\kappa_2\varepsilon^2}{f'(0)}| = |\frac{2\kappa_2\varepsilon}{f_0'(a)}| \tag{5.36}$$

where we have used the fact that $f'(0) = \varepsilon f_0'(a)$. Since the convex hull of the control points of **F** is completely contained in the region bounded by L_1 and L_2, the size of the interval computed in step 3 of PROJECTED-POLYHEDRON must be less than or equal to Δt. Remember, however, that this interval must be rescaled in step 4; after scaling, the size ε' of the new interval is given by

$$\varepsilon' \leq |\frac{2\kappa_2\varepsilon^2}{f_0'(a)}| \tag{5.37}$$

Let c equal the minimum value of $|f_0'|$ over the interval $[x_0 - \delta, x_0 + \delta]$, and let $\lambda \equiv \frac{2\kappa_2}{c}$. Then (5.37) implies that

$$\varepsilon' \leq \lambda\varepsilon^2. \tag{5.38}$$

Note that since κ_2 and c are independent of ε, so is λ, and therefore the algorithm exhibits quadratic convergence in one dimension.□

5.3.5 Convergence in Higher Dimensions

Unfortunately, the quadratic convergence of one dimension does not carry over into higher dimensions. Instead, we observe at best linear convergence which, as we shall see, is due to the use of projections. Projection of control points simplifies the problem, but we pay for the privilege with a lower order of convergence.

Theorem 5.12 *Suppose we are given a box* $B = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$ *which contains one and only one root* $\mathbf{x_0}$ *of a system of polynomials* $f_{0,k} : [0,1]^n \to \mathbf{R}$ *where*

k ranges from 1 to n. Assume further that $\mathbf{x}_0$ is a simple root. Suppose f_k represents the subdivision of $f_{0,k}$ over B. Execute steps 2–4 of PROJECTED-POLYHEDRON, *and let $B' = [a'_1, b'_1] \times [a'_2, b'_2] \times \ldots \times [a'_n, b'_n]$ be the box that results in step 4. Let j be an integer between 1 and n, and let $\varepsilon' \equiv b'_j - a'_j$, the length of the jth side of the new box. Then ε' is an $\mathcal{O}(\varepsilon)$ quantity.*

Proof: For the moment, fix k. Let $f_k(\mathbf{x}) = \sum_I^{M^{(k)}} p_I^{(k)} B_{I,M}(\mathbf{x})$, and let $g_k(\mathbf{x}) = \sum_I^{M^{(k)}} q_I^{(k)} B_{I,M}(\mathbf{x})$ be the degree-elevated linear approximation to f_k. Let $\mathbf{F}_k(\mathbf{x}) = (\mathbf{x}, f_k(\mathbf{x}))$ and $\mathbf{G}_k(\mathbf{x}) = (\mathbf{x}, g_k(\mathbf{x}))$ be the graphs of f_k and g_k. Let the control points of $\mathbf{F}_k$ be $\mathbf{p}_I^{(k)}$ and the control points of $\mathbf{G}_k$ be $\mathbf{q}_I^{(k)}$.

Apply the projection function $\boldsymbol{\pi}_j = (\pi_j, \pi_{n+1})$ to the $\mathbf{p}_I^{(k)}$ and the $\mathbf{q}_I^{(k)}$ in order to generate sets of two-dimensional points $\mathbf{p}_{j,I}^{(k)}$ and $\mathbf{q}_{j,I}^{(k)}$:

$$\mathbf{p}_{j,I}^{(k)} = (\pi_j(\mathbf{p}_I^{(k)}), \pi_{n+1}(\mathbf{p}_I^{(k)})) = (\frac{j}{m_j^{(k)}}, p_I^{(k)}) \tag{5.39}$$

$$\mathbf{q}_{j,I}^{(k)} = (\frac{j}{m_j^{(k)}}, q_I^{(k)}). \tag{5.40}$$

Now, by lemma 5.10, we know that $|p_I^{(k)} - q_I^{(k)}| \leq \kappa_2 \varepsilon^2$. Therefore, the points $\mathbf{p}_{j,I}^{(k)}$ and $\mathbf{q}_{j,I}^{(k)}$ are separated by a vertical distance of not more than $\kappa_2 \varepsilon^2$.

Unfortunately, the projection of $\mathbf{G}_k$ into the (x_j, x_{n+1})-plane is not a straight line if $n > 1$; in fact, it is generally a planar region with a nonzero area. For a fixed x_j, the difference between the maximum y-coordinate and the minimum y-coordinate of this region is given by

$$\Delta y = \sum_{i=1, i \neq j}^{n} |\frac{\partial f_k}{\partial x_i}(\mathbf{0})|. \tag{5.41}$$

This equation is obtained by taking the minimum value of (5.13) over all $\mathbf{x} \in [0,1]^n$ with x_j fixed and subtracting it from the maximum value of the same quantity. Now, since this region is the projection of a hyperplanar region into two dimensions, it must be a parallelogram, bounded above and below by two parallel lines separated by Δy and on the sides by the vertical lines $x = 0$ and $x = 1$. Notice that if $n = 1$, Δy is identically 0.

Using this value of Δy, we can proceed as we did in the previous section and find two lines $\mathbf{L}_1$ and $\mathbf{L}_2$ which have slope equal to $\frac{\partial f}{\partial x_j}$ and which are separated by a distance $\Delta y + 2\kappa_2 \varepsilon^2$. Together with the vertical lines $x = 0$ and $x = 1$, these lines form a region

enclosing the control points of $\mathbf{F}_k$; since this region is convex, it also encloses the convex hull of the control points.

Following the same steps as the proof of theorem 5.11, we can show that the intersection between this region and the x-axis is an interval with width

$$\Delta t = |\frac{\Delta y + 2\kappa_2 \varepsilon^2}{\frac{\partial f}{\partial x_j}(\mathbf{0})}|. \tag{5.42}$$

But the numerator and denominator of Δt are both $\mathcal{O}(\varepsilon)$ quantities if Δy and $\frac{\partial f}{\partial x_j}(\mathbf{0})$ are both nonzero, and so Δt is an $\mathcal{O}(1)$ quantity. Now, we still have to intersect this interval with the other $n-1$ intervals obtained using different values of k; after intersection, the final interval may be considerably smaller than Δt. But if, on the other hand, for each k the intervals are about the same size, their intersection may well be about as large or the same size as Δt. Therefore, this intersection is an $\mathcal{O}(1)$ quantity. After scaling it step 4, we finally obtain $\varepsilon' = \mathcal{O}(\varepsilon)$.□

It is possible to construct special cases for $n > 1$ where one can observe quadratic or close to quadratic convergence. However, most multivariate systems of polynomial equations demonstrate linear convergence in accordance with this theorem; typically the constant hidden inside the $\mathcal{O}(\varepsilon)$ expression increases with degree and number of variables, as one might expect. Intuitively, a higher number of variables makes the projected area "thicker" and a higher degree typically makes the intersecting hypersurfaces less like hyperplanes; both of these factors will contribute to a larger constant. In the preceding analysis, we saw that the extra "thickness" of the intersecting convex hulls due to projection reduces the order of convergence from quadratic to linear. Since the intersection region decreases in width quadratically, we expect there to be a tighter box around the region which gives rise to a quadratically convergent algorithm. As we shall see in the next chapter, we can use linear programming to determine such a box: this technique gives rise to a quadratically convergent algorithm regardless of dimension, which performs exceptionally well for $n = 2$.

Chapter 6

The Linear Programming Root Isolation Algorithm

In this chapter, we present and analyze the second of our root-finding algorithms for systems of multivariate polynomials. Since most of the derivation of this algorithm is given in chapter 5, we will concentrate mostly on those aspects of the Linear Programming (LP) algorithm which are different from the Projected-Polyhedron technique. The principal difference between the two methods is the way that they generate bounding boxes for the set of feasible points A' (see figure 6-1). Therefore, we begin by examining the way this box is generated in the LP approach. In the following discussion we will be working with both of the two sets A and A'. Recall that $A = A' \times \{0\}$; in other words, for every element of A', there is an element of A with the first n coordinates identical and an $n+1$st coordinate equal to 0. Technically speaking, the linear programming subroutine will be searching for minima and maxima of a linear function over the set A, but these extrema will be used to bound A'. Distinguishing between the two sets A and A' is unimportant, since they are isomorphic.

6.1 Generating the Bounding Box

We start by splitting the problem of determining $B = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_n, b_n]$ into the $2n$ independent subproblems of computing each a_i and each b_i. If we are attempting to compute a_i, the question we ask is "if we look at the ith coordinates of all points lying in A, what is the value of the smallest?" If we are looking for b_i, the end of that statement

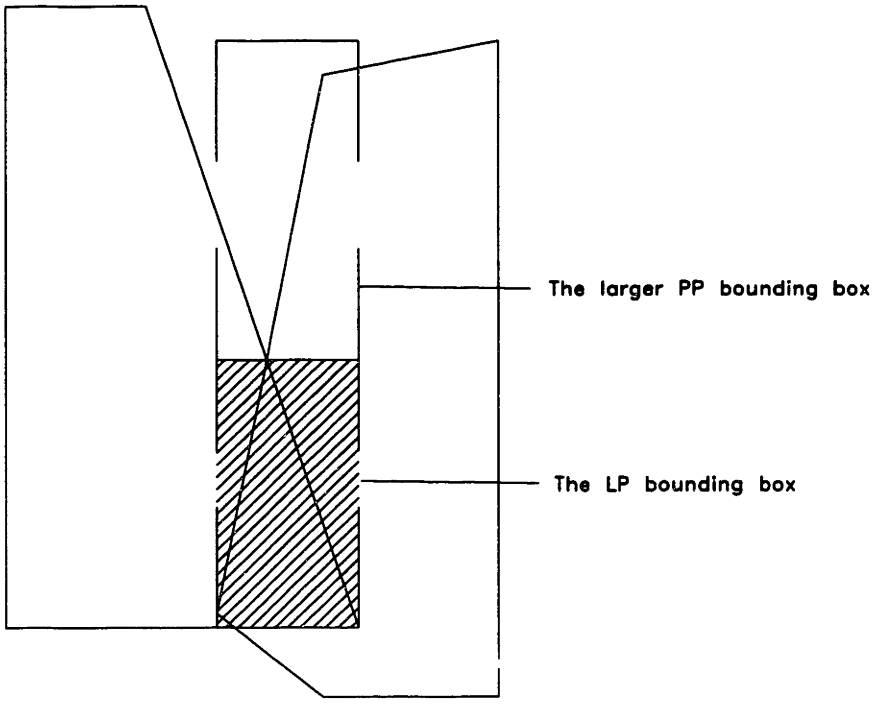

Figure 6-1: A comparison between a PP and an LP bounding box

would read "what is the value of the biggest?" More formally,

$$a_i = \min_{\mathbf{x} \in A}\{x_i\}, \quad b_i = \max_{\mathbf{x} \in A}\{x_i\} \tag{6.1}$$

where i ranges from 1 to n. The proof that A' lies within the box determined by the a_i and the b_i is trivial, so we omit any proof of this fact.

