Enclosure Methods for Systems of Polynomial Equations and Inequalities

Dissertation

zur Erlangung des akademischen Grades des
Doktors der Naturwissenschaften (Dr. rer. nat.)
an der

Universität
Konstanz

Mathematisch-Naturwissenschaftliche Sektion
Fachbereich Mathematik und Statistik

vorgelegt von

Andrew Paul Smith

Tag der mündlichen Prüfung: 5. Oktober 2012
1. Referent: Prof. Dr. Jürgen Garloff, Universität Konstanz und HTWG Konstanz
2. Referent: Prof. Dr. Jon Rokne, University of Calgary

Abstract

Many problems in applied mathematics can be formulated as a system of nonlinear equations or inequalities, and a broad subset are those problems consisting either partially or completely of multivariate polynomials. Broadly speaking, an 'enclosure' method attempts to solve such a problem over a specified region of interest, commonly a box subset of $\mathbb{R}^n$, where n is the number of unknowns. Subdivision (or branch-and-bound) is a commonly-applied scheme, wherein a starting box is successively subdivided into sub-boxes; sub-boxes for which a proof of non-existence of a solution can be completed are discarded. As a component of such, a boundary method attempts to exploit the properties of component functions over the boundary of a sub-box, without considering their behaviour within it.

Two main types of non-existence proof are considered for polynomial systems over boxes or sub-boxes. Firstly, the topological degree of a system of equations (considered as a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$) can be computed over a sub-box, which possesses a root-counting property. Together with an enclosure for the determinant of the Jacobian, the existence (or otherwise) of roots in the sub-box can be ascertained. Secondly, and alternatively, a range-enclosing method can be used to compute a guaranteed outer enclosure for the range of a multivariate polynomial over a sub-box; if it does not contain zero, a root is excluded. The Bernstein expansion is used, due to the tightness of the enclosure yielded and its rate of convergence to the true enclosure. In both cases, interval arithmetic is employed to obtain guaranteed enclosures and ensure the rigour of these existence tests.

Advances have been made in four main areas. Firstly, an existing recursive algorithm for the computation of topological degree is investigated in detail, including a complexity analysis, and algorithmic improvements are proposed. Secondly, a simple branch-and-bound method for systems of polynomial equations, utilising Bernstein expansion and an existence test by Miranda, is developed, and alternative subdivision strategies are considered. Thirdly, a major improvement of the Bernstein expansion itself is achieved with the development of the implicit Bernstein form, which exhibits greatly improved performance for many categories of polynomials. Finally, several new methods are developed and compared for the computation of affine lower bounding functions for polynomials, which may be employed in branch-and-bound schemes for problems involving inequalities, such as global optimisation problems. Numerical results and an overview of the developed software are given.

Zusammenfassung

Viele Probleme der angewandten Mathematik können als Systeme nichtlinearer Gleichungen oder Ungleichungen formuliert werden. Eine große Teilmenge davon besteht teilweise oder vollständig aus multivariablen Polynomen. Die Idee einer Einschließungsmethode löst ein solches Problem innerhalb einer relevanten Region. Gewöhnlich ist diese Region eine Box-Teilmenge von $\mathbb{R}^n$, wobei n die Anzahl der Unbekannten ist. Subdivision (oder Branch-and-Bound) ist ein oft verwendetes Schema, bei dem eine Anfangsbox sukzessiv in Subboxen unterteilt wird. Subboxen, für die die Nichtexistenz einer Lösung bewiesen werden kann, werden ausgeschlossen. Als Bestandteil dieses Schemas ist die Randmethode, die versucht die Randeigenschaften der Komponentenfunktionen bezüglich der Subbox auszunutzen ohne dabei das Verhalten innerhalb der Subbox zu betrachten.

Die Nichtexistenzbeweise für Systeme polynomer Gleichungen über Boxen oder Subboxen werden durch zwei Haupttypen beschrieben. Der erste Typ kann durch den topologischen Grad eines Gleichungssystems (als eine Abbildung von $\mathbb{R}^n$ auf $\mathbb{R}^n$) über eine Subbox beschrieben werden. Diese besitzt die Eigenschaft, die Anzahl der Wurzeln zählen zu können. Dabei kann die Existenz (oder Nichtexistenz) von Wurzeln in dieser Subbox durch die Bestimmung einer Einschließung der Determinanten der Jacobi-Matrix ermittelt werden. Als zweiter Grundtyp bildet die Bereichseinschließungmethode, die eine äußere Einschränkung des Bildbereichs eines multivariablen Polynoms durch eine Subbox garantiert. Enthält dieses Intervall nicht Null, so kann auch das Vorkommen einer Wurzel ausgeschlossen werden. Fokus der Bernstein-Erweiterung ist die Bestimmung präziserer Grenzen für die Einschließung und eine schnellere Konvergenz gegen die exakte Einschließung. Um eine garantierte Einschließung und die Korrektheit dieses Existenz-Tests sicherzustellen, wird in beiden Fällen die Intervallarithmetik angewandt.

Es wurden Fortschritte in vier Bereichen erzielt. Erstens wurde ein existierender rekursiver Algorithmus für die Berechnung des topologischen Grads detailiert untersucht. Dabei liegt die Betonung auf die Komplexitätsanalyse und algorithmische Verbesserung und Erweiterungen. Der zweite Bereich beschäftigt sich zunächst mit einer einfachen Branch-and-Bound-Methode für polynome Gleichungssysteme. Dafür wurden die Bernstein-Erweiterung und der Existenz-Test von Miranda verwendet. Weiterhin wurden alternative Strategien zur Subdivision in Betracht gezogen. Drittens wurde eine wesentliche Verbesserung der Bernstein-Erweiterung durch die Entwicklung der impliziten Bernstein-Form erzielt. Es zeigt sich eine starke Verbesserung der Leistung für viele Arten von Polynomen. Abschließend wurden mehrere neue Verfahren für die Berechnung von unteren affinen Grenzfunktionen für Polynome entwickelt und miteinander verglichen. Diese neue Verfahren können für Branch-and-Bound-Verfahren bei Problemen mit Ungleichungen, wie z.B. globale Optimierungsprobleme, eingesetzt werden. Numerische Ergebnisse und eine Übersicht der entwickelten Software liegen vor.

Acknowledgements

First and foremost, I would like to warmly thank the two supervisors of this work, Dan Richardson (during my time in Bath, U.K.) and Jürgen Garloff (during my time in Konstanz, Germany), for their expert, friendly, and productive supervision and co-operation, as well as their patience and generosity.

I would like to thank a number of people with whom I have been fortunate enough to co-write research papers, for their productive co-operation, principally Jürgen Garloff, but also (in chronological order) Christian Jansson, Ismail Idriss, Laurent Granvilliers, Evgenija Popova, Horst Werkle, and Riad Al-Kasasbeh.

I would like to thank Hermann Schichl and Evgenija Popova for making their software available to me and for their technical assistance. Thanks also to the authors and maintainers of the *filib++* software package, which I have used extensively.

I would like to thank my family for their love and support; in particular my mother, who never gave up gently pestering me to complete this thesis during the excessively long hiatus — I can't believe it's finished it, either!

I'd like to pay tribute to a number of people who have helped to keep me (somewhat) sane during this odyssey in various ways: From the gaming group, I'd like to mention Christoph, Volker, Jan, Ole, Martin, Stefan, Sünje, Regina, Kris, Tamara, Patrick, and Christine. From the Institute for Optical Systems (IOS) at the HTWG Konstanz, Dang Le, Alex, David, Marcus, Thomas, Jan, Andy, Manuel, Klaus, Matthias Franz, as well as several others. I'd like to thank Dang for his support and motivation during our mutual struggles to write up our theses! I'd also like to thank several friends in the U.K., Mark, Richard, Phil, Caroline and Stuart, and Russ for staying in contact (in spite of my frequent silences!) during my long sojourn in Germany.

I would like to thank a number of people who have helped to arrange financial support and employment in various ways: at the University of Bath, Dan Richardson; at the HTWG Konstanz, Jürgen Garloff, Horst Werkle, Florin Ionescu, and Riad Al-Kasasbeh (Al-Balqa Applied University, Jordan); at the University of Konstanz, Jürgen Garloff, Markus Schweighofer, Stefan Volkwein, and Rainer Janßen. In particular I would like to thank Riad for his motivational support and his helpful urging during the final stages of this work, as well as his friendship and his help with potential future employment.

I would like to gratefully acknowledge several funding agencies that have supported this work, either directly or indirectly. The University of Bath and the Engineering and Physical Sciences Research Council (EPSRC), in the U.K., and the University of Konstanz, in Germany, have partially funded my doctoral research with stipendia. The German Research Foundation (DFG), the German Ministry of Education, Science, Research, and Technology, the State of Baden-Württemberg, and the University of Applied Sciences (HTWG) Konstanz, in Germany, have funded several research projects on which have I have worked together with Jürgen Garloff; some of that research appears in this thesis.

I would like to thank the two referees, Jürgen Garloff and Jon Rokne, as well as the oral examiners, for taking the time to review my work. For proofreading, thanks to Volker Bürkel, Martin Spitzbarth, and Dang Le.

I would like to dedicate this thesis to the memory of my two grandfathers, Frank Smith and William Brown, both of whom sadly passed away before the completion of this work.

By a serendipitous coincidence, 2012 marks the occasion of the 100th anniversary of both L. E. J. Brouwer's proof of the fixed point theorem (and the accompanying introduction of the Brouwer degree) and S. N. Bernstein's proof of the Weierstrass approximation theorem (and the accompanying introduction of the Bernstein polynomials).

Notation

General Notation

$conv$	convex hull
deg	Brouwer degree
δ	an error term or discrepancy
ε	a small number or tolerance
σ	an orientation (either equal to $+1$ or -1)
∂	set boundary
cl	set closure
$\#$	set cardinality
P	probability function
E	statistical expectation function

Notation for One Dimension

$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}$	the set of integers
$\mathbb{R}$	the set of real numbers
$\mathbb{C}$	the set of complex numbers
$\mathbb{IR}$	the set of non-empty bounded and closed intervals over $\mathbb{R}$
x	a real-valued variable
$\mathbf{x} = [\underline{x}, \overline{x}]$	an interval from $\mathbb{IR}$; it is delimited with square brackets, an underscore denotes a lower bound and an overscore an upper bound
$\mathbf{I} = [0, 1]$	the unit interval
$f : \mathbb{R} \to \mathbb{R}$	a real-valued function
$\mathbf{f} : \mathbb{IR} \to \mathbb{IR}$	an interval-valued function
$p : \mathbb{R} \to \mathbb{R}$	a polynomial
$q : \mathbb{R} \to \mathbb{R}$	a monomial
$c : \mathbb{R} \to \mathbb{R}$	an affine bounding function
l	the degree of a polynomial
$B_i^l : \mathbb{R} \to \mathbb{R}$	the ith Bernstein basis polynomial of degree l
a_i	ith coefficient of a polynomial in power form
b_i	ith coefficient of a polynomial in Bernstein form (Bernstein coefficient)
$\mathbf{b}_i$	control point associated with the Bernstein coefficient b_i
mid	midpoint of an interval
r	radius of an interval
w	width of an interval
$dist$	(Hausdorff) distance between two intervals
$\binom{l}{i}$	$:= \frac{l!}{i!(l-i)!}$ (the binomial coefficient)

Notation for Multiple Dimensions

n	dimension and/or number of variables
m	number of functions, usually equal to n
$\mathbb{R}^n$	n-ary Cartesian product of $\mathbb{R}$
$\mathbb{IR}^n$	n-ary Cartesian product of $\mathbb{IR}$
$\mathbb{Q}[x_1, \ldots, x_n]$	the set of polynomials in real variables $x_1, \ldots, x_n$
$x = (x_1, \ldots, x_n)$	a variable taking values from $\mathbb{R}^n$
$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$	a box in $\mathbb{IR}^n$ (a Cartesian product of n intervals)
$\mathbf{I} = [0, 1]^n$	the unit box
s or $\mathbf{X}_i^{\pm}$	a face of a box
L	an array of faces
$\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$	an n-tuple of real-valued functions in n variables
$f : \mathbb{R}^n \to \mathbb{R}$	a real-valued function in n variables
$\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$	an interval-valued function in n interval variables
$\mathcal{P} : \mathbb{R}^n \to \mathbb{R}^n$	an n-tuple of multivariate polynomials
$p : \mathbb{R}^n \to \mathbb{R}$	a multivariate polynomial
$q : \mathbb{R}^n \to \mathbb{R}$	a multivariate polynomial or monomial
$c : \mathbb{R}^n \to \mathbb{R}$	a multivariate affine bounding function
$g : \mathbb{R}^n \to \mathbb{R}$	a real-valued function defining an inequality constraint
$h : \mathbb{R}^n \to \mathbb{R}$	a real-valued function defining an equality constraint
$H : [0, 1] \times \mathbb{R}^n \to \mathbb{R}^n$	a homotopy between two functions mapping $\mathbb{R}^n$ to $\mathbb{R}^n$
$l = (l_1, \ldots, l_n)$	the degree of a multivariate polynomial
$i = (i_1, \ldots, i_n)$	a multi-index (an n-tuple of nonnegative integers)
$\hat{l}$	$:= \max\limits_{j=1}^{n} l_j$ (maximum degree by variable)
$B_i^l : \mathbb{R}^n \to \mathbb{R}$	the ith multivariate Bernstein basis polynomial of degree l
a_i	ith coefficient of a polynomial in power form
b_i	ith coefficient of a polynomial in Bernstein form (Bernstein coefficient)
$b_i^{[l]\{p\}\mathbf{X}}$	ith degree l Bernstein coefficient of p over $\mathbf{X}$
$\mathbf{b}_i$	control point associated with the Bernstein coefficient b_i
0	$:= (0, \ldots, 0)$ (multi-index)
1	$:= (1, \ldots, 1)$ (multi-index)
J	the Jacobian
$\mathbf{\Sigma}$	the solution set of a system of (polynomial) inequalities
F	the set of feasible solutions to an optimisation problem
x^i	$:= \prod\limits_{\mu=1}^{n} x_\mu^{i_\mu}$ (a multi-power / multivariate monomial)
$\sum\limits_{i=0}^{l}$	$:= \sum\limits_{i_1=0}^{l_1} \ldots \sum\limits_{i_n=0}^{l_n}$ (a nested sum)
$\binom{l}{i}$	$:= \prod\limits_{\mu=1}^{n} \binom{l_\mu}{i_\mu}$ (the generalised binomial coefficient)

Contents

List of Figures

List of Tables

1 Introduction

The abstraction and formalisation of a real-world problem into precise mathematical terms is a fundamental process. Typically each discrete piece of information is rewritten as a mathematical statement; consequently one is presented with a set of variables and parameters, together with one or more conditions or constraints that have to be satisfied. The set of variables defines the search space — a solution typically consists of a tuple of values for these variables satisfying all the constraints, each of which represents some limitation or property that a valid solution is required to observe, a relationship typically expressed in the form of an equality or inequality, or perhaps as a set-theoretic statement involving quantifiers. Different values for parameters characterise each problem instance.

For example, consider the problem of determining the landing point of a parachutist. The solution might consist of the x and y co-ordinates of the landing location, plus t, the time taken. Parameters might include the starting position, wind direction and speed, size of parachute, weight of the parachutist, etc. Possible solutions are constrained by the laws of physics and the range of controlled movement of the parachute. One might wish to determine the solution exactly, or it might suffice merely to be satisfied that it lies within a certain 'safe' region. Alternatively one might wish to minimise the time in the air, or maximise the displacement in the x-direction, for example. In the latter case we have an optimisation problem and it is desired to compute the control inputs which correspond to the optimal outcome. Sources of uncertainty might include wind variability and measurement imprecision; this may cause the real-world solution to deviate from the mathematical solution, unless such uncertainty is explicitly included in the model.

Many such problems are highly non-trivial to solve. The number of variables and the complexity of the constraints influences the solution difficulty in a generally unforeseeable fashion. However, higher-dimensional problems (i.e. problems with many variables) are generally harder than lower-dimensional ones; problems involving nonlinear functions are typically orders of magnitude harder than those involving linear functions only. Added complexity is added if one wishes to robustly model uncertainty, for example by giving the parameters ranges of values, or by using a computation scheme such as *interval arithmetic* which explicitly tracks small uncertainties such as rounding errors.

Examples of the types of problem under consideration, restricted to $\mathbb{R}^n$, include:

- Problems involving only equations:
  - A system of n linear equations in n unknowns (variables). Such a system can be represented as a matrix coupled with a right-hand-side vector and is classically solved by Gaussian elimination; it has a single solution in all but degenerate cases.

  - A system of n nonlinear equations (e.g. polynomial equations) in n unknowns. Such a system may have zero or more unique solutions, or a continuum of solutions; it may be desired to compute them all, or only those within a certain region.

- Problems involving only inequalities:
  - A system of one or more inequalities in n unknowns. Here, the solution is a subset (either bounded or unbounded) of $\mathbb{R}^n$; robust control problems fall into this category. If bounded, the solution set may be determined exactly, or else an enclosure for it is sought. In some cases, it suffices to determine a single feasible solution.

- Problems involving both equations and inequalities:
  - A system of inequalities and equations in n unknowns defining a feasible set. It is desired to find one or more feasible solutions (or else complete a proof that no feasible solutions exist); this is a *constraint satisfaction* problem.

- Problems involving an objective function:
  - A system of linear inequalities together with a linear objective function in n unknowns. This is a *linear programming* problem, where it is desired to find the solution which, in canonical form, maximises the value of the objective function, whilst satisfying all the inequality constraints.
  - A system of equality and inequality constraints (at least some of which are nonlinear) together with an objective function in n unknowns. This is a *constrained optimisation* problem. In the canonical form, it is desired to find the (possibly unique) point in the feasible set minimising the objective function.

A wide and interesting subset of nonlinear problems are those involving polynomial functions. Many relationships and laws in real-world and abstract problems have a natural expression in polynomial form. Polynomial problems tend to be (but are not always) somewhat more tractable than those involving arbitrary functions.

In this thesis we consider techniques mostly for polynomial problems which may be broadly labelled as *enclosure methods*. Here, 'enclosure' is used to refer to the principle of searching for solutions within a specified region, as well as to the notion of partitioning solutions into isolated regions (i.e. subdivision). Somewhat related to this is the use of information on the behaviour of component functions over the edges of such regions (i.e. boundary methods). It is possible to employ concepts such as the *topological degree* of a mapping over a region, which exhibits a root-counting property, or the *Bernstein expansion* of a polynomial over a box, which exhibits a range-enclosing property. The term 'enclosure' is also often used to refer to an interval range of values.

A commonly-employed technique for the determination of solutions within a specified region of interest is the category of algorithms known as *branch-and-bound* methods (sometimes also called *branch-and-prune* or *subdivision* methods). Given a starting box of interest, a branch-and-bound algorithm recursively splits it into sub-boxes, eliminating infeasible boxes by using bounds for the range of the functions under consideration over each of them. Refinements may include proofs of non-existence of solutions to discard (prune) sub-boxes, box contraction techniques, and various strategies for subdivision, e.g. a choice of subdivision direction or a choice of subdivision point corresponding to a bisection or otherwise. At the end, it delivers a union of boxes of a suitably small size that contains all solutions to the system which lie within the given box.

In (correctly) formulating a real-world problem in mathematical terms, the goal of achieving a solution may be regarded as already half-completed. We do not stray into the issues regarding problem formulation (which are many), but consider problems that are already presented in one of the formulations listed above.

1.1 Systems of Nonlinear Equations

A common problem that arises in applied mathematics is that of solving a system of nonlinear equations for individual solutions. Such a system can be written in the form $\mathcal{F} = 0$ where

$$\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n.$$

The expected solutions $(x_1, \ldots, x_k)$, where the number of solutions k is not generally known in advance, are present in the solution space $\mathbb{R}^n$; for many problems, some or all of the solutions are in $\mathbb{C}^n$.

A problem specification can usually be reduced to a set of n component functions $f_i : \mathbb{R}^n \to \mathbb{R}$, where the f_i may be described in terms of elementary functions, composed with the usual arithmetic operators. Except in the simplest of cases, this forms a nonlinear system with a set of solutions that are highly non-trivial.

Since the scope of all nonlinear systems is so wide, a significant amount of research effort is focussed upon polynomial systems. Systems of polynomial equations are a major subset of systems of nonlinear equations and they appear in many applications, such as geometric intersection computations, chemical equilibrium problems, combustion, kinematics, and many more.

Let n polynomials p_i, $i = 1, \ldots, n$, in the real variables $x_1, \ldots, x_n$ and a box $\mathbf{X}$ in $\mathbb{R}^n$ be given. We wish to determine the set of all solutions to the equations $p_i(x) = 0$, $i = 1, \ldots, n$, within $\mathbf{X}$, viz. the set

$$\{x \in \mathbf{X} \mid p_i(x) = 0, \ \forall i = 1, \ldots, n\}. \tag{1.1}$$

There are several varied categories of methods for solving such systems, including elimination methods based on a construction of Gröbner bases and symbolic computation, continuation methods, and branch-and-bound methods. Due to the computational overheads associated with symbolic computation, elimination methods are typically unsuitable for high-dimensional problems. Continuation methods deliver all complex solutions of the system, which may be unnecessarily time-consuming for many applications in which only the solutions in a given area of interest — typically a box — are sought. The final category consists of methods which utilise a domain-splitting approach, and it is here that tools such as interval computation techniques and the Bernstein expansion may be employed.

1.2 Systems of Polynomial Inequalities

Numerous problems in control system design and analysis may be recast and reduced to systems of inequalities involving multivariate polynomials in real variables. This corresponds to the following problem:

Let $p_1, \ldots, p_m$ be n-variate polynomials and let a box $\mathbf{X}$ in $\mathbb{R}^n$ be given. We wish to determine the *solution set* $\mathbf{\Sigma}$ of the system of polynomial inequalities, given by

$$\mathbf{\Sigma} \ := \ \{x \in \mathbf{X} \ \mid \ p_i(x) > 0, \ \forall i = 1, \ldots, m\}\,. \tag{1.2}$$

A common example is the problem of determining the stability region of a family of polynomials in a given parameter box, a type of control problem. This problem can be reformulated as a system of polynomial inequalities. The structure of the solution set may be complex, for example it may be non-convex or even disconnected. A typical approach is to seek an outer enclosure for this set, either in the form of a single bounding box, or as a union of such boxes generated by a branch-and-bound algorithm.

A system of inequalities may be augmented with one or more equations, yielding a constraint satisfaction problem. Instead of an enclosure for a many-dimensional solution set, we seek a single feasible solution (although in general no assumption is made about the dimensionality of the solution set). Two main recursive search techniques for such solutions are constraint propagation and backtracking. Local search methods may also be employed, but are not guaranteed to find a solution to a feasible problem.

1.3 Global Optimisation Problems

Global optimisation problems are very widely occurring, arising naturally in application areas such as the optimisation of manufacturing and chemical processes, logistics, and finance, as well as in abstract mathematical problems such as curve fitting or sphere packing. A constrained global optimization problem may be written as follows:

$$\min_{x \in F} f(x), \tag{1.3}$$

where the set of *feasible solutions* F is defined by

$$F \ := \ \left\{ x \in S \ \left| \ \begin{array}{l} g_i(x) \leq 0 \ \text{ for } \ i = 1, \ldots, t \\ h_j(x) = 0 \ \text{ for } \ j = 1, \ldots, s \\ x \in \mathbf{X} \end{array} \right. \right\},$$

and where some $S \subseteq \mathbb{R}^n$, $\mathbf{X}$ is an axis-aligned box contained in S, and f, g_i, h_j are real-valued functions defined on S.

Among the most frequently used deterministic methods for solving such problems are branch-and-bound and interval algorithms. Other types of methods involve stochastics (e.g. Monte-Carlo sampling) or heuristics.

1.4 Outline

The thesis is divided into two main parts. In the first part we present the background material and current state of the art in greater depth and review the existing literature. The major topics of relevance are: the theory and techniques of the solution of systems of equations, inequalities, and optimisation problems; topological degree theory; Bernstein expansion; and interval arithmetic. The second part of the thesis consists of contributions in four areas, each of which is given in a separate chapter: a major investigation of a recursive topological degree computation algorithm, a branch-and-bound method for systems of polynomial equations, an improved Bernstein expansion, and affine bounding functions. This second part includes results which appear in the publications [GS01a, GS01b, GJS03a, GJS03b, GS04, GS05, GS07, GS08, Smi09]. Here follows a brief overview of each chapter:

Chapter 2: An introduction to interval analysis and arithmetic. Beginning with a brief discussion on uncertainty, we proceed to review the fundamental theory of interval arithmetic and its properties. We introduce interval enclosures and give an overview of the main categories of interval algorithms. The algorithm in Chapter 7 deals centrally with intervals and the Bernstein expansion (utilised in Chapters 8–10) is closely related to interval analysis.

Chapter 3: A treatment of the Bernstein expansion. Firstly we deal with the Bernstein basis, the Bernstein basis polynomials, and conversions to and from the Bernstein and power bases. The major properties of the Bernstein coefficients are then discussed, before proceeding to a number of important algorithms for the computation of the Bernstein coefficients. Finally, there is a brief discussion of the mean value Bernstein form and Bézier curves.

Chapter 4: An introduction to topological degree. We review the foundations of topological degree theory, in particular the Brouwer degree and its properties. There follows a review of existing known algorithms for its calculation, which are relatively few and mostly grounded upon similar ideas. The recursive method of Aberth [Abe94] is explained at some length, with an illustrative worked example, serving as the starting point for Chapter 7.

Chapter 5: A discussion on systems of polynomial equations. We firstly mention several application areas where such systems arise and give a simple example. There follows a discussion on the categorisation of solution methodology and an overview of five known techniques.

Chapter 6: A review of problems involving polynomial inequalities. We briefly cover systems of polynomial inequalities, including stability regions, and global optimisation problems, including the use of relaxations.

Chapter 7: A detailed study of a recursive algorithm for the computation of Brouwer degree. The behavioural attributes of the recursive method in practice (especially the complexity) were not previously well-understood and so we begin with some open questions. A detailed abstract analysis is followed by a number of computational examples. An estimate of the complexity, with a focus on the face subdivision process, is obtained through an unorthodox abstract complexity study, a probabilistic analysis based on a simplified geometric model. The software is exercised with a catalogue of examples, and data are obtained through the large-scale repetition of randomly-generated test problems, which support the conclusions of the complexity study. A further abstract analysis concerning an optimal face subdivision strategy is undertaken, introducing some new terminology for strategy attributes and identifying a crucial sub-problem. This material motivates the conception of a proposed new subdivision heuristic.

Chapter 8: A branch-and-bound method for the solution of systems of polynomial equations. We describe a new algorithm, which is based on the Bernstein expansion, coupled with a simple existence test which utilises interval arithmetic. The method is tested with several examples, and a couple of simple strategies for the choice of subdivision direction selection are compared. The method is then improved by the addition of a preconditioning scheme which results in a reduced complexity.

Chapter 9: A new representation for the Bernstein coefficients which offers a major computational advantage. We firstly derive some key results concerning the Bernstein coefficients of monomials, and their monotonicity. This motivates the formulation of an implicit Bernstein representation. Coupled with three tests which are proposed, this form potentially allows a much faster computation of the Bernstein enclosure. An illustrative example is given, and the scheme is tested with several example polynomials from the literature. It is seen that the complexity is much reduced for many types of sparse polynomial.

Chapter 10: Several different types of affine bounding functions for polynomials, based upon the Bernstein expansion. After a brief discussion on convex envelopes, we consider the advantageous construction of affine bounding functions, which might be utilised in a relaxation scheme within a branch-and-bound framework. Six different algorithms, some simple and some elaborate, are described. They are compared with a large catalogue of randomly-generated polynomials. Finally, there is a brief discussion on the potential use of the implicit Bernstein form, as well as the application of such affine bounding functions for polynomials.