Now, as (6.1) currently stands, it would appear that we need to have the set A to work with. However, we have already stated that the intersection set A is difficult to compute explicitly. The important thing to realize is that we do not actually have to determine A in order to evaluate (6.1); instead we write down a set of *constraint equations* which uniquely determine A. Then we run an optimization algorithm to determine the minimum and maximum values of x_i subject to these constraints. Because all of the constraints turn out to be linear, the optimization algorithm we use is a linear programming algorithm.

It turns out that the constraint equations are quite easy to write down. In order for a point $\mathbf{x} \in \mathbf{R}^{n+1}$ to be in A, the following two conditions must hold:

1. For each k between 1 and n, $\mathbf{x} \in C(V^{(k)})$.

2. $x_{n+1} = 0$.

By the definition of A, these conditions are also sufficient; that is, if the conditions hold for a given point $\mathbf{x}$, then $\mathbf{x} \in A$.

In order to use linear programming, we need to rewrite the constraint equations using definition 3.6. Recall that $\mathbf{x} \in C(V^{(k)})$ if and only if there exist $c_I^{(k)}$ such that

$$\mathbf{x} = \sum_I^{M^{(k)}} c_I^{(k)} \mathbf{v}_I^{(k)} \tag{6.2}$$

where the $c_I^{(k)}$ are nonnegative and sum to 1, $\mathbf{v}_I^{(k)} \in V^{(k)}$, and $\mathbf{v}_I^{(k)} \neq \mathbf{v}_J^{(k)}$ if $I \neq J$.

Using equation (4.10) to substitute for the $\mathbf{v}_I^{(k)}$, we can rewrite the two conditions above in a more useful form:

Theorem 6.1 *A point $\mathbf{x} \in \mathbf{R}^{n+1}$ is in A if and only if there exist nonnegative coefficients $c_I^{(k)}$ such that*

1. *for each j between 1 and n and for each k between 1 and n such that $m_j^{(k)} \neq 0$,*

$$\sum_I^{M^{(k)}} c_I^{(k)} \frac{i_j}{m_j^{(k)}} = x_j, \tag{6.3}$$

2. *for each k between 1 and n,*

$$\sum_I^{M^{(k)}} c_I^{(k)} w_I^{(k)} = 0, \tag{6.4}$$

3. *and for each k between 1 and n,*

$$\sum_I^{M^{(k)}} c_I^{(k)} = 1. \tag{6.5}$$

Suppose that we wish to solve (6.1) — that is, find the minimum value of x_i over all $\mathbf{x} \in A$ for some specific i. Then, since (6.3) must hold for $i = j$ and for all k, minimizing x_i is equivalent to minimizing

$$x_i = \sum_I^{M^{(1)}} c_I^{(1)} \frac{i_i}{m_i^{(1)}} \tag{6.6}$$

subject to (6.3), (6.4), and (6.5).

Now, linear programming problems are usually posed as follows: given a $p \times q$ matrix C of constraint function coefficients, a q-vector $\mathbf{u}$ of nonnegative unknowns, a p-vector $\mathbf{r}$ of right-hand sides, and a q-vector $\mathbf{c}$ of objective function coefficients, minimize $\mathbf{c}^T\mathbf{u}$ over all possible $\mathbf{u}$ which satisfy $C\mathbf{u} = \mathbf{r}$ [13]. With a small amount of algebra, we can rewrite our minimization problem in this form. Note that if we want to find the *maximum* of $\mathbf{c}^T\mathbf{u}$ instead, we can simply negate all of the elements of $\mathbf{c}$ and obtain a minimization problem.

The vector $\mathbf{u}$ of unknowns will simply be all of the $c_I^{(k)}$. Therefore, q, the number of elements of $\mathbf{u}$, will be given by

$$q = \sum_{k=1}^{n} (m_1^{(k)} + 1)(m_2^{(k)} + 1) \dots (m_n^{(k)} + 1) \tag{6.7}$$

The $2n$ constraints obtained by writing (6.4) and (6.5) for each k may be directly inserted into C. However, in order to put (6.3) into C for all j and k which satisfy $m_j^{(k)} \neq 0$, we must eliminate the x_j from the equations. When we have done this, the first of the three conditions of theorem 6.1 may be restated as follows:

Corollary 6.2 *For all j between 1 and n and all k between 1 and $n-1$ such that $m_j^{(k)} \neq 0$, if $\mathbf{x}$ is in A and if there exists a k' which is the smallest integer such that $k < k' \leq n$ and $m_j^{(k)} \neq 0$, then*

$$\sum_{I}^{M^{(k)}} c_I^{(k)} \frac{i_j}{m_j^{(k)}} - \sum_{I}^{M^{(k')}} c_I^{M^{(k')}} \frac{i_j}{m_j^{(k')}} = 0. \tag{6.8}$$

Notice that if $m_j^{(k)} \neq 0$ for all j and k, then $k' = k + 1$.

We insert these equations (there are no more than $n^2 - n$) into C directly, resulting in a total number of constraint equations $p \leq n^2 + n$.

Now we can build an algorithm LP-FIND which looks much like PROJECTED-POLYHEDRON based on this box generation scheme. In fact, the only difference between the two occurs in step 3, since the boxes are generated differently. Since the other steps are the same, we will not write the steps of LP-FIND explicitly; instead, we will refer to the steps of the algorithm as we did in chapter 5.

6.2 Analysis

In this section, we will analyze the performance of the algorithm, just as we did with PROJECTED-POLYHEDRON. We will use some of the theorems in chapter 5 to show that LP-Find exhibits quadratic convergence. Notice that we can define a bounding-box tree for LP-FIND just as we did for PROJECTED-POLYHEDRON and use it to prove that the algorithm terminates. We also define a notion of an *iteration* of the algorithm just as we did in chapter 5.

6.2.1 Complexity Analysis

As before, we will assume that each polynomial is degree m in each variable, resulting in a total degree of mn for each of the n polynomials. Again, we do not know *a priori* how many iterations will be required, but we can at least assess the cost of one iteration of LP-FIND.

The cost of linear programming depends on the method used; so let us assume we are using the simplex method. This method operates on the $p \times q$ matrix C to generate a solution usually in $\mathcal{O}(p^2q)$ time [13]. Assume that p is its largest possible value, n^2+n; p will attain this value if all variables appear in all equations, and this is therefore a conservative estimate. As we shall see in chapter 8, there exist many examples in geometric modeling which involve sparse systems of equations and therefore result in a much smaller p. Since $q = (m+1)^n$, the total execution time of one call to the linear programming solver is $\mathcal{O}(n^4m^n)$. Note that a total of $2n$ calls to the solver are necessary — one maximum and one minimum value for each of n dimensions. The total cost of this number of calls is therefore $\mathcal{O}(n^5m^n)$. (Note that there exist methods for simultaneously optimizing a number of objective functions on the same constraint matrix which are more efficient than naively calling one routine n times [30], but we will not consider such improvements in this thesis.)

As before, we assess the cost of subdivision on the children of the current box, so the current box only has to pay for the subdivision which creates it. As we saw before, this subdivision costs $\mathcal{O}(n^2m^{n+1})$. Thus the total amortized cost for an iteration of LP-FIND is $\mathcal{O}(n^5m^n+n^2m^{n+1})$. As we shall see in the next section, LP-FIND exhibits quadratic convergence; thus, usually LP-FIND needs fewer steps to finish than PROJECTED-POLYHEDRON. The smaller number of steps often mitigates the extra n^3 factor we notice in comparing first terms of the LP and PP methods for small n. Furthermore, the LP method constructs

tighter bounding boxes and is therefore likely to involve many fewer splitting steps than the PP method.

6.2.2 Quadratic Convergence in n Dimensions

In this section, we will prove that close to an isolated root, the LP algorithm exhibits quadratic convergence regardless of the dimension of the problem, obviously an improvement over the PP technique. As before, the proof involves linearization of the intersecting functions.

Theorem 6.3 *Suppose we are given a box* $B = [a_1, b_1] \times [a_2, b_2] \times [a_n, b_n]$ *which contains one and only one root* $\mathbf{x_0}$ *of a system of polynomials* $f_{0,k} : [0,1]^n \to \mathbf{R}$ *where* k *ranges from* 1 *to* n. *Let* f_k *represent the subdivision of* $f_{0,k}$ *over* B. *Execute steps 2–4 of* LP-FIND *once with these parameters, and let the new box at step 4 be denoted by* $B' = [a'_1, b'_1] \times [a'_2, b'_2] \times [a'_n, b'_n]$. *Define* $\boldsymbol{\varepsilon} \equiv \mathbf{b} - \mathbf{a}$ *and* $\boldsymbol{\varepsilon}' \equiv \mathbf{b}' - \mathbf{a}'$. *There exists a neighborhood* U *of* $\mathbf{x_0}$ *and a* $\lambda > 0$ *depending only on* U *such that if* $B \subset U$ *and* $\det \frac{\partial\{f_1, f_2, \ldots, f_n\}}{\partial\{x_1, x_2, \ldots, x_n\}} \neq 0$, *then* $\max_{k=1}^{n}\{\varepsilon'_k\} \leq \lambda \max_{k=1}^{n}\{\varepsilon_k^2\}$.

Proof:

Step 1: In the interest of clarity, we will define ε to be $\max_{k=1}^{n}\{\varepsilon_k\}$. Let us start by linearizing all n of the f_k according to (5.13):

$$g_k(\mathbf{x}) = f_k(\mathbf{0}) + \sum_{j=1}^{n} x_j \frac{\partial f_k}{\partial x_j}(\mathbf{0}) \tag{6.9}$$

Suppose we determine the solution point in $\mathbf{R}^{n+1}$ to $g_1 = g_2 = \ldots = g_n = x_{n+1} = 0$. Now, if the determinant of the matrix $\frac{\partial\{f_1, f_2, \ldots, f_n\}}{\partial\{x_1, x_2, \ldots, x_n\}}$ is nonzero, then this system will have exactly one solution by Cramer's rule (the solution may not be in $[0,1]^n$, but we will not worry about that at the moment). Let us call this intersection point $\mathbf{p} = (p_1, p_2, \ldots, p_n, 0)$.

Step 2: Pick an arbitrary point $\mathbf{q} = (q_1, q_2, \ldots, q_n, 0)$ in the intersection set A defined previously. (Recall that A is the intersection of $\mathbf{F_1}, \mathbf{F_2}, \ldots, \mathbf{F}_n$ with the hyperplane $x_{n+1} = 0$.) Now, from (5.15) we know that

$$f_k(\mathbf{x}) - g_k(\mathbf{x}) = \mathcal{O}(\varepsilon^2) \tag{6.10}$$

for all $\mathbf{x} \in [0,1]^n$. Substituting $(q_1, q_2, \ldots, q_n)$ for $\mathbf{x}$, we see that since all f_k are zero at this point, g_k must be $\mathcal{O}(\varepsilon^2)$. Or equivalently, letting $r_k \equiv g_k(q_1, q_2, \ldots, q_n)$, there exist n

points $\mathbf{r}_k = (q_1, q_2, \ldots, q_n, r_k)$ such that

1. $\mathbf{r}_k$ lies on the linear approximation to $\mathbf{F}_k$ and

2. r_k is an $\mathcal{O}(\varepsilon^2)$ quantity.

Step 3: Because $\mathbf{r}_k$ lies on the linear approximation to $\mathbf{F}_k$, it must be true that for each k,

$$r_k = f_k(\mathbf{0}) + \sum_{j=1}^{n} q_j \frac{\partial f_k}{\partial x_j}(\mathbf{0}). \tag{6.11}$$

Of course, since $\mathbf{p}$ is the intersection point of all of these linear approximations, it must also be true that for each k,

$$0 = f_k(\mathbf{0}) + \sum_{j=1}^{n} p_j \frac{\partial f_k}{\partial x_j}(\mathbf{0}). \tag{6.12}$$

Subtracting (6.12) from (6.11) for each k and defining $\delta_k \equiv q_k - p_k$, we obtain the system of k equations

$$\sum_{j=1}^{n} \delta_j \frac{\partial f_k}{\partial x_j}(\mathbf{0}) = r_k. \tag{6.13}$$

Assuming, again, that the Jacobian matrix $J = \frac{\partial\{f_1, f_2, \ldots, f_n\}}{\partial\{x_1, x_2, \ldots, x_n\}}$ is nonsingular, Cramer's rule enables us to find each δ_k matrix easily. According to the rule, if we let J_k be the matrix obtained by exchanging the kth column of J with the column vector $(r_1, r_2, \ldots, r_n)^T$, then

$$\delta_k = \frac{\det J_k}{\det J} \tag{6.14}$$

Recall (as in the proof of theorem 5.11) that the chain rule gives $\frac{\partial f_k}{\partial x_j} = \mathcal{O}(\varepsilon)$, and so each column of J has an element which is $\mathcal{O}(\varepsilon)$. But since every column of J is an $\mathcal{O}(\varepsilon)$ quantity, $\det J$ is $\mathcal{O}(\varepsilon^n)$. J_k has $n-1$ columns which are $\mathcal{O}(\varepsilon)$ and one column $(\mathbf{r}_k)$ which is $\mathcal{O}(\varepsilon^2)$, and so $\det J_k$ is $\mathcal{O}(\varepsilon^{n+1})$. Thus, δ_k is $\mathcal{O}(\varepsilon)$.

Step 4: Since δ_k is $\mathcal{O}(\varepsilon)$ and $\mathbf{q}$ was chosen as an arbitrary element of A, we conclude that for each k, the kth coordinate of any point in A never deviates from a fixed value (i.e., p_k) by more than an $\mathcal{O}(\varepsilon)$ quantity. Therefore, by the definition of the LP bounding box, the length of each side of the box is an $\mathcal{O}(\varepsilon)$ quantity. But remember that the previous discussion has assumed that the input box is $[0,1]^n$ when in fact it is B. Scaling is therefore performed in step 4; since the scaling factor is $b_k - a_k = \mathcal{O}(\varepsilon)$, we conclude that the resulting

interval size

$$b'_k - a'_k = \mathcal{O}(\varepsilon)\mathcal{O}(\varepsilon) = \mathcal{O}(\varepsilon^2). \tag{6.15}$$

By applying the definition of the $\mathcal{O}$ notation, we conclude that there is some set U containing B (and thus B' and $\mathbf{x_0}$) such that the constant $\lambda > 0$ hidden within the $\mathcal{O}$ notation depends only on U. Translating (6.15) then gives $\varepsilon'_k = b'_k - a'_k \leq \lambda\varepsilon^2$, which in turn leads immediately to $\max_{k=1}^{n}\{\varepsilon'_k\} \leq \lambda \max_{k=1}^{n}\{\varepsilon_k^2\}$.□

Chapter 7

Optimization Problems

The problem of determining maxima and minima of polynomial functions is one that appears often in Computer Aided Design. Perhaps the most common instance is the determination of minimum or maximum distance between geometric entities described by polynomial functions. (Such entities include points, curves, and surfaces.) The formulation of such problems is treated in [47], and we include an example in chapter 8 as well. In this chapter, however, we will focus on strategies for classifying stationary points of polynomial functions as maxima, minima, or saddle points.

The computation of stationary points of distance functions gives rise to another problem: that of non-isolated sets of roots. Although such degeneracies may occur in many classes of root-finding problems, they are especially prevalent in distance computation. As an example, consider two surfaces $\mathbf{F}(u_1, v_1)$ and $\mathbf{G}(u_2, v_2)$ and the function

$$D(u_1, v_1, u_2, v_2) = |\mathbf{F}(u_1, v_1) - \mathbf{G}(u_2, v_2)|^2. \tag{7.1}$$

This polynomial function of four variables describes the squared distance between all pairs of points selected from $\mathbf{F}$ and $\mathbf{G}$ respectively. The local minima of this function will indicate the points on $\mathbf{F}$ and $\mathbf{G}$ where the surfaces are locally closest to one another. But suppose the two surfaces actually intersect in a curve. Then *every* point on that curve represents a local minimum of D, and none of them are isolated. As one can imagine, this situation poses a significant problem for our algorithms. Both the PP and LP techniques are designed to find a finite number of roots, and therefore fall short when operating on a system with an infinite number of roots. If we feed such a system into our techniques, we notice that although areas

not containing roots are eliminated, a tremendous number of small boxes around the solution set remain. The computer slows down as computation becomes excessive and memory is exhausted. Some alternative method is clearly needed here, which we will consider later in section 7.2. At the moment, however, we turn our attention to the classification of stationary points.

7.1 Classification of Isolated Stationary Points

Let us first recall the definitions of local extrema at stationary points:

Definition 7.1 *Suppose that* $D : \mathbf{R}^n \to \mathbf{R}$ *is a scalar field on* $\mathbf{R}^n$. *Let* $\mathbf{x}$ *be a* stationary point *of* D, *that is* $\nabla D(\mathbf{x}) = \mathbf{0}$. *Then*

1. $\mathbf{x}$ *is a* local maximum *if there exists a neighborhood* U *of* $\mathbf{x}$ *such that for all* $\mathbf{y} \in U$, $D(\mathbf{y}) \leq D(\mathbf{x})$.

2. $\mathbf{x}$ *is a* local minimum *if there exists a neighborhood* U *of* $\mathbf{x}$ *such that for all* $\mathbf{y} \in U$, $D(\mathbf{y}) \geq D(\mathbf{x})$.

To illustrate the application of these definitions, let us begin with a simple one-parameter function $D(u)$ (which may arise, for example, in the point-curve distance problem). Suppose u_0 is a stationary point of $D(u)$, i.e. $D'(u_0) = 0$. Then a Taylor series expansion tells us that

$$\begin{aligned} D(u_0 + h) - D(u_0) &= hD'(u_0) + h^2 D''(u_0 + ht) \\ &= h^2 D''(u_0 + ht) \end{aligned} \tag{7.2}$$

where $t \in [0, 1]$ is some specific value depending on h. From definition 7.1, we see that u_0 is a local maximum if the right hand side of (7.2) is never positive for h close to 0 and a local minimum if the right hand side is nonnegative for h close to 0. Now, if $D''(u_0) \neq 0$, then by continuity, there is some δ such that for $|h| < \delta$, $D''(u_0)$ has the same sign as $D''(u_0 + ht)$. Therefore, if $D''(u_0) < 0$, then $h^2 D''(u_0 + ht) < 0$ for $|h| < \delta$ and hence u_0 is a local maximum. Similarly, if $D''(u_0) > 0$, then u_0 is a local minimum. These results are familiar from calculus.

Unfortunately, the problem becomes more complicated if $D''(u_0) = 0$. In this case we can't say anything about the sign of $D''(u_0 + ht)$ and must therefore expand the Taylor series to higher derivatives. For example, if the second derivative is zero but the third derivative is nonzero, then we will have neither a maximum nor a minimum but a *point of inflection.* Consider the function $y = x^3$; in any neighborhood of the stationary point $x = 0$, the function takes on both positive and negative values and thus $x = 0$ is neither a maximum nor a minimum. If the third derivative is also zero, we have to look at the fourth derivative, and so on. Fortunately, these exceptions are rare, but a robust point classification algorithm should be able to handle such cases.