Chapter 11: Summary and conclusions. In the final chapter we review the main results of the thesis, placing them in context, and outline possible directions for further related work.

Appendix A: Software. In the appendix we present an overview of the developed software, both for the computation of topological degree and for the computation of the Bernstein expansion and associated bounding functions.

Part I: Background and Existing Work

2 Interval Analysis

When a mathematical model involving continuous data (e.g. real numbers) is applied to a real world problem, one must typically reckon with uncertainty. The values of the parameters or variables on which computations are performed are only an approximation of their true, real-world values. These true values are either known only approximately, or else they cannot be represented in a computer exactly. Sources of such uncertainty may be categorised as follows:

- **Aleatory (irreducible, statistical) uncertainty**: Inherent variability due to fundamental 'unknowable' factors. For example, a physical quantity may vary according to environmental conditions (e.g. temperature, pressure, humidity, etc.) or as a result of manufacturing imperfection; there may be an element of risk (e.g. risk of failure or deformation) associated with a process. For practical purposes, these factors may be considered to be effectively random. Consequently, the corresponding parameters for supposedly identical problems will vary from instance to instance.

- **Epistemic (reducible, systematic) uncertainty**: Variability due to lack of knowledge about the problem. This uncertainty is often caused by measurement imprecision, due either to random errors (e.g. the limits of human observation) or systematic errors (e.g. hidden data, neglect of additional factors, faulty equipment). It is generally possible to reduce such uncertainty by improving the accuracy of a measurement technique, eliminating sources of error, or adopting a more complete model.

- **Computational uncertainty**: Variability due to the way that the data are stored and manipulated in a computer. Continuous data are typically stored as floating-point numbers, which are (very close) approximations of the true values, with a (very small, possibly zero) rounding error. When arithmetic operations are performed on two floating-point numbers, the rounding error is compounded. Where many such operations are performed in succession, there is the potential for the rounding error to accumulate in magnitude to many times the initial rounding error. Well-known examples of important failures due to rounding errors include a failure of the Patriot missile system during the first Gulf War [Ske92] and the faulty distribution of seats after a parliamentary election in the German State of Schleswig-Holstein [WW92]. S. M. Rump [Rum88] gives an example of a seemingly straightforward arithmetic expression for which even extended precision floating-point arithmetic gave a faulty result; it was also explored in [Han03]. Computational uncertainty may be considered to be a subset of epistemic uncertainty, since it is reducible, for example by the use of arbitrarily high precision floating-point arithmetic, albeit at greater computational expense.

In the context of uncertainty, an interval is closely related to the concept of a *margin of error*, which is commonly quoted in any field involving measurement (e.g. engineering, physics, statistics, weather forecasting, risk assessment, and many more). The latter is usually expressed as the radius of a confidence interval centred on a nominal value. In some cases, a quantity is implied to lie within the margin of error (i.e. within the associated confidence interval) with 100% certainty; in other cases the confidence is lower. For example, a manufacturer might *guarantee* that the length of a certain component will lie within a specified margin of error, whereas the results of an opinion poll are usually quoted with a margin of error representing a 95% confidence interval.

Classical interval analysis is predicated on the assumption that the probability distribution of a measure within an interval is flat, i.e. all values within the interval are equally likely. (In fact, interval analysis is merely a special case of fuzzy analysis where the probability distributions are flat.) It should be noted that the true probability distribution of the measure need not be flat, and typically is not flat, for interval analysis to be applicable — the type of distribution is merely disregarded. Thus interval analysis concerns itself with the extent of the range of possible values of a measure, and not the shape of the distribution of such values within the range.

Where one is aware of the existence of uncertainty in a computation scheme, it is natural to question the quality and/or veracity of the outcome. Depending on the precision to which the solution is quoted, the following questions naturally arise:

- **How likely to be correct is the solution?**
- **How accurate is the solution?**

Under an interval-based approach, these questions are answered as follows:

The solution is 100% correct within the specified range.

This is the most common model for handling uncertainty: **precision is sacrificed for accuracy**. Although interval analysis can still be performed with less than full confidence intervals, the most typical (and most powerful) use of interval analysis is with 100% confidence intervals. Amongst others, typical adjectives for this type of computation include: *guaranteed*, *reliable*, *rigorous*, and *verified* (some authors point out subtle differences in meaning between these terms). Reliable computing has obvious applications in many real-world engineering problems and safety-critical systems, where guaranteed results are highly desirable or essential.

2.1 Interval Arithmetic

The foundations of interval arithmetic began to become somewhat widely studied with the publication of R. E. Moore's seminal work [Moo66] in 1966, wherein the philosophy, basic rules, and some applications of computation with intervals were introduced. The moment of this publication is widely considered to be the beginning of modern interval arithmetic.

However a small handful of earlier interval works are known: M. Warmus [War56] and T. Sunaga [Sun58] each independently proposed computation schemes for intervals a decade earlier, around the time of Moore's own early interval papers. The first known paper including interval analysis was published as early as 1931 by R. C. Young [You31]. That numerous authors independently proposed the same basic rules of interval arithmetic is a reflection of the fact that the basic arithmetic operations on intervals are readily deducible. After the appearance of Moore's work, much further early work was performed by German researchers, most notably U. W. Kulisch [Kul08], K. L. E. Nickel [Gar10], and G. Alefeld [AH83].

Although interval arithmetic still does not enjoy widespread recognition, fields of application have expanded to include systems of equations, global optimisation, matrix methods, and differential equations, amongst others. It has achieved notable success with computer-aided proofs of some classic conjectures, e.g. [Hal05]; a good overview is presented in [Rum05]. Related schemes of arithmetic include *fuzzy arithmetic* and *affine arithmetic*. Standards for machine implementation, including an IEEE standard [IEE], are ongoing, and numerous software packages for interval arithmetic have been produced (see Subsection 2.1.8). Interval computations are becoming increasingly popular, but cannot yet be considered to be mainstream. As the technology improves, this trend should continue. Efforts to realise interval arithmetic in hardware are ongoing by Oracle (formerly Sun Microsystems) and others.

We provide here an introduction to the field of interval arithmetic; for further details see [Moo66, Moo79, AH83, RR88, Neu90, Sta95, Kul08, MKC09]. A further introduction, detailing applications of interval arithmetic, resources, and an exhaustive bibliography is available [Kea96a]; see also [Int].

2.1.1 Elementary Definitions

Definition 2.1 (Interval)**.** *An* interval $\mathbf{x} = [\underline{x}, \overline{x}]$ *where* $\underline{x} \leq \overline{x}$ *and* $\underline{x}, \overline{x} \in \mathbb{R}$ *denotes the set of real numbers* $\{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$*.*

A *degenerate* interval is one consisting of a single point, i.e. its lower and upper bounds are identical. (Strictly, a degenerate interval is a singleton set, but is often treated as equivalent to a real number, and that convention will be followed here.) Interval arithmetic is an arithmetic which operates on the set of intervals with real endpoints $\mathbb{IR}$ rather than the real numbers $\mathbb{R}$.

Definition 2.2 (Box)**.** $\mathbf{X} = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n] \in \mathbb{IR}^n$*, a Cartesian product of* n *intervals, where* $n \geq 1$*, is termed a* box*.*

2.1.2 Idealised Interval Arithmetic

We would wish to define interval operations in such a way that they yield the exact range of the corresponding real operation. For example, $[-1, 2.5] + [1, 4]$ should yield $[0, 6.5]$. In other words, an elementary operation $\odot \in \{+, -, *, \div\}$ for *idealised* interval arithmetic must obey

$$\mathbf{x} \odot \mathbf{y} \;=\; \{x \odot y \;\mid\; x \in \mathbf{x},\; y \in \mathbf{y}\}. \tag{2.1}$$

2.1.3 Operational Definitions

The above definition of idealised arithmetic is not sufficient to permit calculation on a computer. For this purpose, we require a definition for each operation in terms of the corresponding real operation:

Definition 2.3 (Interval Operations). *For* $\mathbf{x} = [\underline{x}, \overline{x}]$ *and* $\mathbf{y} = [\underline{y}, \overline{y}]$,

$$\begin{aligned}
\mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\
\mathbf{x} - \mathbf{y} &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \\
\mathbf{x} * \mathbf{y} &= [\min\{\underline{x} * \underline{y}, \underline{x} * \overline{y}, \overline{x} * \underline{y}, \overline{x} * \overline{y}\}, \max\{\underline{x} * \underline{y}, \underline{x} * \overline{y}, \overline{x} * \underline{y}, \overline{x} * \overline{y}\}], \\
\mathbf{x} \div \mathbf{y} &= \mathbf{x} * (1 \,/\, \mathbf{y}), \quad \text{where} \\
1 \,/\, \mathbf{y} &= [1 \,/\, \overline{y}, 1 \,/\, \underline{y}], \quad \text{if} \quad \overline{y} < 0 \quad \text{or} \quad \underline{y} > 0 \quad \text{(otherwise undefined).}
\end{aligned}$$

It should be clear that these operators are defined in a way such that a single operation $\mathbf{x} \odot \mathbf{y}$ will yield an exact (ideal) interval result. But what properties does the operational arithmetic satisfy when these operators are composed?

Theorem 2.1 (Algebraic Properties). *The elementary interval operations $+$ and $*$ satisfy the properties of associativity, commutativity, and the existence of a zero/one element, i.e. for all* $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{IR}$,

$$\begin{aligned}
(\mathbf{x} + \mathbf{y}) + \mathbf{z} &= \mathbf{x} + (\mathbf{y} + \mathbf{z}), & \mathbf{x} + \mathbf{y} &= \mathbf{y} + \mathbf{x}, & 0 + \mathbf{x} &= \mathbf{x}, \\
(\mathbf{x} * \mathbf{y}) * \mathbf{z} &= \mathbf{x} * (\mathbf{y} * \mathbf{z}), & \mathbf{x} * \mathbf{y} &= \mathbf{y} * \mathbf{x}, & 1 * \mathbf{x} &= \mathbf{x}.
\end{aligned}$$

Theorem 2.2 (Subdistributivity). *For all* $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{IR}$

$$\mathbf{x} * (\mathbf{y} + \mathbf{z}) \;\subseteq\; \mathbf{x} * \mathbf{y} + \mathbf{x} * \mathbf{z} \tag{2.2}$$

with equality if either $\mathbf{x} \in \mathbb{R}$, $\underline{y}, \underline{z} \geq 0$, *or* $\overline{y}, \overline{z} \leq 0$.

2.1.4 Overestimation and the Dependency Problem

As Theorem 2.2 suggests, we can very quickly arrive at simple examples of operational arithmetic with non-ideal results, either from simple compound expressions where the operators are composed, or from the presence of multiple occurences of the same variable. In such cases, outer bounds on the range of a real function are obtained.

Example 2.1. *Let* $\mathbf{x} = [-1, 1]$.

- $\mathbf{x} - \mathbf{x}$*, is computed as* $[\underline{x} - \overline{x}, \overline{x} - \underline{x}] = [-2, 2]$*, not* 0 *(although it contains* 0*). (This illustrates the fact that intervals do not have inverses.)*
- $\mathbf{x}^2$*, if implemented as* $\mathbf{x} * \mathbf{x}$*, yields* $[-1, 1]$*, which is only an outer bound for the true range,* $[0, 1]$*.*
- *Let* $f(x) = x(x + 1)$*. Then* $f(\mathbf{x}) = [-0.25, 2]$*, however* $\mathbf{f}(\mathbf{x}) = [-1, 1]([-1, 1] + 1) = [-1, 1][0, 2] = [-2, 2]$*, where* $\mathbf{f}$ *is the natural interval extension (see Subsection 2.1.7) of* f*.*

We can see that overestimation rapidly becomes a fact of life with interval arithmetic. This phenomenon is called the *dependency problem* (also *decorellation*): In general, if a variable appears more than once in a symbolic expression for a real function, the computed range will exhibit an undesirable overestimation which can rapidly become excessive, because each variable instance is treated as if it were an independent variable in its own right.

The quality of an interval method may therefore be judged not only by its efficiency and range of applicability, but also by the tightness of the solution intervals it delivers (i.e. the amount of overestimation). There is often a tradeoff of quality versus computational expense. An easy answer for a single real number result is $(-\infty, +\infty) = \mathbb{R}$, which is always valid, but not very helpful. (From a computational perspective, this is sometimes written as $[-\infty, +\infty]$, since most types of floating-point format include encodings for positive and negative infinities.)

Higher-order functions, defined in terms of elementary operations, may therefore need to be rewritten in order to minimise these inaccuracies. Where possible, multiple occurrences of the same variable should be avoided. A good example is exponentiation — the standard definition in terms of a simple recursive multiplication is not satisfactory. A suitable rewrite is given below.

2.1.5 Further Operational Definitions

These definitions apply to intervals $\mathbf{x} = [\underline{x}, \overline{x}]$ and $\mathbf{y} = [\underline{y}, \overline{y}]$.