Similar analysis can be performed with functions of more than one variable. Consider a function $D(u, v)$ of two variables, which might arise, for example, in a curve-curve distance problem. A Taylor expansion around the stationary point (u_0, v_0) to second order leads to

$$\begin{aligned} D(u_0 + h, v_0 + k) - D(u_0, v_0) &= \frac{h^2}{2} D_{uu}(u_0 + ht, v_0 + kt) + hk D_{uv}(u_0 + ht, v_0 + kt) \\ &\quad + \frac{k^2}{2} D_{vv}(u_0 + ht, v_0 + kt) \end{aligned} \tag{7.3}$$

where $t \in [0, 1]$. Instead of one second order term, this time we have three to deal with, which of course makes classification more difficult. However, by simply *completing the square* with the second order terms, we can easily formulate conditions for minima, maxima, and saddle points. To see this, let $a \equiv \frac{D_{uu}}{2}, b \equiv \frac{D_{uv}}{2}$, and $c \equiv \frac{D_{vv}}{2}$. Then we have

$$\begin{aligned} D(u_0 + h, v_0 + k) - D(u_0, v_0) &= ah^2 + 2bhk + ck^2 \\ &= a(h^2 + \frac{2b}{a}hk + \frac{b^2}{a^2}k^2) + (c - \frac{b^2}{a})k^2 \\ &= a(h + \frac{b}{a}k)^2 + (c - \frac{b^2}{a})k^2 \end{aligned} \tag{7.4}$$

Now h and k appear only within squared expressions; this new form enables us to ignore the signs of h and k and instead concentrate on the signs of the coefficients a and $(c - \frac{b^2}{a})$. To see this, notice that within any arbitrarily small neighborhood of (u_0, v_0), the ratio $\frac{h}{k}$ takes on all possible values between $-\infty$ and ∞, and therefore the first term of the right hand side of (7.4) can become extremely small or extremely large compared to the second term. However, the signs of each term are not so easily changed; these are determined solely by the signs of a and $(c - \frac{b^2}{a})$. If both coefficients are nonzero at (u_0, v_0), a continuity argument

shows that the expressions $D_{uu}(u_0+ht, v_0+kt)$ and $D_{vv}(u_0+ht, v_0+kt) - \frac{D_{uv}^2(u_0+ht,v_0+kt)}{D_{uu}(u_0+ht,v_0+kt)}$ do not change sign as long as (u_0+ht, v_0+kt) is sufficiently close to (u_0, v_0). Therefore, in order to determine the signs of these coefficients within a small neighborhood of (u_0, v_0), we can simply evaluate them at (u_0, v_0); if the coefficients are nonzero there, then their signs will enable us to classify the stationary point. For example, if both coefficients are positive at the stationary point, then clearly the right hand side of (7.4) is positive within some neighborhood of the stationary point, and hence (u_0, v_0) can be classified as a minimum. If the first coefficient is negative and the second is positive, then because either term may be made zero at different parts of any neighborhood, the right hand side of (7.4) may be positive or negative arbitrarily close to the stationary point, and thus (u_0, v_0) is neither a maximum nor a minimum. The other possibilities lead to the following theorem, which is proven more fully in [16]:

Theorem 7.2

1. *If $D_{uu} < 0$ and $D_{uv}^2 < D_{uu}D_{vv}$ at the stationary point (u_0, v_0), then (u_0, v_0) is a local maximum.*
2. *If $D_{uu} > 0$ and $D_{uv}^2 < D_{uu}D_{vv}$, then (u_0, v_0) is a local minimum.*
3. *If $D_{uv}^2 > D_{uu}D_{vv}$ then (u_0, v_0) is a saddle point (neither a maximum nor a minimum).*
4. *If none of the above conditions apply, then it is necessary to examine higher-order derivatives.*

Notice that the third condition above applies even if $D_{uu} = 0$. In this case, the second order term reduces to $2bhk + ck^2$. As long as $b \neq 0$, specific choices of h and k within any arbitrarily small neighborhood of the stationary point can make this term either positive or negative, and hence (u_0, v_0) is neither a maximum nor a minimum.

It is possible to complete the square for functions of more than two variables; however, in order to develop general conditions for minima and maxima of such functions, we will employ the powerful theory of *quadratic forms.*

Definition 7.3 *A* quadratic form *is an expression of the form $\mathbf{x}^{\mathrm{T}}A\mathbf{x}$, where $\mathbf{x} = [x_1\ x_2\ \ldots\ x_n]^{\mathrm{T}}$ is an n-dimensional column vector of unknowns and A is an $n \times n$ matrix. A quadratic form $\mathbf{x}^{\mathrm{T}}A\mathbf{x}$ is said to be*

1. positive definite *if* $\mathbf{x}^{\mathrm{T}}A\mathbf{x} > 0$ *for all* $\mathbf{x} \neq 0$.

2. positive semidefinite *if* $\mathbf{x}^{\mathrm{T}}A\mathbf{x} \geq 0$ *for all* $\mathbf{x}$.

3. negative definite (semidefinite) *if* $-\mathbf{x}^{\mathrm{T}}A\mathbf{x}$ *is positive definite (semidefinite).*

4. indefinite *otherwise (i.e. the form takes on both positive and negative values).*

To see how these quadratic forms are used, let us take a Taylor expansion of a function of n variables $D(\mathbf{u})$ around the stationary point $\mathbf{u} = \mathbf{u_0}$:

$$D(\mathbf{u_0} + \mathbf{h}) - D(\mathbf{u_0}) = \frac{1}{2}\mathbf{h}^{\mathrm{T}}H\mathbf{h} + \mathcal{O}(|\mathbf{h}|^3). \tag{7.5}$$

Here H is the *Hessian matrix* of D; the element of the ith row, jth column, is given by

$$H_{ij} = \frac{\partial^2 D}{\partial u_i \partial u_j}(\mathbf{u_0}). \tag{7.6}$$

Now clearly, if the quadratic form $\mathbf{h}^{\mathrm{T}}H\mathbf{h}$ is positive definite, then within some neighborhood of the stationary point $\mathbf{u_0}$, the right hand side of (7.5) is nonnegative, and therefore $\mathbf{u_0}$ is a local minimum. Similarly, if the quadratic form is negative definite, then $\mathbf{u_0}$ is a local maximum.

At this point, we can use a familiar theorem of linear algebra whose proof is given in [43]:

Theorem 7.4 *Let* $\mathbf{x}^{\mathrm{T}}A\mathbf{x}$ *be a quadratic form, let* A *be a real symmetric matrix, and for* $i = 1, 2, \ldots, n$, *let* d_i *be the determinant of the upper left* $i \times i$ *submatrix of* A *(which we will denote* A_i*). Then the quadratic form is positive definite if and only if* $d_i > 0$ *for all* i.

If the $>$ condition in the above theorem is relaxed to $\geq$, a corresponding theorem applies to positive semidefinite forms (that is, a form is positive semidefinite if and only if $d_i \geq 0$ for all i).

Theorem 7.4 immediately gives us a necessary and sufficient condition for negative definiteness as well. For, if we let $B \equiv -A$, the quadratic form $\mathbf{x}^{\mathrm{T}}A\mathbf{x}$ is negative definite if and only if $\mathbf{x}^{\mathrm{T}}B\mathbf{x}$ is positive definite if and only if $\det B_i > 0$ for all i. But $\det B_i = (-1)^i \det A_i$ and so we have the following corollary:

Corollary 7.5 *Let A and d_i be as before. Then the quadratic form $\mathbf{x}^{\mathrm{T}}A\mathbf{x}$ is negative definite if and only if $(-1)^i d_i > 0$ for all i.*

Simply applying these conditions gives us the following theorem:

Theorem 7.6 *Let D be as before, and let H be its Hessian matrix, evaluated at the stationary point $\mathbf{u}_0$. Let d_i be the determinant of the upper left $i \times i$ submatrix H_i of H.*

1. *If $d_i > 0$ for all i, then $\mathbf{u}_0$ is a local minimum.*
2. *If $(-1)^i d_i > 0$ for all i, then $\mathbf{u}_0$ is a local maximum.*

If, on the other hand, the form $\mathbf{h}^{\mathrm{T}}H\mathbf{h}$ is *indefinite*, we can conclude that $\mathbf{u}_0$ is neither a minimum nor a maximum (in two dimensions, such points are usually called *saddle points*), as the following theorem shows:

Theorem 7.7 *Let $H, D, \mathbf{u}_0$, and d_i be as before. Suppose that the d_i do not satisfy the conditions for definiteness or semidefiniteness. Then $\mathbf{u}_0$ is neither a local minimum or local maximum.*

Proof: Because $\mathbf{h}^{\mathrm{T}}H\mathbf{h}$ is indefinite, there exist vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ such that $\mathbf{h}_1^{\mathrm{T}}H\mathbf{h}_1 < 0$ and $\mathbf{h}_2^{\mathrm{T}}H\mathbf{h}_2 > 0$. Pick an arbitrary neighborhood U of $\mathbf{u}_0$. Then for any $\varepsilon > 0$ smaller than some positive number δ, $\mathbf{u}_0 + \varepsilon\mathbf{h}_1 \in U, \mathbf{u}_0 + \varepsilon\mathbf{h}_2 \in U, (\varepsilon\mathbf{h}_1)^{\mathrm{T}}H(\varepsilon\mathbf{h}_1) < 0$, and $(\varepsilon\mathbf{h}_2)^{\mathrm{T}}H(\varepsilon\mathbf{h}_2) > 0$.

Using (7.5), we obtain

$$D(\mathbf{u}_0 + \varepsilon\mathbf{h}_1) - D(\mathbf{u}_0) = \frac{1}{2}\varepsilon^2\mathbf{h}_1^{\mathrm{T}}H\mathbf{h}_1 + \mathcal{O}(\varepsilon^3|\mathbf{h}_1|^3) \tag{7.7}$$

$$D(\mathbf{u}_0 + \varepsilon\mathbf{h}_2) - D(\mathbf{u}_0) = \frac{1}{2}\varepsilon^2\mathbf{h}_2^{\mathrm{T}}H\mathbf{h}_2 + \mathcal{O}(\varepsilon^3|\mathbf{h}_2|^3) \tag{7.8}$$

Clearly, for sufficiently small ε, the right hand side of (7.7) will remain negative, while the right hand side of (7.8) will remain positive. Thus we have, for sufficiently small ε,

$$D(\mathbf{u}_0 + \varepsilon\mathbf{h}_1) < D(\mathbf{u}_0) < D(\mathbf{u}_0 + \varepsilon\mathbf{h}_2). \tag{7.9}$$

Since U was an arbitrary neighborhood of $\mathbf{u}_0$, there is no neighborhood of $\mathbf{u}_0$ where $D(\mathbf{u}_0) \geq D(\mathbf{u})$ for all $u \in U$ or where $D(\mathbf{u}_0) \leq D(\mathbf{u})$ for all $u \in U$. Therefore $\mathbf{u}_0$ is neither a local minimum nor a local maximum.□

Unfortunately, trouble can occur if the Hessian is only semidefinite and not definite. In this case we need to look at higher order derivatives or use some other method to classify the point precisely. Although this is not too difficult for one-parameter functions, it becomes progressively more involved as the number of variables increase. This subject is beyond the scope of this thesis, but plenty of literature is available on such problems (for example, [34] [15]).

7.2 Identification of Non-isolated Stationary Points

Because the equation $\nabla D = \mathbf{0}$ is equivalent to a system of n equations in n unknowns, we expect that in most cases the solution set of this system will consists of a few discrete, isolated points in $\mathbf{R}^n$. However, it is possible for the solution set to contain curves, surfaces, or hypersurfaces as well as points. For example, suppose $D(x, y) = x^2y^2$; then the solution set of $\nabla D = \mathbf{0}$ consists of the two lines $x = 0$ and $y = 0$.

In this section, we will examine a marching method which shows promise in tracing curves of critical points. This type of degeneracy can occur if, for example, we are trying to find the minimum of the squared distance between two surfaces which happen to intersect. The method we use to trace out such curves involves setting up a system of differential equations which can be solved numerically using a standard ordinary differential equation solver [29].

We begin our discussion by defining the concept of a *degenerate critical point* [34]:

Definition 7.8 *A critical point* $\mathbf{u}_0$ *of a function* $D : \mathbf{R}^n \to \mathbf{R}$ *(that is, a point at which* $\nabla D = \mathbf{0}$*) is called* degenerate *if the Hessian matrix of* D *evaluated at* $\mathbf{u}_0$ *is singular.*

The following well-known theorem of differential geometry relates the concepts of degeneracy and isolation.

Theorem 7.9 *Suppose a critical point* $\mathbf{u}_0$ *of* D *is nondegenerate, i.e. its Hessian is nonsingular. Then there exists a neighborhood* U *of* $\mathbf{u}_0$ *such that for all* $\mathbf{u} \in U - \{\mathbf{u}_0\}$, $\nabla D(\mathbf{u}) \neq \mathbf{0}$. *(A point* $\mathbf{u}_0$ *having this property is called an* isolated critical point.*)*

Proofs of this theorem are given in [34] [15]. Unfortunately, the converse is not true; all nondegenerate critical points are isolated, but some isolated critical points are degenerate.

For example, the function $f(x,y) = x^3 - 3xy^2$ has an isolated critical point at the origin, but its Hessian matrix there is the zero matrix and therefore singular. Nevertheless, this theorem is practically helpful in eliminating most cases.

We will use this theorem to help us detect curve branches of nonisolated stationary points as follows:

1. Run the PP or LP algorithm on our initial box of search, with a fairly coarse level of accuracy. For example, if we start with the search box $[0,1]^n$, we may run our root-finding algorithm with a tolerance of 10^{-2} or 10^{-3}.

2. Check the remaining bounding boxes after this step. If we observe a number of boxes adjacent to one another, there may be a curve in the solution set.

3. Use Newton-Raphson iteration to find roots within these boxes. Check the Hessian at these points; if it is singular, it is very likely that a curve exists. Accordingly, choose one of these roots as a starting point for our tracing technique.

This approach is by no means infallible; it is theoretically possible for there to be a number of degenerate isolated roots that will fool us into thinking that there is a curve involved. It is also possible that the solution set is a surface or even a higher dimensional entity. However, in practice these exceptions rarely occur.

Let us now assume that we have found a point on this curve, which we will call $\mathbf{f} : \mathbf{R} \to \mathbf{R}^n$. We will assume that $\mathbf{f}$ is a function of the parameter t, which takes on a value 0 at our starting point and that $\mathbf{f}$ has unit speed everywhere (i.e. $|\mathbf{f}(t)| = 1$ everywhere). This unit speed condition is imposed to ensure that $\mathbf{f}$ has a good parameterization.

Rewrite the equation $\nabla D = \mathbf{0}$ as a system of n equations in n unknowns:

$$\begin{cases} D_{u_1}(u_1, \ldots, u_n) = 0 \\ \ldots \\ D_{u_n}(u_1, \ldots, u_n) = 0 \end{cases} \tag{7.10}$$

Now, in order to stay on the curve $\mathbf{f}$ in moving a small distance from $\mathbf{f}(t)$ to $\mathbf{f}(t + dt)$, we need to know what increments have to be added to $u_1, \ldots, u_n$. Accordingly, we take a

Taylor expansion of each equation around the critical point $\mathbf{u_0}$ to first order:

$$\begin{cases} D_{u_1u_1}(\mathbf{u_0})du_1 + \ldots D_{u_1u_n}(\mathbf{u_0})du_n = 0 \\ \ldots \\ D_{u_nu_1}(\mathbf{u_0})du_1 + \ldots D_{u_nu_n}(\mathbf{u_0})du_n = 0 \end{cases} \tag{7.11}$$

Or, rewriting this as a matrix equation, we have

$$H\mathbf{du} = \mathbf{0} \tag{7.12}$$

where $\mathbf{du} = [du_1 \ \ldots \ du_n]^{\mathrm{T}}$ and H is the Hessian of $D(\mathbf{u})$ evaluated at $\mathbf{u_0}$.

Because $\mathbf{u_0}$ is degenerate, the rank of H should be less than n. In fact, we anticipate that it will be $n-1$, since any $\mathbf{du}$ satisfying (7.12) must be a vector tangent to $\mathbf{f}$ at $\mathbf{u_0}$, and hence the nullspace of H should have dimension 1. (It is possible on rare occasions that there is a curve passing through $\mathbf{u_0}$ but the rank of the Hessian is less than $n-1$. This problem may occur if the first-order Taylor expansion in (7.11) contains insufficent information due to zero derivatives. Fortunately, these cases are extremely rare in practice; see [34] for more information.)

Because we need to find a tangent vector to $\mathbf{f}$ at $\mathbf{u_0}$ with unit length, we need to find some vector in the nullspace of H and then normalize it to length 1. The Singular Value Decomposition method [43] may be applied to H to generate such a vector in a stable manner. Then, after making the vector unit length, we pass it to our ordinary differential equation solver as the tangent to $\mathbf{f}$ at $\mathbf{u_0}$. We can trace backwards by changing the sign of the tangent vector.

7.3 Remarks

We have shown the derivation of a tracing algorithm for a system of equations resulting from the gradient of a scalar field, since in our experience it is the most commonly occuring case. However, a similar approach may be attempted for more general systems of equations.

The problem of tracing surfaces as well as curves remains to be addressed. Certainly we anticipate surfaces to arise even less often than curves in stationary point problems, but they do occur. (Consider the minimum distance between two concentric spheres.) We

anticipate that one may generate a set of points on such a surface in a similar fashion; namely, by starting at a point on the surface, finding the basis vectors of the Hessian's nullspace at that point, and using these basis vectors to find other points nearby. This is a topic of future research.

Chapter 8

Examples and Applications

In this chapter, we give a few numerical examples to demonstrate how well the techniques presented in this thesis work. Then, we show how some other problems may be restated as systems of simultaneous polynomial equations and solved with one of the two algorithms. Finally, we include a running time analysis of our algorithms and provide some recommendations on choosing the most efficient algorithm.

8.1 Numerical Examples

The following six examples are designed to give the reader a better idea of the capabilities of the two algorithms. Implementation of both algorithms was in AT&T C++ version 2.0. C++ was chosen because both *rational and floating point* implementations could be written with little modification to the code.

8.1.1 Intersecting Two Simple Planar Curves

Our first example shows how to solve a simple planar intersection problem with both techniques. Suppose we want to intersect a circular arc with an elliptical arc

$$x^2 + y^2 - 1 = 0,\ \frac{x^2}{4} + 4y^2 - 1 = 0,\ 0 \le x \le 1,\ 0 \le y \le 1 \tag{8.1}$$

Simple algebra will show that these arcs intersect only in one point, $(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}})$, which to 9 digits is $(0.894427191, 0.447213595)$. In order to solve them with our methods, we first convert these two equations into the Bernstein basis, using the algorithm of appendix A.1.

Bounding Boxes from Projected-Polyhedron					
Iter	a_1	b_1	a_2	b_2	*Message*
1	0	1	0	1	Binary subdivision in x
2	0	0.5	0	1	No root in box
3	0.5	1.0	0.1875	0.61718750	None
4	0.74605306	0.98242188	0.37952697	0.48294336	None
5	0.86764611	0.92684826	0.44013056	0.45158886	None
6	0.89175192	0.89838481	0.44668154	0.44756546	None
7	0.89424540	0.89469787	0.44717954	0.44723644	None
8	0.89441574	0.89444424	0.44721146	0.44721503	None
9	0.89442648	0.89442826	0.44721346	0.44721368	None
10	0.89442715	0.89442726	0.44721359	0.44721360	None
11	0.89442719	0.89442719	0.44721360	0.44721360	Root Found

Table 8.1: Bounding boxes in the Projected-Polyhedron algorithm for a simple two variable case

This results in the system

$$\begin{bmatrix} b_{0,2}(x) & b_{1,2}(x) & b_{2,2}(x) \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_{0,2}(y) \\ b_{1,2}(y) \\ b_{2,2}(y) \end{bmatrix} = 0 \tag{8.2}$$

and

$$\begin{bmatrix} b_{0,2}(x) & b_{1,2}(x) & b_{2,2}(x) \end{bmatrix} \begin{bmatrix} -1 & -1 & 3 \\ -1 & -1 & 3 \\ \frac{3}{4} & \frac{3}{4} & \frac{13}{4} \end{bmatrix} \begin{bmatrix} b_{0,2}(y) \\ b_{1,2}(y) \\ b_{2,2}(y) \end{bmatrix} = 0. \tag{8.3}$$

We can now directly apply either method to this system. Running the Projected-Polyhedron algorithm gives us the results of table 8.1.