Definition 2.4 (Exponentiation)**.**

$$\begin{aligned}\mathbf{x}^n &= [\min\{\underline{x}^n, \overline{x}^n\}, \max\{\underline{x}^n, \overline{x}^n\}], \quad \text{if} \quad \overline{x} \le 0 \quad \text{or} \quad \underline{x} \ge 0, \\ &= [\min\{\underline{x}^n, \overline{x}^n, 0\}, \max\{\underline{x}^n, \overline{x}^n\}], \quad \text{if} \quad \underline{x} < 0 < \overline{x}.\end{aligned}$$

Definition 2.5 (Infimum, Supremum)**.**

$$\begin{aligned}inf(\mathbf{x}) &= \underline{x}, \\ sup(\mathbf{x}) &= \overline{x},\end{aligned}$$

Definition 2.6 (Midpoint, Radius, Width)**.**

$$\begin{aligned} mid(\mathbf{x}) &= \frac{1}{2}(\underline{x}+\overline{x}), \\ r(\mathbf{x}) &= \frac{1}{2}(\overline{x}-\underline{x}), \\ w(\mathbf{x}) &= \overline{x}-\underline{x}. \end{aligned}$$

Definition 2.7 (Absolute Value, Magnitude, Mignitude)**.**

$$\begin{aligned} |\mathbf{x}| &= \{|x| \mid x \in \mathbf{x}\} \\ &= [\underline{x}, \overline{x}], \quad \textit{if } \underline{x} \geq 0 \\ &= [-\overline{x}, -\underline{x}], \quad \textit{if } \overline{x} \leq 0 \\ &= [0, \max\{|\underline{x}|, |\overline{x}|\}], \quad \textit{otherwise}, \\ mag(\mathbf{x}) &= \max_{x\in\mathbf{x}} |x| \;=\; \max\{|\underline{x}|, |\overline{x}|\}, \\ mig(\mathbf{x}) &= \min_{x\in\mathbf{x}} |x| \\ &= 0, \quad \textit{if } 0 \in \mathbf{x} \\ &= \min\{|\underline{x}|, |\overline{x}|\}, \quad \textit{otherwise}. \end{aligned}$$

The distance between two intervals is usually defined as the Hausdorff distance, which yields:

Definition 2.8 (Distance)**.**

$$dist(\mathbf{x}, \mathbf{y}) \;=\; \max\{|\underline{x}-\underline{y}|, |\overline{x}-\overline{y}|\}.$$

Union and intersection can be performed on intervals in the usual set-theoretic fashion. However these operators do not always yield an interval result; union either yields a single interval or the union of two disjoint intervals, intersection either yields an interval or the empty set. Instead we can define the following operators:

Definition 2.9 (Meet, Join)**.**

$$\begin{aligned} \mathbf{x} \wedge \mathbf{y} &= [\max(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})], \quad \textit{if } \max(\underline{x}, \underline{y}) \leq \min(\overline{x}, \overline{y}), \quad \textit{(meet)} \\ \mathbf{x} \vee \mathbf{y} &= [\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y})]. \quad \textit{(join)} \end{aligned}$$

For further operational definitions, including the exponential function, trigonometric functions, square root, etc., see [LTWvG01].

2.1.6 Relations on Intervals

Definition 2.10 (Interval Relations).

$$\begin{aligned}
\mathbf{x} = \mathbf{y} &\iff \underline{x} = \underline{y} \;\; \textit{and} \;\; \overline{x} = \overline{y}, \\
\mathbf{x} \subseteq \mathbf{y} &\iff \underline{x} \geq \underline{y} \;\; \textit{and} \;\; \overline{x} \leq \overline{y}, \\
\mathbf{x} \leq \mathbf{y} &\iff \underline{x} \leq \underline{y} \;\; \textit{and} \;\; \overline{x} \leq \overline{y}, \\
\mathbf{x} < \mathbf{y} &\iff \underline{x} < \underline{y} \;\; \textit{and} \;\; \overline{x} < \overline{y}.
\end{aligned}$$

2.1.7 Interval Functions

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-valued function and let $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$ be an interval-valued function. Given a box

$$\mathbf{X} \;=\; [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n],$$

the range of f (the real-valued function) on $\mathbf{X}$ is given by

$$f(\mathbf{X}) \;=\; \{f(x) \;\mid\; x \in \mathbf{X}\}.$$

Definition 2.11 (Interval Extension). *The interval-valued function* $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$ *is an interval extension of* $f : \mathbb{R}^n \to \mathbb{R}$ *if*

$$\begin{aligned}
\mathbf{f}(x) &= f(x), \quad \forall x \in \mathbb{R}^n, \quad \textit{and} \\
\mathbf{f}(\mathbf{X}) &\supseteq f(\mathbf{X}), \quad \forall \mathbf{X} \in \mathbb{IR}^n.
\end{aligned}$$

Interval extensions (sometimes also known as *inclusion functions*, cf. [RR88, Section 2.6]) are used to bound the range of a real function.

Definition 2.12 (Inclusion Isotone). *Let* $\mathbf{f} : \mathbb{IR}^n \to \mathbb{IR}$. $\mathbf{f}$ *is inclusion isotone if*

$$\mathbf{X}_1 \subseteq \mathbf{X}_2 \implies \mathbf{f}(\mathbf{X}_1) \subseteq \mathbf{f}(\mathbf{X}_2), \;\; \forall \mathbf{X}_1, \mathbf{X}_2 \in \mathbb{IR}^n.$$

This property is often also called *inclusion monotone.* Where an interval-valued function is inclusion isotone, equality with a real-valued function on all degenerate intervals (i.e. point values) suffices for it to be an interval extension:

Theorem 2.3. *If* $\mathbf{f}$ *is inclusion isotone and*

$$\mathbf{f}(x) \;=\; f(x), \quad \forall x \in \mathbb{R}^n,$$

then $\mathbf{f}$ *is an interval extension of* f.

Moore's fundamental theorem of interval arithmetic [Moo66, Theorem 3.1] states that an interval-valued function which is a rational expression of interval variables is inclusion isotone (over the valid domain of the arithmetic operations appearing therein). The *natural interval extension* (see Definition 2.13) of a real-valued function is therefore inclusion isotone.

Definition 2.13 (Natural Interval Extension [Moo66])**.** *The natural interval extension of a real-valued function $f : \mathbb{R}^n \to \mathbb{R}$ is the interval extension obtained by replacing the real variables and operations in the function expression for f by the corresponding interval variables and operations.*

Note that a precise mathematical definition for the natural interval extension requires recursion and an unambiguous lexical notion of function expression; it would thus be more accurate to consider the natural interval extension of a particular function *expression.* The natural interval extension provides an overestimation for the range of f, except in the case where each variable appears at most once in the rational expression of the extension, in which case it provides the actual range.

In practice, where f is a non-trivial composition of operational elementary and higher-order functions, the natural interval extension yields a non-ideal wider interval (which nevertheless contains the true range of f over the box). As illustrated by Example 2.1, and according to the principle of decorellation, the tightness of the bounds depends largely upon the *expression* of the function. In this regard, it is highly desirable to first simplify it as much as possible, reducing the number of interval operations and in particular the number of occurrences of each variable, where possible. This process itself is non-trivial and is a topic for research. In extreme cases, a badly-presented interval function may yield $(-\infty, +\infty)$ for all but the narrowest of arguments. Some types of rewriting which may be advantageous for regular arithmetic are less suitable for the purpose of interval computations. However it is fairly straightforward to obtain rigorous enclosures for functions such as trigonometric functions and the exponential function, permitting a wide range of interval functions to be computed effectively.

2.1.8 Computer Implementation

Since no mainstream hardware implementation of interval arithmetic exists (to date), it must generally be realised in software, based upon a hardware implementation of floating-point arithmetic. Since, as illustrated earlier, floating-point arithmetic is subject to computational uncertainty, achieving guaranteed results is therefore not trivial. Fortunately, modern standards for floating-point arithmetic include encodings for positive and negative infinities and NaNs ('Not a Number'; used to signal an undefined or unrepresentable value), as well as different modes for rounding, which can be used to support verified computation.

Broadly speaking, two steps are necessary:

- Any floating-point value is stored as the smallest 'safe' interval which is guaranteed to enclose it. This is sometimes called the *hull approximation*; it is usually an interval of machine-precision width.
- Outward rounding is used throughout: whenever an arithmetic operation is performed, the floating-point operations on the endpoints are performed using the appropriate *directed rounding* mode.

With such an implementation, erroneous results due to rounding errors associated with traditional floating-point arithmetic may be avoided and we achieve result interval(s) that are guaranteed to contain the true function range, irrespective of machine rounding.

A more detailed specification of interval arithmetic implementation may be found in, e.g., [Sta95, LTWvG01].

Software Libraries

Numerous interval arithmetic libraries for the programming language C++ exist, as well as one or two others for other languages. The earliest libraries to achieve recognition were *Pascal-XSC* [KK$^+$92] and *C-XSC* [XSCb] (a brief history of their development is given in [XSCa]), as well as *PROFIL/BIAS* (Programmer's Runtime Optimised Fast Interval Library, Basic Interval Arithmetic Subroutines) [Knu94, PRO]. In this thesis, the library *filib++* (Fast Interval LIBrary for C++) [LTWvG01] was used. Other C++ libraries include *Boost* [BMP03] and *Gaol* (not Just Another Interval Library) [GAO]. A number of commercial mathematical softwares also include interval arithmetic components or addons, for example *INTLAB* [Rum99] is a toolbox for the *MATLAB* system; in particular it has fast routines for interval vector and matrix manipulation.

2.2 Interval Enclosures

A valid interval extension yields a guaranteed enclosure for the range of a real-valued function over an interval, but the tightness of the resulting interval (i.e. the quality or usefulness of this enclosure) may vary, depending on its construction. Firstly it should be noted that the natural interval extension (see Definition 2.13, i.e. the straightforward application of interval arithmetic) can often deliver bounds which are tighter and/or quicker to compute than those obtained by traditional means with floating-point arithmetic, such as the use of Lipschitz conditions. Nevertheless, as the interval width is shrunk, the natural interval extension converges only linearly to the true enclosure. However it can be improved upon considerably and the comparison of different types of interval enclosures is a fruitful topic for research. In many cases, there is a tradeoff between the tightness of an enclosure and its computational complexity.

One of the most fundamental interval enclosures is the *mean value form* (or mean value interval extension), cf. [RR88, Neu90], which exhibits quadratic convergence to the true enclosure with shrinking interval width. In the univariate case, it derives from the mean value theorem, which states that if a function f is continuous over an interval $[a,b]$ and differentiable over (a,b) then there exists a point $c \in (a,b)$ for which

$$f'(c) \;=\; \frac{f(b)-f(a)}{b-a}. \tag{2.3}$$

Now let $x, \tilde{x} \in [a,b]$. Applying (2.3) and rearranging yields

$$f(x) \;=\; f(\tilde{x}) + f'(\xi)(x-\tilde{x}), \tag{2.4}$$

where ξ lies between x and $\tilde{x}$. Therefore given an interval $\mathbf{x}$ we must have for all $x, \tilde{x} \in \mathbf{x}$

$$f(x) \;\in\; f(\tilde{x}) + f'(\mathbf{x})(x - \tilde{x}). \tag{2.5}$$

Therefore

$$f(\mathbf{x}) \;\subseteq\; f(\tilde{x}) + f'(\mathbf{x})(\mathbf{x} - \tilde{x}). \tag{2.6}$$

One typically chooses $\tilde{x} := mid(\mathbf{x})$. Here, an interval extension for f' is required; it is commonly the case that it is tighter than that for f and the mean value form yields a tighter enclosure than the natural interval extension.

A detailed treatment of other types of centred form, including variants on the mean value form, types of Taylor form (based on the Taylor expansion), as well as interval enclosures specific to polynomials, such as the Horner form (see, e.g., [PS00]) and Bernstein form (see Section 3.1.2) is given in [Sta95], with a detailed comparison. Centred forms specific to polynomials are considered in [Rok81]. For some classes of random polynomials, it is demonstrated in [Sta95] that the Horner form and simple variants of the mean value form deliver relatively wide enclosures, which are improved upon by more sophisticated variants, whereas the Bernstein form generally gives the tightest enclosures.

2.3 Interval Algorithms

Due to the dependency problem, a straightforward adaption of an algorithm designed for floating-point arithmetic is usually unsuitable, typically delivering excessively wide result intervals. For example, an unmodified interval Gaussian elimination for the solution of a system of linear equations with interval coefficients will generally quickly fail due to division by the interval values for pivots containing zero, cf. Subsection 2.3.3. In general, therefore, floating-point algorithms have to be adapted in order to minimise the effect of the dependency problem. In some cases, completely new algorithms have to be devised.

2.3.1 Branch-and-Bound Methods

Branch-and-bound algorithms are a general scheme most often employed for the solution of optimisation problems, but also for systems of nonlinear equations and other types of problems. Required is a starting finite search space — in most formulations, a box — which is successively subdivided into smaller sub-regions until they can be excluded from the search. The search space is structured in the form of a tree and the sub-regions are usually (but not always) defined as boxes (see Definition 2.2).

It should be noted that the branch-and-bound method is not an interval method per se, however in the common case where boxes and sub-boxes are used, it naturally lends itself to interval approaches, which are often (but by no means always) employed for the bounding step (see below).

The first branch-and-bound methods, not utilising interval analysis, appeared the in 1960s; the first interval branch-and-bound method was the Moore-Skelboe algorithm [Moo76], also described in [RR88]. Thereafter, interval branch-and-bound methods were further developed by H. Ratschek and J. Rokne [RR88] and E. R. Hansen [Han80, Han03], amongst others.

There are two key steps to a branch-and-bound approach:

- A *subdivision* (or branching, splitting) step, which partitions the current (sub-)region under investigation into two (or more) smaller sub-regions. Typically, such a region consists of an n-dimensional box, and a subdivision (often a bisection) is performed in one direction.
- A *bounding* (or pruning) step, where sub-regions are safely discarded as candidates for containing a solution. The character of this step depends on whether a system of equations or an optimisation problem is being solved. In the former case, bounds on the ranges of the component functions may be computed; where one such function can be guaranteed to exclude zero, the sub-region may be discarded. In the latter case, which is canonically presented as a minimisation problem, the sub-region may likewise be tested for feasibility, but a lower bound for the objective function is also computed. This is compared against a global variable which records the minimum upper bound for sub-regions so far; where the lower bound for a sub-region exceeds this, it can safely be discarded. This step is also called 'pruning', because branches of the search tree are removed.

Also required is a search strategy for the order in which sub-regions are processed and a termination criterion which stipulates when sub-regions become satisfactorily small. At the end, zero or more sub-regions of this small size remain, which enclose the solution set (in the case of a system of equations) or contain the minimiser and provide an upper bound on the corresponding value of the objective function (in the case of an optimisation problem).

Heuristics and tests may be employed to accelerate the algorithm, by minimising the total number of sub-regions to be processed [RR88, Section 3.11]. These may include tree search strategies, heuristics for selection of subdivision direction, e.g. [RC95], monotonicity tests, and box contraction techniques, e.g. [VHMK97].