This particular example was run at a tolerance of 10^{-8}. Notice the linear convergence rate as we approach the root.

Now let us turn to the LP algorithm. On the first iteration, the LP routine will set up

Bounding Boxes $[a_1, b_1] \times [a_2, b_2]$ from LP-Find					
Iter	a_1	b_1	a_2	b_2	*Message*
1	0	1	0	1	None
2	0.375	1	0.1875	0.625	None
3	0.80012077	0.94619865	0.38069053	0.48248792	None
4	0.88925091	0.89831844	0.44364954	0.44931108	None
5	0.89440075	0.89444401	0.44720171	0.44722200	None
6	0.89442719	0.89442719	0.44721360	0.44721360	Root Found

Table 8.2: Bounding boxes in the Linear Programming algorithm for a simple two variable case

the constraint matrix C as follows:

$$\begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -1 & -1 & -1 \\ 0 & \frac{1}{2} & 1 & 0 & \frac{1}{2} & 1 & 0 & \frac{1}{2} & 1 & 0 & -\frac{1}{2} & -1 & 0 & -\frac{1}{2} & -1 & 0 & -\frac{1}{2} & -1 \\ -1 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 & -3 & 1 & 1 & -3 & \frac{3}{4} & \frac{3}{4} & -\frac{13}{4} \\ -1 & -1 & 0 & -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{8.4}$$

and the vector of right-hand-sides $\mathbf{r}$ will be

$$\mathbf{r}^T = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{8.5}$$

The vector of unknown convex hull coefficients $\mathbf{u}$ is constrained to satisfy $C\mathbf{u} = \mathbf{r}$. Now if, for example, we want to find a_1, the smallest x-coordinate of the bounding box, then we simply find the minimum value of $\mathbf{c}^T\mathbf{u}$ where $\mathbf{c}$ equals

$$\begin{bmatrix} 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{8.6}$$

b_1, a_2, and b_2 may be found similarly, with the same matrix C and right-hand side $\mathbf{r}$, but with different coefficient vectors $\mathbf{c}$. If we let the algorithm continue, we observe the results of table 8.2.

These results were obtained with a tolerance of 10^{-8}. Notice how this example illustrates the quadratic convergence of the linear-programming approach; the number of significant

digits of each coordinate approximately doubles at each step.

8.1.2 The Wilkinson Polynomial

Our next example attempts to solve the following twentieth degree polynomial which was shown by Wilkinson [45] to be extremely ill-conditioned:

$$P_{20}(x) = (x-1)(x-2)\dots(x-20). \tag{8.7}$$

In [10], it is shown that the roots of this polynomial may be extracted with much less error by considering the polynomial in the Bernstein basis before attempting solution. In accordance with this approach, we first scale the polynomial so that all 20 of the roots lie within $[0,1]$, i.e.

$$P(x) = \prod_{k=1}^{20}(x - \frac{k}{20}). \tag{8.8}$$

We begin by converting this polynomial into the Bernstein basis, as shown in Appendix A.1. Some care is needed here, because although once the polynomial has been transformed it is fairly stable, nevertheless the conversion process itself is ill-conditioned. To eliminate this problem, we start with exact rational coefficients and perform the conversion in exact rational arithmetic; once the process has been completed, we can easily convert the rational Bernstein coefficients into floating-point numbers with accuracy on the order of machine precision [19]. Since this is only a one-dimensional problem, both the PP and the LP techniques are quadratically convergent. Given a tolerance of 10^{-7}, the LP technique generated the results of table 8.3.

As the table indicates, both the linear programming and subdivision phases were stable enough to find all roots of this extremely unstable polynomial.

8.1.3 Significant Points of a Planar Algebraic Curve

A well known problem in computer-aided design is the discovery and subsequent tracing of all branches of an implicit algebraic curve $f(u,v) = 0$. The computation of *turning points* (where $f = \frac{\partial f}{\partial u} = 0$ or $f = \frac{\partial f}{\partial v} = 0$) or *critical points* (where $\frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} = 0$) is important in solving this problem [33] [38] [20] [37]. As an example, let us find the critical points of the

Roots of $P(x)$		
Root number	root	error
1	0.0500000002437804	2.4×10^{-10}
2	0.10000000000016	1.6×10^{-13}
3	0.150000000000427	4.3×10^{-13}
4	0.200000000000014	1.4×10^{-14}
5	0.249999999999785	2.2×10^{-13}
6	0.300000000009474	9.5×10^{-12}
7	0.350000000092444	9.2×10^{-11}
8	0.400000000006641	6.6×10^{-12}
9	0.450000024696511	2.5×10^{-8}
10	0.499999980911975	1.9×10^{-8}
11	0.549999999963676	3.6×10^{-11}
12	0.599999984022245	1.6×10^{-8}
13	0.649999993324551	6.7×10^{-9}
14	0.699999993937085	6.1×10^{-9}
15	0.749999999997894	2.1×10^{-12}
16	0.79999999503061	5.0×10^{-9}
17	0.849999968721786	3.1×10^{-8}
18	0.899999999995995	4.0×10^{-12}
19	0.949999989125186	1.1×10^{-8}
20	1.00000000000001	1.0×10^{-14}

Table 8.3: Roots of the Wilkinson Polynomial

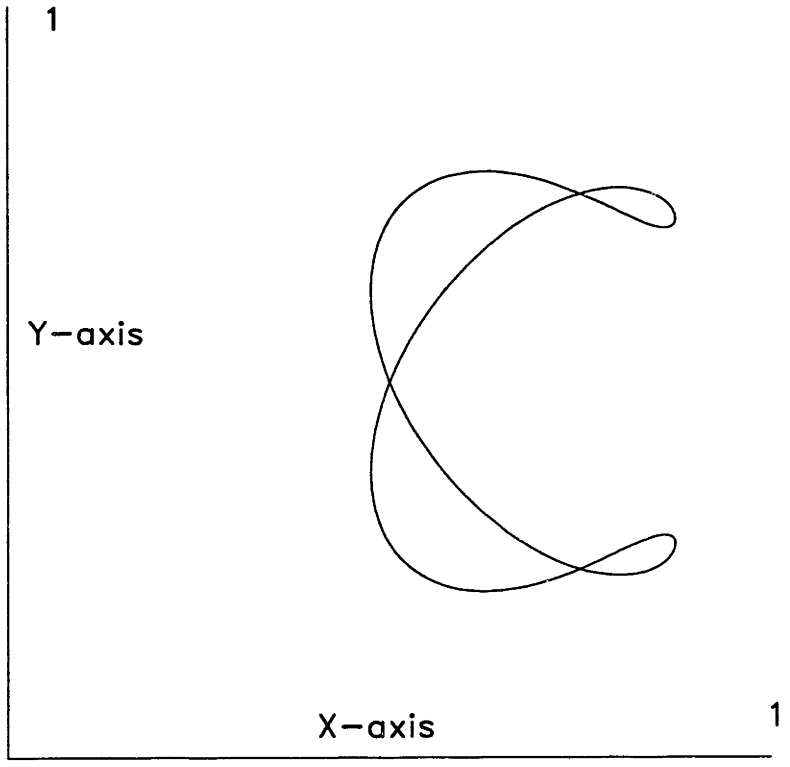

Figure 8-1: The curve $f(u,v) = 0$ of section 8.1.3

curve $f(u,v) = 0$ illustrated in figure 8-1, which is defined by

$$f(u,v) = -64v^4 + 128v^3 - 96u^2v^2 + 140uv - 139v^2 + 96u^2v - 140uv + 75v \qquad (8.9)$$
$$--96u^4 + 276u^3 - 313u^2 + 165u - 36 = 0$$

Taking the two partial derivatives gives us two simultaneous equations which we can use either method to solve. With a tolerance of 10^{-8}, the LP method computes the roots in table 8.4.

8.1.4 Stationary Points of a Distance Function

Our next example is essentially a problem of finding extrema. Suppose we are given two implicit algebraic surfaces $f(x,y,z) = 0$ and $g(x,y,z) = 0$. Now consider all possible pairs of points $\mathbf{p_1} = (x_1, y_1, z_1)$ and $\mathbf{p_2} = (x_2, y_2, z_2)$, where $\mathbf{p_1}$ lies on f and $\mathbf{p_2}$ lies on g. Suppose we want to find those pairs of points which are farthest apart, or closest together. Then

Critical points of f to 8 digits		
Root number	u	v
1	0.92143996	0.50000000
2	0.83640890	0.73280861
3	0.75000000	0.75000000
4	0.73481004	0.50000000
5	0.55109110	0.69759134
6	0.50000000	0.50000000
7	0.83640890	0.26719139
8	0.75000000	0.25000000
9	0.55109110	0.30240866

Table 8.4: Critical points of an algebraic curve

clearly these points are extrema of the square of the distance metric

$$D = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 \tag{8.10}$$

As an example, let

$$f(x_1, y_1, z_1) = (x_1 - \frac{1}{5})^2 + (y_1 - \frac{1}{5})^2 + (z_1 - \frac{1}{5})^2 - \frac{1}{25} = 0 \tag{8.11}$$

$$g(x_2, y_2, z_2) = (x_2 - \frac{1}{5})^2 + (y_2 - \frac{1}{5})^2 + (z_2 - \frac{4}{5})^2 - \frac{1}{25} = 0. \tag{8.12}$$

We will consider x_1, y_1, x_2, and y_2 to be independent variables, while z_1 depends on x_1 and y_1 and z_2 depends on x_2 and y_2. Now, for $(x_1, y_1, z_1, x_2, y_2, z_2)$ to be an extremum, the partial derivatives of D with respect to the independent parameters must be 0. Accordingly, consider x_1, y_1, x_2, and y_2 to be independent, and take partial derivatives of D with respect to these four variables. We obtain

$$\frac{\partial D}{\partial x_1} = 2(x_1 - x_2) + 2(z_1 - z_2)\frac{\partial z_1}{\partial x_1} = 0 \tag{8.13}$$

and similar expressions for the other three derivatives. Implicit differentiation of (8.11) and (8.12), gives us

$$\frac{\partial z_1}{\partial x_1} = -\frac{x_1 - \frac{1}{5}}{z_1 - \frac{1}{5}} \tag{8.14}$$

and similar expressions for the partial derivatives $\frac{\partial z_1}{\partial y_1}$, $\frac{\partial z_2}{\partial x_2}$, and $\frac{\partial z_1}{\partial y_1}$. Substituting these four

Critical points of the distance function to 8 digits						
Root number	x_1	y_1	z_1	x_2	y_2	z_2
1	0.2	0.2	0	0.2	0.2	1
1	0.2	0.2	0.4	0.2	0.2	1
1	0.2	0.2	0	0.2	0.2	0.6
1	0.2	0.2	0.4	0.2	0.2	0.6

Table 8.5: Critical points of a distance function

equations into the partial derivatives for D and simplifying gives

$$(x_1 - x_2)(z_1 - \frac{1}{5}) - (x_1 - \frac{1}{5})(z_1 - z_2) = 0 \tag{8.15}$$

$$(y_1 - y_2)(z_1 - \frac{1}{5}) - (y_1 - \frac{1}{5})(z_1 - z_2) = 0 \tag{8.16}$$

$$(x_2 - x_1)(z_2 - \frac{4}{5}) - (x_2 - \frac{1}{5})(z_2 - z_1) = 0 \tag{8.17}$$

$$(y_2 - x_1)(z_2 - \frac{4}{5}) - (y_2 - \frac{1}{5})(z_2 - z_1) = 0 \tag{8.18}$$

These four equations together with (8.11) and (8.12) form a system of six polynomial equations in six unknowns, which may now be converted to Bernstein form and solved with the PP technique.

The PP technique yields the four roots in table 8.5 when run at a tolerance of 10^{-8}:

8.1.5 A Stationary Point Problem at a Strict Tolerance

Consider the function

$$f(x,y) = [(x - \frac{1}{4})^3 + (y - \frac{1}{4})^3 - (\frac{3}{5})^3]^2 + [(x - \frac{3}{4})^3 + (y - \frac{3}{4})^3 - (\frac{3}{5})^3]^2 \tag{8.19}$$

and set its partial derivatives equal to zero. Solving the resulting system at a strict tolerance of 10^{-14} gives one real root at (0.72660262158698, 0.72660262158698).

8.1.6 A Distance Problem with Tangency

Consider the first quadrant of the unit circle $x^2 + y^2 = 1$ and the portion of the parabola $y = 1 - x^2$ lying above the x-axis, both expressed as parametric rational Bézier curves.

We wish to compute the stationary points of the squared distance between the two curves; accordingly, we set $D(t,u) = (x_1 - x_2)^2 + (y_1 - y_2)^2$ (where $(x_1(t), y_1(t))$ is the parametric equation for the parabola and $(x_2(u), y_2(u))$ is the parametric equation for the circle), take partial derivatives with respect to t and u, and set them equal to zero. Because these curves are both assumed to be rational, we end up with one equation of degree 8 in t and degree 9 in u, and one equation of degree 8 in u and degree 9 in t. In this example, because there is a tangency at $(0,1)$ in the (x,y)-plane and the curves are nearly parallel, the rate of convergence drops and an extensive amount of binary subdivision is performed; consequently, this example takes a fairly long time to run to a tolerance of 10^{-8}. This example has the following pairs of minimum distance points: $(0,1)$ on the circle and $(0,1)$ on the parabola; $(1,0)$ and $(1,0)$; $(\frac{\sqrt{2}}{\sqrt{3}}, \frac{1}{\sqrt{3}})$ and $(\frac{1}{\sqrt{2}}, \frac{1}{2})$.

8.2 Other Applications

In closing this chapter, we will briefly touch on some other applications of our algorithms and how they might be used or extended to solve related problems.

8.2.1 Systems of Piecewise Polynomials

Up to this point, we have said nothing about extending this algorithm to piecewise polynomials. However, since B-splines possess the convex hull property, such an extension should be fairly straightforward. One simple way to solve such a problem, of course, would be to split the piecewise polynomials into separate Bernstein polynomials [2] [5] in order to solve a number of different polynomial subproblems. But since both the PP and LP algorithms rely only the convex hull property to generate new bounding boxes, reformulating the problem in the Bernstein basis is unnecessary.

Problems analogous to the four we have examined above are often encountered with piecewise polynomial entities. Determining the stationary points of the distance function between two piecewise polynomial entities may seem difficult because multiplication of piecewise polynomials is required; however, recent research by Mørken [26] has shown that this is a tractable problem. Whether or not this multiplication of piecewise polynomials represents a significant improvement over subdivision into Bernstein components and then multiplication of Bernstein polynomials remains to be studied.

8.2.2 More Equations than Unknowns, or Vice-Versa

Throughout this thesis, we have been assuming that the number of equations is equal to the number of unknowns, mostly because the problem is easier to think of in this form. However, there is no particular reason that the two numbers must be equal. One common example is determining the singular points of an implicit algebraic curve $f(u, v) = 0$. These points are the solutions of the system $f = \frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} = 0$, which involves three equations in two unknowns. Solving this problem requires nothing more than a few more constraints in the LP approach, or one more polyhedron to project in the PP method.

Fewer equations than unknowns should also be possible to solve. For example, such problems arise in engineering design of complex structures.

8.2.3 Finding Complex Roots

These techniques can easily be applied to find complex roots as well as real roots. Simply substitute $\mathbf{u} + i\mathbf{v}$ for $\mathbf{x}$ in (4.3), where i is the imaginary unit, and separate the real and imaginary parts. In this formulation, the coefficients $w_I^{(k)}$ may be complex. In general, for n simultaneous complex polynomial equations, one obtains $2n$ real equations.

8.3 Choosing an Algorithm

Looking at the complexity estimates per iteration of sections 5.3.2 and 6.2.1 gives a clear indication of which algorithms perform better under various circumstances. There are two cases to consider. If $n^3 \leq m$, the second term of the LP complexity estimate is more important, and thus both the LP and PP algorithms have the same asymptotic time of $\mathcal{O}(n^2 m^{n+1})$. Otherwise, the PP is faster by a factor of $\frac{n^3}{m}$. Here n is the number of variables, m is the degree of each variable in each equation. The LP algorithm will therefore be better when n is small and when the number of iterations required is considerably less. Because the LP algorithm exhibits quadratic convergence near simple roots, we expect it to perform exceptionally well when the tolerance is fairly strict. This is in fact exactly what we observe in practice. In addition, numerical experiments suggest that for small n, the LP is more efficient than the PP for multiple roots.

However, if the problem in question has a large number of roots in the domain, both algorithms are forced to perform a large number of subdivision steps in the early stages of

CPU timings (in seconds)						
	1	2	3	4	5	6
PP	*	0.1	0.4	4.2	1.7	23000
LP	*	0.2	1.1	225	0.7	1810
HY	*	0.2	0.4	65	0.5	1940
Tol	10^{-8}	10^{-7}	10^{-8}	10^{-8}	10^{-14}	10^{-8}

Table 8.6: Comparison of CPU times of PP, LP, and hybrid algorithms in test cases

the algorithm; in such cases, when many iterations are needed, the PP algorithm tends to perform better. Since we do not know *a priori* how many roots a given problem will have, a *hybrid algorithm* based on both methods is useful for low n. In this approach, we perform a number of fast PP iterations until the sizes of the bounding boxes are smaller than a certain coarse tolerance (for example, 10^{-3}). After the boxes drop below this size, we employ a few LP iterations in order to take advantage of quadratic convergence. This hybrid approach typically performs well for most $n = 2$ and some $n = 3$ problems. However, for larger n, LP iterations become too expensive, and we recommend a pure PP approach for such cases.

These observations are reflected in table 8.6. The timings correspond to results obtained on a workstation running at 36 MHz. In this table, the PP, LP and hybrid (HY) algorithms are all tested in double precision floating point arithmetic. For the hybrid algorithm, the PP was run at a tolerance of 10^{-4} before beginning LP iterations. Under the time tests is a row indicating the tolerance at which the examples were run; the examples themselves are numbered according to their section. For instance example number 2 is the Wilkinson polynomial of section 8.1.2. An entry of * in the table means that the example executed too quickly to be measured.

Chapter 9

Concluding Remarks

As mentioned in the introduction, although both algorithms are formulated to determine real roots, they can be used to find complex roots as well by simply substituting $u+iv$ for x in every equation and splitting each equation into real and imaginary parts. Furthermore, both algorithms may be extended to systems of m equations in n unknowns, where $m \neq n$; such over- and under-determined systems may appear in the determination of singular points of a planar implicit algebraic curve and in engineering design. Such problems merely require a different number of constraints in the LP method, or a different number of polyhedra to project in the PP method.

We have experimented with exact rational arithmetic as well as floating point in our implementations of the PP and LP methods. The rational arithmetic implemementation was motivated by robustness considerations. However, our experience indicates that this implementation tends to be expensive for high-degree and high-dimensional problems. In addition, our experience suggests that both PP and LP methods are inherently stable; nevertheless, an implementation of both methods in floating-point *interval arithmetic* could be quite useful in achieving robustness. Recent research in this direction is reported in [24] [22] [23] and in [17].

A few modifications to the algorithms could improve performance. For example, either method might be altered to intersect polynomials expressed in terms of barycentric coordinates [9]. This has already been done for $n = 2$ in [39] using projections of convex hulls. Another improvement would be the ability to select, by examing the pattern of bounding-box reduction, a better place than the midpoint to split a box when convergence slows.

The relative merits of our methods and Newton-based interval techniques should also be studied. An algorithm along the lines of the proposed technique in section 7.2 needs to be implemented in order to make our algorithms capable of handling non-isolated roots. Finally, more examples need to be run on fully optimized versions of the two algorithms. The resulting comparisons between the two and between other methods would help in choosing an algorithm to fit a given class of problems.

Appendix A

Supplemental Algorithms

A.1 Converting from the Power to the Bernstein Basis

In this section we present some pseudocode which demonstrates how to convert a polynomial in n variables from the power basis to the Bernstein basis. On input, we assume that we are given an array m indexed from 1 to n and an n-dimensional array a of power basis coefficients indexed in the ith dimension from 0 to $m[i]$. The Bernstein coefficients are stored in the identically sized n-dimensional array b.