2.3.2 Interval Newton Methods

The interval Newton method is obtained by replacing the real (floating-point) variables in the standard Newton iteration by intervals. It is discussed in further detail in Subsection 5.3.3.

2.3.3 Matrix Methods

An interval matrix is one whose entries are intervals, instead of real numbers. Repetetive operations on such matrix entries or rows, where each interval variable often appears multiple times, make them very susceptible to the dependency problem. A common example is the classic Gaussian elimination, used to solve a system of linear equations or find a matrix inverse; a naive interval version of this method will fail for all but the simplest of problems, with the excessive widths of the entries used as pivots rapidly causing a division by zero. Such methods therefore need to be adapted in order to circumvent or mitigate these problems, e.g. with a scheme for the tightening of interval pivots [Gar09].

2.3.4 Other Interval Methods

Many other interval methods for a number of different application areas exist. A good overview is given in [Int, Kea08] and a few are listed below:

- Taylor models with interval arithmetic, commonly used for the verified solution of ordinary differential equations,
- Methods for automated theorem proving (a notable example is a proof of the Kepler conjecture, concerning sphere packing in three dimensions, by T. Hales [Hal05]; see [Rum05] for further examples),
- Constraint propagation, applied to global optimisation or constraint satisfaction problems, e.g. [Gra07],
- Various methods for the solution of finite element truss structure models with uncertain parameters, e.g. [Zha05, GPS12].

3 Bernstein Expansion

Bernstein expansion refers to the process of rewriting a polynomial in *Bernstein form*, a procedure which can be employed in function approximation and bounding the ranges of polynomials. A polynomial in Bernstein form is expressed as a linear combination of Bernstein basis polynomials. These polynomials were first defined by S. N. Bernstein in 1912 [Ber12] in a constructive proof of the Weierstrass approximation theorem, which states that a continuous function may be uniformly approximated over a finite interval arbitrarily closely by a polynomial.

In the case of the real numbers $\mathbb{R}$ the theorem may be stated as follows:

Theorem 3.1 (Weierstrass Approximation Theorem)**.** *Let f be a continuous real-valued function defined over an interval $[a, b]$. For any $\varepsilon > 0$ there exists a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that*

$$|f(x) - p(x)| \ < \ \varepsilon, \ \ \forall x \in [a, b].$$

Bernstein explicitly provided a sequence of polynomials which converge uniformly to a given continuous function over the unit interval $\mathbf{I} = [0, 1]$. It should firstly be observed that we can consider the unit interval without loss of generality. It is readily apparent that an affine transformation of a polynomial to or from $\mathbf{I}$ and a non-degenerate interval $[a, b]$ yields a polynomial of the same degree and that the same transformation preserves the continuity of an arbitrary function.

Given a function f which is continuous over $\mathbf{I}$, for a given degree l Bernstein's approximation is

$$f_l(x) \ = \ f(0)(1-x)^l + lf(\frac{1}{l})x(1-x)^{l-1} + \ldots + \binom{l}{i} f(\frac{i}{l})x^i(1-x)^{l-i} + \ldots + f(1)x^l. \quad (3.1)$$

As $l \to \infty$, $f_l(x)$ tends uniformly to $f(x)$ over $\mathbf{I}$. The proof [Ber12] relies upon the observation that $f_l(x)$ (3.1) is a weighted average of $f(0), \ldots, f(\frac{i}{l}), \ldots, f(1)$. In particular, as l becomes large, only those terms for which $\frac{i}{l} \approx x$ make a significant contribution, since the values of the weights depend upon x. The result is obtained by equating this to a probability distribution and employing a bound provided by Chebyshev's inequality.

It can be seen that each of the weights (or multipliers) for the f values appearing in (3.1) is a degree l polynomial in x; these are the Bernstein basis polynomials and f_l is a degree l polynomial presented in Bernstein form.

There is an early monograph on Bernstein polynomials [Lor53]; thereafter the Bernstein expansion was first applied to the range of univariate polynomials in [CS66] and then [Riv70] and [Rok77]; an extension to the multivariate case appears in [Gar86]. More recent treatment appears in [NA07], [BCR08] and [RN09], and a comprehensive survey paper has just been completed [Far12]. An alternative explanation of the theory of Bernstein polynomials is presented in [Zum08, Chapter 4].

3.1 Fundamentals

In this section the fundamental theory of Bernstein polynomials and Bernstein expansion is introduced.

3.1.1 Bernstein Basis Polynomials

The ith Bernstein basis polynomial of degree l is given by

$$B_i^l(x) \;=\; \binom{l}{i} x^i (1-x)^{l-i}, \quad i = 0, \ldots, l. \tag{3.2}$$

Where required, we shall adopt the convention that $B_i^l(x) = 0$ for all x if $i < 0$ or $i > l$. For l up to and including 4, the Bernstein basis polynomials are given explicitly in Table 3.1 and, for $l = 4$, depicted in Figure 3.1.

l	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
0	1				
1	$1 - x$	x			
2	$1 - 2x + x^2$	$2x - 2x^2$	x^2		
3	$1 - 3x + 3x^2 - x^3$	$3x - 6x^2 + 3x^3$	$3x^2 - 3x^3$	x^3	
4	$1 - 4x + 6x^2 - 4x^3 + x^4$	$4x - 12x^2 + 12x^3 - 4x^4$	$6x^2 - 12x^3 + 6x^4$	$4x^3 - 4x^4$	x^4

Table 3.1: The Bernstein basis polynomials $B_i^l(x)$ for l up to 4.

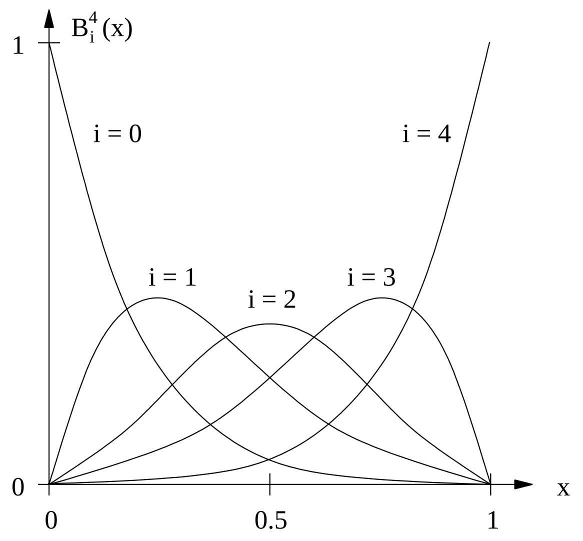


Figure 3.1: Graphs of the Bernstein basis polynomials $B_i^l(x)$ for l equal to 4.

Here, the term ‘Bernstein polynomial’ is used to mean ‘Bernstein basis polynomial’, and *not* a polynomial which is presented in Bernstein form (see Subsection 3.1.2). Some authors use this latter meaning, but this would appear to be redundant since *every* polynomial is a Bernstein polynomial in this sense — any polynomial can be rewritten in Bernstein form, as stated below.

Basic Properties

It can readily be seen that the Bernstein basis polynomials satisfy the following properties for all $l \in \mathbb{N}$, $i = 0, \ldots, l$:

- $B_i^l(x) \geq 0, \quad \forall x \in \mathbf{I}$.
- The set of Bernstein polynomials of degree l form a partition of unity, i.e.

$$\sum_{i=0}^{l} B_i^l(x) \ = \ 1, \quad \forall x \in \mathbb{R}.$$

- $B_0^l(0) = 1$; $B_i^l(0) = 0$, if $i > 0$.
- $B_l^l(1) = 1$; $B_i^l(0) = 0$, if $i < l$.
- $B_i^l(x) = B_{l-i}^l(1-x), \quad \forall x \in \mathbb{R}$.
- The Bernstein polynomials of degree l form a basis for the vector space of polynomials of degree less than or equal to l. Thus a Bernstein representation exists for every polynomial, and the coefficients are defined in Section 3.1.3.

Degree Elevation and Reduction Properties

- A Bernstein polynomial of degree l can be generated recursively from lower-degree polynomials:

$$B_i^l(x) \ = \ xB_{i-1}^{l-1}(x) + (1-x)B_i^{l-1}(x). \tag{3.3}$$

This can be proven using the recursive relation for binomial coefficients:

$$\begin{aligned}
B_i^l(x) &= \binom{l}{i} x^i (1-x)^{l-i} \\
&= \left[\binom{l-1}{i-1} + \binom{l-1}{i}\right] x^i (1-x)^{l-i} \\
&= \binom{l-1}{i-1} x^i (1-x)^{l-i} + \binom{l-1}{i} x^i (1-x)^{l-i} \\
&= x\binom{l-1}{i-1} x^{i-1} (1-x)^{(l-1)-(i-1)} + (1-x)\binom{l-1}{i} x^i (1-x)^{(l-1)-i} \\
&= xB_{i-1}^{l-1}(x) + (1-x)B_i^{l-1}(x). \quad \square
\end{aligned}$$

- A Bernstein polynomial of degree l can also be expressed as a linear combination of two Bernstein polynomials of degree $l+1$:
$$B_i^l(x) \;=\; \frac{l+1-i}{l+1} B_i^{l+1}(x) + \frac{i+1}{l+1} B_{i+1}^{l+1}(x). \tag{3.4}$$
This is proven by combining the following two results:

- A Bernstein polynomial of degree l can further be related to a single Bernstein polynomial of degree $l+1$ as follows:
$$\begin{aligned}(1-x)B_i^l(x) &= \binom{l}{i} x^i (1-x)^{l+1-i}\\ &= \frac{\binom{l}{i}}{\binom{l+1}{i}} \binom{l+1}{i} x^i (1-x)^{l+1-i}\\ &= \frac{l+1-i}{l+1} B_i^{l+1}(x);\end{aligned}$$
also
$$\begin{aligned}xB_i^l(x) &= \binom{l}{i} x^{i+1} (1-x)^{l-i}\\ &= \frac{\binom{l}{i}}{\binom{l+1}{i+1}} \binom{l+1}{i+1} x^{i+1} (1-x)^{(l+1)-(i+1)}\\ &= \frac{i+1}{l+1} B_{i+1}^{l+1}(x).\end{aligned}$$

- By repeated application of (3.4), a Bernstein polynomial of degree l can be expressed as a linear combination of Bernstein polynomials of degree $l+r$, where $r \in \mathbb{N}$:
$$B_i^l(x) \;=\; \sum_{j=i}^{i+r} \frac{\binom{l}{i}\binom{r}{j-i}}{\binom{l+r}{j}} B_j^{l+r}(x). \tag{3.5}$$

3.1.2 Bernstein Form

Here we begin to make use of the abbreviated notation for multipowers and vectors. If n is set to 1, then we have the univariate case. Let us suppose that we have an n-variate polynomial p presented in the usual power form (i.e. in the power basis) as
$$p(x) \;=\; \sum_{i=0}^{l} a_i x^i, \quad x = (x_1, \ldots, x_n). \tag{3.6}$$
It may be rewritten in Bernstein form (i.e. in the Bernstein basis) over $\mathbf{I} = [0,1]^n$ as
$$p(x) \;=\; \sum_{i=0}^{l} b_i B_i^l(x), \tag{3.7}$$

where B_i^l is the ith Bernstein basis polynomial of degree l (3.8) and b_i are the so-called *Bernstein coefficients*. The derivation of these coefficients is given below in Subsection 3.1.3. In the n-dimensional case, the ith Bernstein polynomial of degree $l = (l_1, \ldots, l_n)$ is given as a straightforward product of univariate Bernstein polynomials (3.2) as follows:

$$B_i^l(x) \;=\; B_{i_1}^{l_1}(x_1) B_{i_2}^{l_2}(x_2) \cdot \ldots \cdot B_{i_n}^{l_n}(x_n). \tag{3.8}$$

We may allow that the degree of p is given by some r, where $r < l$. In this case, the formulae herein remain in force with the convention that $a_j = 0$ if $j > r$.

3.1.3 Basis Conversion

Here we derive formulae for converting between the coefficients $\{a_i\}$ of a polynomial in the power basis and the coefficients $\{b_i\}$ in the Bernstein basis.

Conversion From Power Basis to Bernstein Basis

Theorem 3.2. *Let p be a multivariate polynomial in power form given as in (3.6). Then its coefficients in Bernstein form (its Bernstein coefficients) are given by*

$$b_i \;=\; \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{l}{j}} a_j, \quad 0 \le i \le l. \tag{3.9}$$

Proof:

$$\begin{aligned}
p(x) &= \sum_{j=0}^{l} a_j x^j \;=\; \sum_{j=0}^{l} a_j x^j \left(x + (1-x)\right)^{l-j} \\
&= \sum_{j=0}^{l} a_j x^j \sum_{k=0}^{l-j} \binom{l-j}{k} x^{(l-j)-k} (1-x)^k \\
&= \sum_{j=0}^{l} \sum_{k=0}^{l-j} a_j \binom{l-j}{k} x^{l-k} (1-x)^k \\
&= \sum_{k=0}^{l} \sum_{j=0}^{l-k} a_j \binom{l-j}{k} x^{l-k} (1-x)^k \\
&= \sum_{i=0}^{l} \sum_{j=0}^{i} a_j \binom{l-j}{l-i} x^i (1-x)^{l-i} \qquad \text{(by setting } i+k=l\text{)} \\
&= \sum_{i=0}^{l} \sum_{j=0}^{i} a_j \frac{\binom{l-j}{l-i}}{\binom{l}{i}} \binom{l}{i} x^i (1-x)^{l-i} \\
&= \sum_{i=0}^{l} \left\{ \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{l}{j}} a_j \right\} B_i^l(x). \quad \square
\end{aligned}$$

Conversion From Bernstein Basis to Power Basis

Theorem 3.3. *Let p be a multivariate polynomial in Bernstein form given as in (3.7). Then its coefficients in power form are given by*

$$a_i = \sum_{j=0}^{i} (-1)^{i-j} \binom{l}{i} \binom{i}{j} b_j, \quad 0 \leq i \leq l. \tag{3.10}$$

Proof:

$$\begin{aligned}
p(x) &= \sum_{j=0}^{l} b_j B_j^l(x) \\
&= \sum_{j=0}^{l} b_j \binom{l}{j} x^j (1-x)^{l-j} \\
&= \sum_{j=0}^{l} b_j \binom{l}{j} x^j \sum_{k=0}^{l-j} \binom{l-j}{k} (-1)^k x^k \\
&= \sum_{j=0}^{l} b_j \binom{l}{j} x^j \sum_{i=j}^{l} \binom{l-j}{i-j} (-1)^{i-j} x^{i-j} \qquad \text{(by setting } i-j=k\text{)} \\
&= \sum_{j=0}^{l} \sum_{i=j}^{l} b_j \binom{l}{j} \binom{l-j}{i-j} (-1)^{i-j} x^i \\
&= \sum_{j=0}^{l} \sum_{i=j}^{l} b_j \binom{l}{i} \binom{i}{j} (-1)^{i-j} x^i \\
&= \sum_{i=0}^{l} \left\{ \sum_{j=0}^{i} (-1)^{i-j} \binom{l}{i} \binom{i}{j} b_j \right\} x^i. \quad \square
\end{aligned}$$

Example 3.1. *Let $p(x_1, x_2) = 2x_1^3 - \frac{1}{2}x_1^2 x_2 + 7x_2^2 - 4x_1 x_2 + 3$. Its representation in Bernstein form is $p(x_1, x_2) = \sum_{i=0}^{l} b_i B_i^l(x_1, x_2)$, where the degree $l = (3, 2)$ and the Bernstein coefficients $\{b_i\}$ over $[0, 1]^2$ calculated according to (3.9) are given in Table 3.2.*

	$i_1 = 0$	$i_1 = 1$	$i_2 = 2$	$i_3 = 3$
$i_2 = 0$	3	3	3	5
$i_2 = 1$	3	$\frac{7}{3}$	$\frac{19}{12}$	$\frac{11}{4}$
$i_2 = 2$	10	$\frac{26}{3}$	$\frac{43}{6}$	$\frac{15}{2}$

Table 3.2: Bernstein coefficients b_i of $p(x_1, x_2) = 2x_1^3 - \frac{1}{2}x_1^2 x_2 + 7x_2^2 - 4x_1 x_2 + 3$.

3.1.4 Generalised Bernstein Form

In many cases it is desired to calculate the Bernstein expansion of a polynomial over a general non-degenerate n-dimensional box

$$\mathbf{X} \;=\; [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]. \tag{3.11}$$

It is possible to firstly apply the affine transformation which transforms the unit box **I** to **X** and then apply (3.9), using the coefficients of the transformed polynomial. However it is sometimes useful to consider a direct calculation. Here, the ith Bernstein polynomial of degree $l = (l_1, \ldots, l_n)$ over **X** may be written as

$$B_i^l(x) \;=\; \binom{l}{i} \frac{(x-\underline{x})^i (\overline{x}-x)^{l-i}}{(\overline{x}-\underline{x})^l}, \quad 0 \leq i \leq l. \tag{3.12}$$

The Bernstein coefficients b_i of a polynomial p (3.6) of degree l over **X** (3.11) are given by

$$b_i \;=\; \sum_{j=0}^{i} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x}-\underline{x})^j \sum_{k=j}^{l} \binom{k}{j} \underline{x}^{k-j} a_k, \quad 0 \leq i \leq l. \tag{3.13}$$

3.2 Properties of the Bernstein Coefficients

From the point of view of bounding the ranges of polynomials, the Bernstein coefficients (3.9, 3.13) exhibit a number of important and useful properties, encapsulating the behaviour and properties of a polynomial over a box of interest. These coefficients may be collected in an n-dimensional array (i.e. a tensor); in the field of computer-aided geometric design such a construct is typically labelled as a *patch*. For simplicity, we will refer to the tensor of the Bernstein coefficients hereafter as an 'array'.

Where the general case is considered, we shall assume that we have an n-dimensional polynomial p (3.6) of degree $l = (l_1, \ldots, l_n)$ over a box **X** (3.11). In this case the number of coefficients appearing in this array is

$$\#\{b_i\} \;=\; \prod_{i=1}^{n} (l_i + 1). \tag{3.14}$$

3.2.1 Vertex Values

It is readily apparent that the 2^n Bernstein coefficients occurring at a vertex of the array are identical to the values attained by p at the corresponding vertices of **X**. In the univariate case over the unit box:

$$b_0 \;=\; a_0 \;=\; p(0), \tag{3.15}$$

$$b_l \;=\; \sum_{i=0}^{l} a_i \;=\; p(1). \tag{3.16}$$

In the general case, if $i_j \in \{0, l_j\}$ $\forall j = 1, \ldots, n$, then $b_i = p(v)$ where

$$v_j = \begin{cases} \underline{x}_j & \text{if} \quad i_j = 0, \\ \overline{x}_j & \text{if} \quad i_j = l_j, \end{cases} \qquad \forall j = 1, \ldots, n.$$

3.2.2 Face Values

Lemma 3.1. *Let p be an n-variate polynomial (3.6) of degree $l = (l_1, \ldots, l_n)$ over a box* $\mathbf{X}$ *(3.11). Then the Bernstein coefficients of p over the m-dimensional faces of* $\mathbf{X}$*, where $0 \leq m \leq l-1$, are the same as the coefficients located at the corresponding m-dimensional faces of the array of Bernstein coefficients of p over* $\mathbf{X}$*.*

Proof (see also [GS01b]): We consider the case of the unit box $\mathbf{I} = [0,1]^n$ without loss of generality. It suffices to prove the statement for $m = l-1$; the statement for smaller m then follows by repeated application. Here we indicate by a subscript (k) that the quantity under consideration is taken without the contribution of the kth component, e.g. $x_{(k)} = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$. The ith Bernstein coefficient of

$$p(x_1, \ldots, x_{k-1}, 0, x_{k+1}, \ldots, x_n) = \sum_{i \leq l, i_k = 0} a_i (x^i)_{(k)},$$

considered as a polynomial in $n-1$ variables, is given by

$$\sum_{j \leq i, j_k = 0} \frac{\binom{i}{j}_{(k)}}{\binom{l}{j}_{(k)}} a_j,$$

which coincides with $b_{i_1 \ldots i_{k-1}\, 0\, i_{k+1} \ldots i_n}$. Similarly, the ith Bernstein coefficient of

$$p(x_1, \ldots, x_{k-1}, 1, x_{k+1}, \ldots, x_n) = \sum_{i_{(k)} \leq l_{(k)}} \left\{ \sum_{i_k = 0}^{l_k} a_i \right\} (x^i)_{(k)},$$

is given by

$$\sum_{j_{(k)} \leq i_{(k)}} \frac{\binom{i}{j}_{(k)}}{\binom{l}{j}_{(k)}} \sum_{j_k = 0}^{l_k} a_j,$$

which coincides with

$$b_{i_1 \ldots i_{k-1}\, l_k\, i_{k+1} \ldots i_n} = \sum_{j \leq i, i_k = l_k} \frac{\binom{i}{j}}{\binom{l}{j}} a_j. \quad \square$$

An application of Lemma 3.1 for bounding the range of a polynomial over an edge of a box was given in [ZG98, Edge Lemma].

3.2.3 Linearity

Let $p(x) = \alpha p_1(x) + \beta p_2(x)$, where p_1 and p_2 are polynomials and l is the degree of p. Then

$$b_i \;=\; \alpha b_i^{\{p_1\}} + \beta b_i^{\{p_2\}} \quad \forall\, 0 \le i \le l,$$

where $b_i^{\{p_1\}}$ and $b_i^{\{p_2\}}$ are the ith coefficients of the degree l Bernstein expansions of p_1 and p_2, respectively.

3.2.4 Range Enclosure

The essential property of the Bernstein expansion, given by Cargo and Shisha [CS66], is that the range of p over $\mathbf{X}$ is contained within the interval spanned by the minimum and maximum Bernstein coefficients (which is called the *Bernstein enclosure*):

$$\min_i\{b_i\} \;\le\; p(x) \;\le\; \max_i\{b_i\}, \quad x \in \mathbf{X}. \tag{3.17}$$

The proof follows readily by firstly applying an affine transformation to the unit box $\mathbf{I}$ and observing, as in [Riv70], from (3.7) that, for all $x \in \mathbf{I}$, $p(x)$ is a convex combination of $b_0, \ldots, b_l$.

By computing all the Bernstein coefficients, one therefore obtains bounds on the range of p over $\mathbf{X}$; this is central to the application of the Bernstein expansion in enclosure methods. As discussed in Section 2.2, it is known [Sta95] that these bounds are in general tighter than those given by interval arithmetic and many centered forms. However it should be noted that the computational effort of generating all the Bernstein coefficients is often much greater than that of these alternatives.

3.2.5 Sharpness

The lower bound on the range of p over $\mathbf{X}$ provided by the minimum Bernstein coefficient is *sharp*, i.e. there is no underestimation, if and only if this coefficient occurs at a vertex of $\mathbf{X}$. Likewise, the upper bound provided by the maximum Bernstein coefficient is sharp (in this case, there is no overestimation), if and only if the coefficient occurs at a vertex of $\mathbf{X}$. If both bounds are sharp, then the Bernstein enclosure provides the exact range; this property allows one to easily test whether or not this is the case.

3.2.6 Convex Hull

This property is a generalisation of the range enclosing property. We firstly need to define the *control points* associated with the Bernstein coefficients, and then we shall consider their convex hull.

Definition 3.1 (Control Points)**.** *Given the degree l Bernstein expansion (i.e. the Bernstein coefficients) of an n-dimensional polynomial p over the unit box $\mathbf{I} = [0,1]^n$, the control point* *associated with the ith Bernstein coefficient* b_i *is the point* $\mathbf{b}_i \in \mathbb{R}^{n+1}$ *given by*

$$\mathbf{b}_i := \left(\frac{i_1}{l_1}, \ldots, \frac{i_n}{l_n}, b_i\right). \tag{3.18}$$

The abscissae of the control points thus form a uniform grid over the box and the ordinates are equal to the values of the Bernstein coefficients.

Theorem 3.4 (Convex Hull Property)**.** *The graph of* p *over* $\mathbf{I}$ *is contained within the convex hull of the control points derived from the Bernstein coefficients, i.e.*

$$\{(x_1, \ldots, x_n, p(x)) \mid x \in \mathbf{I}\} \subseteq \mathit{conv}\,\{\mathbf{b}_i \mid 0 \le i \le l\}. \tag{3.19}$$

Proof: Consider the degree l Bernstein expansion of the identity function $id_j(x) := x_j$. From (3.9) we have that $b_i^{\{id_j\}} = \frac{i_j}{l_j}$ and therefore

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \\ p(x) \end{pmatrix} = \sum_{i=0}^{l} \begin{pmatrix} \frac{i_1}{l_1} \\ \vdots \\ \frac{i_n}{l_n} \\ b_i \end{pmatrix} B_i^l(x). \tag{3.20}$$

For all $x \in \mathbf{I}$, $(x_1, \ldots, x_n, p(x))$ is thus a convex combination of $\mathbf{b}_0, \ldots, \mathbf{b}_l$ and is therefore contained within their convex hull. □

Figure 3.2 illustrates the convex hull property for a univariate polynomial of degree 5 over the unit interval. The property holds in the case of a general box $\mathbf{X}$ by simply adjusting the control points accordingly so that they form a uniform grid over $\mathbf{X}$.

3.2.7 Inclusion Isotonicity

The Bernstein form is inclusion isotone (see Definition 2.12), i.e. if the interval $\mathbf{X}$ is shrunk to a smaller interval then the Bernstein enclosure shrinks, too. This was proven for the univariate case in [HS95a]. Here we give a shorter proof for the multivariate case and an extension to show that the convex hull of the control points associated with the Bernstein coefficients is also inclusion isotone [GJS03a]. For simplicity we consider the case of the unit interval $\mathbf{I} = [0, 1]$.

Theorem 3.5 (Inclusion Isotonicity of the Convex Hull [GJS03a])**.** *The convex hull of the control points associated with the Bernstein coefficients of a univariate polynomial is inclusion isotone.*

Proof: Let p be a degree l univariate polynomial, with Bernstein coefficients $b_0, \ldots, b_l$ over the interval $[0, 1]$. It suffices to show that inclusion isotonicity holds if we shrink only one of the two endpoints. Let $b_0^*, \ldots, b_l^*$ and $b_0^\dagger, \ldots, b_l^\dagger$ be the Bernstein coefficients of p over $[0, 1 - \varepsilon]$ and $[\varepsilon, 1]$, respectively, where $\varepsilon \in (0, 1)$. We refer to the subdivision algorithm

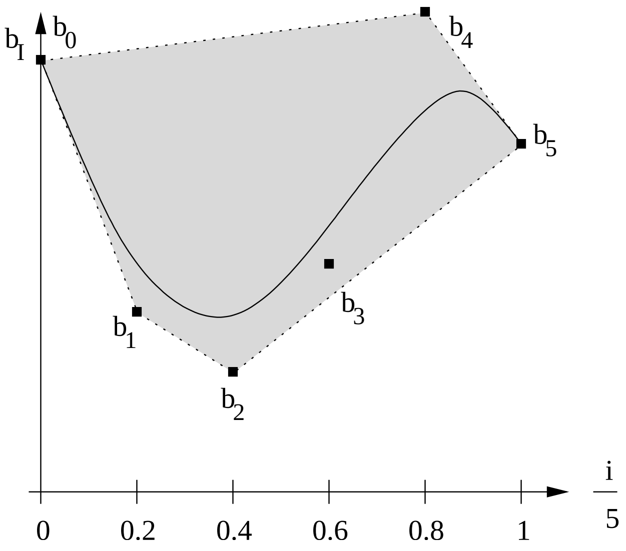


Figure 3.2: The graph of a degree 5 polynomial and the convex hull (shaded) of its control points (marked by squares).