This particular algorithm is a simple adaptation to polynomials in n variables of the algorithm for converting univariate polynomials given in [10]. The extension is straightforward, so we omit the derivation for the sake of brevity.

In the code of figure A-1, we have glossed over a few important details of implementation for the sake of clarity; for example, no mainstream computer language will actually allow the expression $a[j_1][j_2]\ldots[j_n]$ to index an n-dimensional array unless n is fixed at compile-time. (However, this problem may be resolved by using one-dimensional arrays instead of n-dimensional arrays. For example, a three-dimensional array a with elements ranging from $a[0][0][0]$ to $a[m_1][m_2][m_3]$ could be stored as a one dimensional array b of $(m_1 + 1)(m_2 + 1)(m_3 + 1)$ elements. With this scheme, $a[i][j][k]$ would be accessed by referencing $b[i(m_2+1)(m_3+1)+j(m_3+1)+k]$. This is the sort of approach most computer languages take in addressing multidimensional arrays.)

The for loop in line 2 indicates that we loop over all possible multi-indices $(j_1, j_2, \ldots, j_{i-1}, j_{i+1}, \ldots, j_n)$ which are bounded by the multi-index $(m[1], m[2], \ldots, m[i-1], m[i+1], \ldots, m[n])$.

```
POWER-TO-BERN (a, b, m, n)
1  for i ← 1 to n
2      for (j_1, j_2, ..., j_{i-1}, j_{i+1}, ..., j_n) ← (0, 0, ..., 0)
           to (m[1], m[2], ..., m[i − 1], m[i + 1], ..., m[n])
3          for k ← 1 to m[i]
4              sum ← 0
5              for l ← 1 to m[i]
6                  sum ← sum+
                       a[j_1][j_2]...[j_{i-1}][l][j_{i+1}]...[j_n]BINOMIAL(k, l)/BINOMIAL(m[i], l)
7              b[j_1][j_2]...[j_{i-1}][k][j_{i+1}]...[j_n] ← sum
```

Figure A-1: Pseudocode for basis conversion

A.2 Subdivision of a Multivariate Bernstein Polynomial

In this section we give pseudocode which will allow us to subdivide a polynomial f_0 of n variables $x_1, x_2, \ldots, x_n$. We assume that on input we are given two arrays of real numbers a and b which define a box $B = [a[1], b[1]] \times [a[2], b[2]] \times \ldots \times [a[n], b[n]] \subset [0,1]^n$. The act of subdivision will allow us to create a new polynomial f which is defined by

$$f(x_1, x_2, \ldots, x_n) = f_0\{a[1] + (b[1] - a[1])x_1, a[2] + (b[2] - a[2])x_2, \ldots, a[n] + (b[n] - a[n])x_n\} \tag{A.1}$$

We assume that f_0 and f are n-dimensional arrays indexed in the ith dimension from 0 to $m[i]$, where m is an array indexed from 1 to n. The method of subdivision is De Casteljau subdivision, and the algorithm below is an extension of similar algorithms for 1 and 2 dimensions given in [9]. Because of this similarity, we do not bother to give a proof that it works.

```
DC-BOX-SUBDIVIDE (f_0, a, b, m, n, f)
1  for i ← 1 to n
2  if b[i] = 0 a0 ← 0 else a0 ← a[i]/b[i]
3  b0 ← b[i]
4      r ← 2D-ARRAY (0, m[i], 0, m[i])
5      for (j_1, j_2, ..., j_{i-1}, j_{i+1}, j_n) ← (0, 0, ..., 0)
           to (m[1], m[2], ..., m[i − 1], m[i + 1], ..., m[n])
6          for k ← 1 to m[i]
7              r[k][0] ← f_0[j_1][j_2] ... [j_{i-1}][k][j_{i+1}] ... [j_n]
8          for k ← 1 to m[i]
9              for l ← k to m[i]
10                 r[l][k] ← (1 − b)r[i − 1][k − 1] + b r[i][k − 1]
11         for k ← 1 to m[i]
12             r[k][0] ← r[k][k]
13         for k ← 1 to m[i]
14             for l ← k to m[i]
15                 r[l][k] ← (1 − a)r[i − 1][k − 1] + a r[i][k − 1]
16         for k ← 1 to m[i]
17             f[j_1][j_2] ... [j_{i-1}][k][j_{i+1}] ... [j_n] ← r[m[i]][m[i] − k]
```

Figure A-2: Pseudocode for multivariate subdivision

Bibliography

[1] C. Bliek. *Computer Methods for Design Automation.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, July 1992.

[2] W. Boehm. Inserting new knots into B-spline curves. *Computer Aided Design*, 12(4):199–201, 1980.

[3] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory: Progress, Directions and Open Problems in Multidimensional Systems*, pages 184–232, 1985. Dordrecht, Holland: D. Reidel Publishing Company.

[4] J. Canny. Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9:241–250, 1990.

[5] E. Cohen, T. Lyche, and R. F. Riesenfeld. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing*, 14:87–111, 1980.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[7] G. Dahlquist and A. Björck. *Numerical Methods.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[8] T. Dokken. Finding intersections of B-spline represented geometries using recursive subdivision techniques. *Computer Aided Geometric Design*, 2:189–195, 1985.

[9] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press, San Diego, CA, 3rd edition, 1993.

[10] R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191–216, March 1987.

[11] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.

[12] C. B. Garcia and W. I. Zangwill. Global continuation methods for finding all solutions to polynomial systems of equations in n variables. In A. V. Fiacco and K. O. Kortanek, editors, *Extremal Methods and Systems Analysis*, pages 481–497. Springer-Verlag, New York, NY, 1980.

[13] S. I. Gass. *Linear Programming: Methods and Applications.* McGraw-Hill, New York, 1985.

[14] A. Geisow. *Surface Interrogations.* PhD thesis, School of Computing Studies and Accountancy, University of East Anglia, Norwich NR47TJ, U. K., July 1983.

[15] M. Golubitsky and V. Guillemin. *Stable Mappings and their Singularities.* Springer-Verlag, New York, 1973.

[16] F. B. Hildebrand. *Advanced Calculus for Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[17] C.-Y. Hu. *Robust Algorithms for Sculptured Shape Visualization.* Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, July 1993.

[18] R. B. Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. *Computing*, 47:169–191, 1991.

[19] D. E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 1981. 2nd Edition.

[20] G. A. Kriezis, P. V. Prakash, and N. M. Patrikalakis. A method for intersecting algebraic surfaces with rational polynomial patches. *Computer Aided Design*, 22(10):645–654, December 1990.

[21] J. M. Lane and R. F. Riesenfeld. Bounds on a polynomial. *BIT: Nordisk Tidskrift for Informations-Behandling*, 21(1):112–117, 1981.

[22] T. Maekawa. *Robust Computational Methods for Shape Interrogation.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1993.

[23] T. Maekawa and N. M. Patrikalakis. Interrogation of differential geometry properties for design and manufacture. *The Visual Computer*, May 1992. Revised April 1993. To appear.

[24] T. Maekawa and N. M. Patrikalakis. Computation of singularities and intersections of offsets of planar curves. *Computer Aided Geometric Design*, 10:407–429, 1993.

[25] D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991.

[26] K. Mørken. Some identities for products and degree raising of splines. *Constructive Approximation*, 7:195–208, 1991.

[27] A. Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, Cambridge, 1990.

[28] T. Nishita, T. W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *ACM Computer Graphics*, 24(4):337–345, August 1990.

[29] Numerical Algorithms Group, Oxford, England. *NAG Fortran Library Manual, Volumes 1-8*, Mark 14 edition, 1990.

[30] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[31] N. M. Patrikalakis and G. A. Kriezis. Representation of piecewise continuous algebraic surfaces in terms of B-splines. *The Visual Computer*, 5(6):360–374, 1989.

[32] N. M. Patrikalakis, T. Maekawa, E. C. Sherbrocke, and J. Zhou. Computation of singularities for engineering design. In T. L. Kunii and Y. Shinagawa, editors, *Modern Geometric Computing for Visualization*, pages 167–191. Tokyo: Springer-Verlag, June 1992.

[33] N. M. Patrikalakis and P. V. Prakash. Surface intersections for geometric modeling. *Journal of Mechanical Design, ASME Transactions*, 112(1):100–107, March 1990.

[34] T. Poston and I. Stewart. *Catastrophe Theory and its Applications.* Pitman, San Francisco, CA, 1978.

[35] P. V. Prakash and N. M. Patrikalakis. Surface-to-surface intersections for geometric modeling. Technical Report MITSG 88-8, Cambridge, MA: MIT Sea Grant College Program, 1988.

[36] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[37] T. Sakkalis. The topological configuration of a real algebraic curve. *Bulletin of the Australian Mathematical Society*, 43:37–50, 1991.

[38] T. Sakkalis and R. T. Farouki. Singular points of algebraic curves. *Journal of Symbolic Computation*, 9:405–421, 1990.

[39] T. W. Sederberg. Algorithms for algebraic curve intersection. *Computer Aided Design*, 21(9):547–554, November 1989.

[40] T. W. Sederberg and T. Nishita. Geometric Hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.

[41] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the real solutions of nonlinear polynomial systems. Memorandum 91-12, Cambridge MA: MIT Ocean Engineering Design Laboratory, October 1991. Revised June 1992.

[42] E. C. Sherbrooke and N. M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10:379–405, 1993.

[43] G. Strang. *Linear Algebra and its Applications.* Harcourt Brace Jovanovich, San Diego, CA, 1988.

[44] M. E. Vafiadou and N. M. Patrikalakis. Interrogation of offsets of polynomial surface patches. In F. H. Post and W. Barth, editors, *Eurographics '91, Proceedings of the 12th Annual European Association for Computer Graphics Conference and Exhibition*, pages 247–259 and 538, Vienna, Austria, September 1991. Amsterdam: North-Holland.

[45] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* H. M. Stationery Office, London, England, 1963.

[46] W. I. Zangwill and C. B. Garcia. *Pathways to solutions, fixed points, and equilibria.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[47] J. Zhou, E. C. Sherbrooke, and N. M. Patrikalakis. Computation of stationary points of distance functions. *Engineering with Computers*, June 1992. Revised March 1993. To appear.