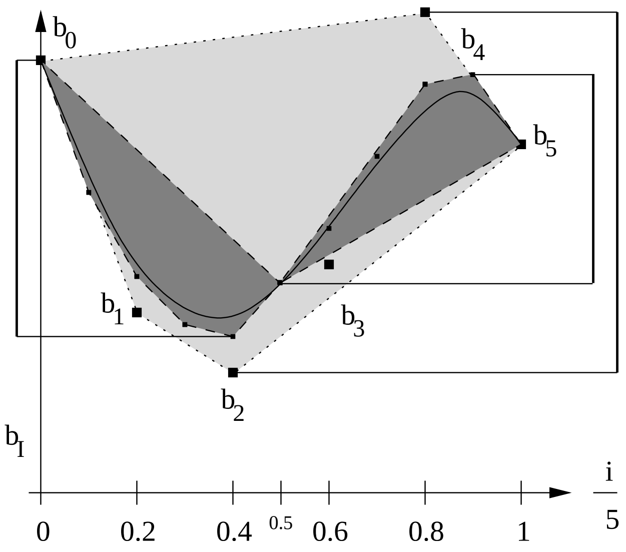


Figure 3.3: The graph of the polynomial in Figure 3.2 with the convex hull (shaded light) of its control points and its Bernstein enclosure, together with the smaller convex hulls (shaded dark) and enclosures over sub-domains arising from a bisection.

in Subsection 3.3.2 and may use it to compute the Bernstein coefficients on the intervals $[0, 1-\varepsilon]$ and $[\varepsilon, 1]$. In (3.27), only convex combinations are formed, therefore

$$\mathbf{b}_i^* \in conv\{\mathbf{b}_0, \ldots, \mathbf{b}_i\} \quad \text{and} \quad \mathbf{b}_i^\dagger \in conv\{\mathbf{b}_i, \ldots, \mathbf{b}_l\}, \;\; i = 0, \ldots, l. \;\; \square \tag{3.21}$$

In the multivariate case, the proof of the inclusion isotonicity of the convex hull of the control points can be obtained similarly by shrinking the unit box in each dimension separately, thus forming convex combinations in only one direction.

As an immediate consequence, we also have:

Corollary 3.1 (Hong and Stahl [HS95a])**.** *The enclosure for the range of a univariate polynomial p over an interval $\mathbf{X}$ provided by the interval spanning its Bernstein coefficients $b_0, \ldots, b_l$ is inclusion isotone.*

Figure 3.3 illustrates this property with the same polynomial as above, after performing a bisection about the midpoint. The inclusion isotonicity of convex-concave extensions and affine bounding functions based upon the Bernstein coefficients is considered in Chapter 10.

3.2.8 Partial Derivatives

As given by Farouki and Rajan [FR88], the partial derivative of p with respect to x_r, where $r = 1, \ldots, n$, is given by

$$\frac{\partial p}{\partial x_r}(x) \;=\; l_r \sum_{i \leq (l_1, \ldots, l_r - 1, \ldots, l_n)^T} (b_{i_1 \ldots\, i_r+1 \,\ldots i_n} - b_i)\, B_i^{l_1 \ldots\, l_r-1 \,\ldots l_n}(x). \tag{3.22}$$

3.3 Algorithms

We may now detail a number of useful algorithms for Bernstein coefficients, in particular efficient difference table schemes for the computation of Bernstein coefficients both from scratch and over subdivided boxes. Formulae for degree elevation and for the Bernstein coefficients of partial derivatives are also given. Further algorithms (including integration, greatest common divisor, etc.) are given by Farouki and Rajan [FR88].

There seems to be very little software in the public domain that deals with the Bernstein expansion either rigorously or in the multivariate case. However a C++ software library with a number of routines for univariate polynomials in Bernstein form, *BPOLY*, exists [TF01].

As before, it is assumed that we have an n-dimensional polynomial p (3.6) of degree $l = (l_1, \ldots, l_n)$ over a box $\mathbf{X}$ (3.11).

3.3.1 Computation of Bernstein Coefficients

To exploit any of the properties of the Bernstein expansion, it is typically required to compute the whole set of $\prod_{i=1}^{n}(l_i + 1)$ Bernstein coefficients. This is necessary even when it is only desired to determine the minimum and maximum Bernstein coefficient, for the Bernstein enclosure.

It is of course possible to compute the b_i directly, using the generalised formula (3.13), however more efficient methods can be employed to avoid the calculation of many of the binomial coefficients and products inherent therein. Algorithms for the univariate case are given in [Rok82, Fis90] and for the multivariate case in [Gar86, ZG98]. A method has also been proposed [Ray07, RN09] whereby the Bernstein coefficients are arranged in matrix form; this allows one to potentially exploit fast native routines for vector and matrix arithmetic.

We describe here the method of J. Garloff [Gar86, ZG98] for the computation of the Bernstein coefficients in the multivariate case using a difference table scheme:

The first stage of the algorithm requires an affine transformation to be applied to p, yielding $p^\dagger$, such that $p(\mathbf{X}) = p^\dagger(\mathbf{I})$, where $\mathbf{I} = [0,1]^n$, i.e.

$$p(x_1, \ldots, x_n) = p^\dagger\left(\underline{x}_1 + (\overline{x}_1 - \underline{x}_1)x_1, \ldots, \underline{x}_n + (\overline{x}_n - \underline{x}_n)x_n\right). \tag{3.23}$$

The coefficients of $p^\dagger$ are labelled $a_i^\dagger$, $0 \le i \le l$. They may be calculated efficiently by the multivariate Horner scheme (see, e.g., [Dow90, PS00]).

The second stage of the algorithm generates the Bernstein coefficients iteratively, from the coefficients of the transformed polynomial:

- $b_i \; := \; \frac{a_i^\dagger}{\binom{l}{i}}, \;\; 0 \le i \le l.$
- For $r = 1, \ldots, n$:
  - For $k = 1, \ldots, l_r$:

$$\begin{aligned} b_i^* \; &:= \; \left\{ \begin{array}{ll} b_i, & \text{if } i_r < k \\ b_i + b_{i_1,\ldots,i_r-1,\ldots,i_n}, & \text{if } i_r \ge k \end{array} \right\}, \;\; 0 \le i \le l. \\ b_i \; &:= \; b_i^*, \;\; 0 \le i \le l. \end{aligned} \tag{3.24}$$

- b_i are the desired Bernstein coefficients, $0 \le i \le l$.

The algorithm (excluding the multivariate Horner scheme) exhibits time complexity $\mathrm{O}(n\hat{l}^{n+1})$ and space complexity (equal to the number of Bernstein coefficients) $\mathrm{O}((\hat{l}+1)^n)$, where $\hat{l} = \max_{j=1}^n l_j$. It may be noted that such exponential complexity renders the computation of the entire set of Bernstein coefficients infeasible for polynomials with moderately many (typically, 10 or more) variables.

3.3.2 Subdivision

Where the Bernstein enclosure is employed in branch-and-bound methods, it is required to subdivide a box into sub-boxes and recompute the Bernstein enclosure (i.e. the set of

Bernstein coefficients) over each sub-box. Due to the inclusion isotonicity (see Section 3.2.7) of the Bernstein form, these enclosures cannot be wider than that for the parent box. In fact, the enclosures for sub-boxes will usually improve (i.e. narrow), if the bounds are not already sharp (see Section 3.2.5). It is also known [Sta95] that a bisection performed around zero *will* yield an improvement of the bounds, unless they are already sharp.

The most common type of subdivision is a bisection performed in one direction by bisecting the corresponding component interval of the box. Heuristics for the selection of subdivision direction are considered in [RC95, ZG98, GS01b, NA07, RN09]. A subdivision algorithm for the univariate case was first proposed in [LR81].

Given the Bernstein coefficients b_i of p over $\mathbf{X}$, we wish to compute the Bernstein coefficients over sub-boxes $\mathbf{X}_1$ and $\mathbf{X}_2$ which result from subdividing $\mathbf{X}$ in the rth direction, i.e.

$$\begin{aligned} \mathbf{X}_1 &= [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_r, x_\lambda] \times \ldots \times [\underline{x}_n, \overline{x}_n], \\ \mathbf{X}_2 &= [\underline{x}_1, \overline{x}_1] \times \ldots \times [x_\lambda, \overline{x}_r] \times \ldots \times [\underline{x}_n, \overline{x}_n], \end{aligned} \tag{3.25}$$

where

$$x_\lambda := (1-\lambda)\underline{x}_r + \lambda \overline{x}_r, \tag{3.26}$$

for some $\lambda \in (0,1)$. In the case of a bisection, $\lambda = \frac{1}{2}$.

It is possible to recompute the Bernstein coefficients over $\mathbf{X}_1$ and $\mathbf{X}_2$ (denoted $b_i^{\mathbf{X}_1}$ and $b_i^{\mathbf{X}_2}$, respectively) from scratch using (3.9), but a more efficient method is similar to the de Casteljau algorithm in computer-aided geometric design (see, e.g., [Far02, PBP02]):

- $b_i^{(0)} := b_i, \; 0 \le i \le l.$
- For $k = 1, \ldots, l_r$:

$$b_i^{(k)} := \left\{ \begin{array}{ll} b_i^{(k-1)}, & \text{if } i_r < k \\ (1-\lambda) b_{i_1,\ldots,i_r-1,\ldots,i_n}^{(k-1)} + \lambda b_i^{(k-1)}, & \text{if } i_r \ge k \end{array} \right\}, \quad 0 \le i \le l. \tag{3.27}$$

- $b_i^{\mathbf{X}_1} := b_i^{(l_r)}, \; 0 \le i \le l.$
- $b_i^{\mathbf{X}_2} := b_{i_1,\ldots,l_r,\ldots,i_n}^{(l_r-i_r)}, \; 0 \le i \le l.$

In the final step, the Bernstein coefficients $b_i^{\mathbf{X}_2}$ on the neighbouring sub-box $\mathbf{X}_2$ are obtained as intermediate values in this computation, since for $k = 0, \ldots, l_r$ the following relation holds [Gar93]:

$$b_{i_1,\ldots,l_r-k,\ldots,i_n}^{\mathbf{X}_2} = b_{i_1,\ldots,l_r,\ldots,i_n}^{(k)}.$$

Figure 3.4 illustrates the subdivision process with the unit box in the bivariate case ($n = 2$) with $\lambda = \frac{1}{2}$.

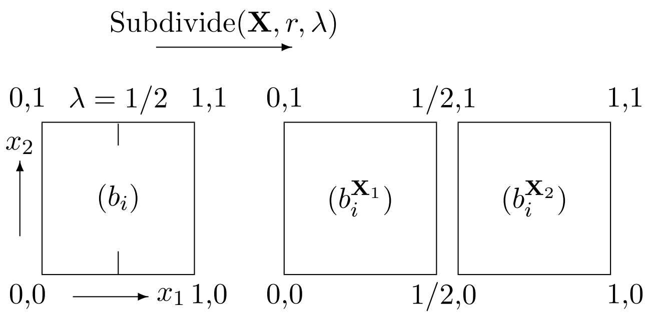


Figure 3.4: Example subdivision of the unit box $\mathbf{X} = \mathbf{I}$ with $n = 2$, $r = 1$, and $\lambda = \frac{1}{2}$.

3.3.3 Degree Elevation

For simplicity we consider here the univariate case. Assuming we have already computed the degree k Bernstein coefficients $b_i^{[k]}$ of p over $\mathbf{x}$, we wish to compute the set of degree $k+1$ Bernstein coefficients, i.e. the coefficients of the degree $k+1$ Bernstein expansion of p, where the superscript in square brackets denotes the degree of the coefficient. Given the formulae for degree elevation of the Bernstein basis polynomials (Section 3.1.1), we can deduce (see also [FR88]) that the coefficients of higher degree can be expressed as a simple weighted sum of the existing lower degree coefficients, as follows:

$$b_i^{[k+1]} = \frac{i b_{i-1}^{[k]} + (k+1-i) b_i^{[k]}}{k+1}, \text{ with } b_{-1}^{[k]} = b_{k+1}^{[k]} = 0, \ i = 0, \ldots, k+1. \tag{3.28}$$

3.3.4 Bernstein Coefficients of Partial Derivatives

As before, assume that we have the Bernstein coefficients b_i of p over $\mathbf{X}$. Here we wish to compute the Bernstein coefficients b'_i, $0 \leq i \leq l^*$, of $p' := \frac{\partial p}{\partial x_r}$, over $\mathbf{X}$, for some $r \in \{1, \ldots, n\}$, and where $l^* := (l_1, \ldots, l_r - 1, \ldots, l_n)^T$. Referring to (3.22) we can see that these coefficients can be calculated in a relatively fast and straightforward manner, by taking linear combinations of the b_i:

$$b'_i = l_r \left(b_{i_1, \ldots, i_r + 1, \ldots, i_n} - b_i \right), \quad 0 \leq i \leq l^*. \tag{3.29}$$

3.4 Mean Value Bernstein Form

The mean value form, a type of interval enclosure, was outlined in Section 2.2. It is possible to derive a new enclosure for the range of a polynomial p over a box $\mathbf{X}$ by combining this with the Bernstein form (3.7) as follows (for simplicity, only the univariate case is considered here):

Using the mean value theorem, from (2.4) we have

$$p(x) = p(\check{x}) + p'(\xi)(x - \check{x}), \tag{3.30}$$

where $\xi \in \mathbf{X}$ lies between x and the midpoint $\check{x}$ of $\mathbf{X}$. To compute an enclosure for $p(\mathbf{X})$ we replace x on the right hand side of (3.30) by $\mathbf{X}$ and have to find an enclosure for $p'(\mathbf{X})$. We can apply Bernstein expansion to p', using the easy calculation of the Bernstein form of p' from the Bernstein form of p, cf. (3.29). This yields the *mean value Bernstein form* (given here with $\mathbf{X} = \mathbf{I} = [0, 1]$, for simplicity)

$$p(\frac{1}{2}) + BE(p')[-\frac{1}{2}, \frac{1}{2}], \tag{3.31}$$

which encloses $p([0, 1])$, where $BE(p')$ denotes the interval spanned by the minimum and maximum of the Bernstein coefficients of p', cf. (3.17). However, the mean value Bernstein form does not yield any improvement on the usual Bernstein form:

Theorem 3.6 ([GS05, Appendix]). *Let p be a univariate polynomial of degree l with Bernstein coefficients $b_i, i = 0, \ldots, n$, over $[0, 1]$. If the lower or upper bound for $p([0, 1])$ provided by the Bernstein coefficients is not sharp, then the width of this enclosure is strictly less than that of the mean value Bernstein form.*

Proof: Using (3.29), we can write (3.31) as

$$p(\frac{1}{2}) + \frac{l}{2} \max_{i=0,\ldots,l-1} |b_{i+1} - b_i|[-1, 1].$$

Thus the width of the mean value Bernstein form is

$$l \max_{i=0,\ldots,l-1} |b_{i+1} - b_i|. \tag{3.32}$$

Let

$$b_L := \min_{i=0,\ldots,l} \{b_i\} \text{ and } b_U := \max_{i=0,\ldots,l} \{b_i\}.$$

Assume without loss of generality that $L < U$. If either bound is non-sharp, we have $U - L \leq l - 1$. The width of the enclosure provided by the usual Bernstein form can be bounded as follows:

$$\begin{aligned}
b_U - b_L &= (b_U - b_{U-1}) + (b_{U-1} - b_{U-2}) + \ldots + (b_{L+2} - b_{L+1}) + (b_{L+1} - b_L) \\
&\leq |b_U - b_{U-1}| + |b_{U-1} - b_{U-2}| + \ldots + |b_{L+2} - b_{L+1}| + |b_{L+1} - b_L| \\
&\leq (U - L) \max_{i=0,\ldots,l-1} |b_{i+1} - b_i| \\
&\leq (l - 1) \max_{i=0,\ldots,l-1} |b_{i+1} - b_i|.
\end{aligned}$$

Comparison with (3.32) concludes the proof. □

3.5 Bézier Curves

An important application of Bernstein polynomials is in the construction of Bézier curves, which are briefly outlined here.

Suppose we are given $l + 1$ *control points* $P_0, \ldots, P_l \in \mathbb{R}^n$. This defines a degree l Bézier curve, which is a function mapping a single parameter t to co-ordinates in $\mathbb{R}^n$:

$$Bz(t) \ = \ \sum_{i=0}^{l} B_i^l(t) P_i, \tag{3.33}$$

where the B_i^l are the degree l Bernstein basis polynomials (3.2). Any point $Bz(t)$, where $t \in [0, 1]$, on the curve is a weighted average of the control points $P_0, \ldots, P_l$ and lies within their convex hull. Furthermore, $Bz(0) = P_0$ and $Bz(1) = P_l$; the curve thus starts at P_0 and passes somewhat close to each of the control points $P_1, \ldots, P_{l-1}$ in turn (without, in general, passing through them), before ending at P_l.

Bézier curves are thus a very simple way to define and represent a smooth curved surface in a computer. They have their origin in automobile design (originally at Citroën and Renault in France) in the 1960s and have since become ubiquitous in the field of computer-aided geometric design; they are also used in computer drawing programs and the design of fonts, e.g. [Knu86]. The operation of degree elevation (3.28) is very commonly used to refine a curve through the addition of a new control point, cf. [TP96]. Bézier curves may be generalised into (and are components of) different types of splines, the most common of which is the B-Spline (basis spline), also widely employed in computer-aided geometric design.

Further material on Bézier curves may be found in [Far02, PBP02]. Detailed discussion on the application of Bernstein polynomials to Bézier curves appears in [Ber00]; another application of Bernstein expansion to computer-aided geometric design is given in [HMSP96].

4 Topological Degree

Topological degree is a concept from mathematical analysis that is central to the issue of counting the number of roots to a system of equations. Root-counting can be a useful tool in solving such systems. Topological degree is related to the theory of fixed points of functions and is a generalisation of the winding number of a curve.

There are various types of topological degree, cf. [Llo78], some concerned with finite-dimensional spaces, such as the *Brouwer degree*, others with infinite-dimensional spaces, such as the *Leray–Schauder* degree, which is useful for demonstrating the existence of solutions to differential equations. It is the Brouwer degree which is of interest to the problem of solution enumeration (and by extension, solution) of systems of nonlinear equations.

4.1 The Brouwer Fixed Point Theorem

The Brouwer fixed point theorem [Bro12] is the foundation of the theory of topological degree and is one of the building blocks of modern topology.

Theorem 4.1 (Brouwer)**.** *A continuous mapping $\mathcal{F}$ of a compact convex subset D of a Euclidean space to itself has a fixed point, i.e. there exists an $x \in D$ such that $\mathcal{F}(x) = x$.*

The condition on D can also be stated as being homeomorphic to the closed unit ball in $\mathbb{R}^n$. Where $D \subseteq \mathbb{R}^n$, the condition holds if and only if it is closed and bounded.

This theorem establishes the principle that the existence of solutions to equations within a region can be determined without detailed information on function behaviour. We only require continuity. As given in Lloyd [Llo78], the general aim of degree theory is as follows:

Suppose that we have an open subset D of a topological space $\mathcal{X}$, where $y \in \mathcal{X}$ and $\mathcal{F} : D \to \mathcal{X}$ is continuous. We wish to define an integer $deg(\mathcal{F}, D, y)$, the *degree* of $\mathcal{F}$ at y relative to D, with the following properties:

1. The degree $deg(\mathcal{F}, D, y)$ is an estimate of the number of solutions of $\mathcal{F}(x) = y$ for all $x \in D$.

2. The degree is continuous in both $\mathcal{F}$ and y.

3. The degree is additive in the domain D, so that if $D = D_1 \cup D_2$, where D_1 and D_2 are disjoint, then

$$deg(\mathcal{F}, D, y) \; = \; deg(\mathcal{F}, D_1, y) + deg(\mathcal{F}, D_2, y).$$

The first property is the most fundamental. Topological degree is a useful tool for root counting precisely because it is a measure which is *defined by* a (form of) root count. The other properties hint at the usefulness of topological degree from a computational viewpoint. Property 2 means that it is possible to compute the degree *exactly*, given a sufficiently good approximation to the arguments by some finite construction. Property 3 demonstrates that solutions can potentially be isolated by subdivision of the region D.

It is hard to define topological degree in greater detail for a general topological space $\mathcal{X}$. In particular, the more general the space $\mathcal{X}$, the more restrictions there are on the function $\mathcal{F}$, which are necessary to preserve the above properties. For a more detailed treatment of general degree theory, see [AH35, Llo78, OR09].

4.2 Brouwer Degree

The Brouwer degree is, broadly speaking, a particular instance of the topological degree where the topological space is specified as $\mathbb{R}^n$.

Conceptually, the degree $deg(\mathcal{F}, D, y)$ of a mapping $\mathcal{F} : \mathbb{R}^n \rightarrow \mathbb{R}^n$, where D is a bounded open subset of $\mathbb{R}^n$ and $y \in \mathbb{R}^n$, is the excess of the number of points of $\mathcal{F}^{-1}(y) \cap D$ at which the Jacobian determinant of $\mathcal{F}$ is positive over those at which it is negative. An integer result, it can be regarded (and utilised) as an estimate to the number of solutions to $\mathcal{F}(x) = y$ in D. In particular, a non-zero result guarantees the presence of at least one such solution. Given a system of equations $\mathcal{F} = 0$ and a box $\mathbf{X}$, $deg(\mathcal{F}, \mathbf{X}, 0)$ thus estimates the number solutions (i.e. roots of $\mathcal{F}$) within $\mathbf{X}$.

Definition 4.1 (Brouwer Degree)**.** *For some n, let D be a bounded open subset of $\mathbb{R}^n$, $\mathcal{F} : cl(D) \rightarrow \mathbb{R}^n$ be continuous, and $y \in \mathbb{R}^n \backslash \mathcal{F}(\partial D)$. Under these conditions, the Brouwer (topological) degree is a function*

$$deg : \{(\mathcal{F}, D, y)\} \rightarrow \mathbb{Z} \tag{4.1}$$

satisfying the following:

- *Identity function property: $deg(id, D, y) = 1$ if $y \in D$.*
- *Additive property: If $D = D_1 \cup D_2$, where D_1 and D_2 are disjoint, or if D_1 and D_2 are disjoint sets such that $y \notin \mathcal{F}(D \backslash (D_1 \cup D_2))$, then*

$$deg(\mathcal{F}, D, y) \ = \ deg(\mathcal{F}, D_1, y) + deg(\mathcal{F}, D_2, y). \tag{4.2}$$

- *Homotopic invariance property: If $\mathcal{F}, \mathcal{G} : cl(D) \rightarrow \mathbb{R}^n$ are continuous, and homotopic with homotopy $H_t(x) = H(t, x)$, then if $H_t^{-1}(y)$ does not intersect the boundary of D for all values of $t \in [0, 1]$, $deg(\mathcal{F}, D, y) = deg(\mathcal{G}, D, y)$.*

See Subsection 5.3.5 for a definition and further discussion of homotopies. It can be shown (and should be clear) that these conditions define the Brouwer degree in a meaningful way for all possible choices of argument satisfying the stated restrictions. Where the restrictions are violated, for example $y \in \mathcal{F}(\partial D)$, it is undefined.

Complex Functions

The above definition for Brouwer degree only caters for real-valued functions of a real variable in a finite number of dimensions. However, it can easily be adapted to cater for complex functions $\mathcal{F}^* : \mathbb{C}^n \to \mathbb{C}^n$. Such a function $\mathcal{F}^*$ may be regarded as a function from $\mathbb{R}^{2n}$ to $\mathbb{R}^{2n}$ and then the degree is defined as before. Indeed, complex functions have an additional beneficial property when treated in this fashion: an analytic function will be *orientation preserving* (i.e. its Jacobian determinant is always non-negative). Therefore the degree will simply count the number of solutions in D, with multiplicity.

4.2.1 Properties of the Brouwer Degree

The following are fundamental properties of the Brouwer degree. They can be of use in guiding the construction of algorithms to compute the degree.

Additive Property

This property (4.2) holds axiomatically and provides that the topological degree is defined as a summation of values for all roots present in a given region.

Root-Counting Property

If $\mathcal{F}(x) = y$ has no solutions in D, then $deg(\mathcal{F}, D, y) = 0$. Otherwise, if all the solutions of $\mathcal{F}(x) = y$ in D are non-singular and $\mathcal{F}$ has continuous derivatives, then $deg(\mathcal{F}, D, y)$ is the sum of the signs of the Jacobian determinants at all the solutions:

$$deg(\mathcal{F}, D, y) \;=\; \sum_{x \in \mathcal{F}^{-1}(y)} sgn(det(J_{\mathcal{F}}(x))). \tag{4.3}$$

This property is meaningful for cases with continuously differentiable functions and non-singular solutions. In the case of singular solutions, the Brouwer degree is less well-defined — here it merely indicates the existence of solutions.

Boundary Determinism

The Brouwer degree $deg(\mathcal{F}, D, y)$ is determined by y and the values of $\mathcal{F}$ on the boundary of D only. This property is crucial since it means that the degree can (in principle) be computed merely by analysing the behaviour of the function $\mathcal{F}$ on the boundary of a region, irrespective of the complexity of the behaviour of $\mathcal{F}$ within the region in question.

Figure 4.1 illustrates this property for a simple system $f_1 = 0, f_2 = 0$ in $\mathbb{R}^2$; the ordering of the zero sets around the boundary of the region determines whether either an even or odd number of solutions are present, not the behaviour of the functions within the box. In the left-hand case, there must be an even number of solutions (since the degree sums to zero in this particular example, half have a positive Jacobian determinant and the other half negative). In the base case (where the zero sets are depicted with solid lines) there are zero solutions; in an alternative case (where the zero set of f_1 is a dashed line) there might be two solutions. Conversely, in the right-hand case, there must be an odd number of solutions.

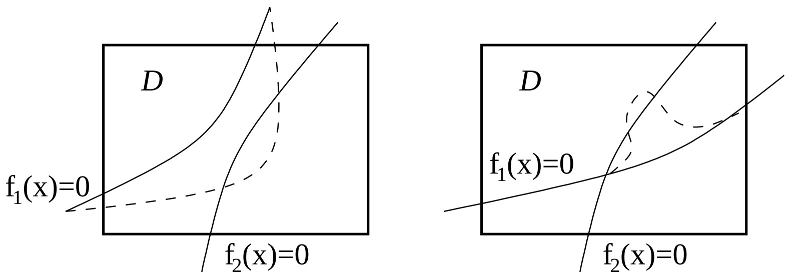


Figure 4.1: Boundary determinism in $\mathbb{R}^2$: even and odd numbers of solutions.

Continuity

This property also holds axiomatically (see Definition 4.1, homotopic invariance property). This provides that the Brouwer degree $deg(\mathcal{F}, D, y)$ is an integer which is (locally) continuous with respect to $\mathcal{F}$ and y. In other words, it is constant for small peturbations in $\mathcal{F}$ and y.

Suppose that $\mathbf{Y}$ is a box which contains points y_1 and y_2. Then if $\mathcal{F}^{-1}(\mathbf{Y})$ does not intersect the boundary of D, $deg(\mathcal{F}, D, y_1) = deg(\mathcal{F}, D, y_2)$.

This illustrates that a smooth change in either y or $\mathcal{F}$, in such a way as to not move solutions to $\mathcal{F}(x) = y$ across the boundary of D, leaves the degree $deg(\mathcal{F}, D, y)$ unchanged. In practice this means that sufficiently good numerical approximations to $\mathcal{F}$ and y will be enough to compute the degree *exactly*, given sufficient precision. This property is completely ruined in the case where a solution lies on the boundary of D, which is why these cases are explicitly excluded.

4.2.2 Example

This is a simple system of equations taken from Aberth [Abe94]. $\mathcal{F} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is given by

$$\begin{aligned} f_1 &= x_1^2 + x_2^2 - x_3, \\ f_2 &= x_2^2 + x_3^2 - x_1, \\ f_3 &= x_3^2 + x_1^2 - x_2. \end{aligned}$$

$\mathcal{F}$ has roots at $(0, 0, 0)$ and $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, the former with negative Jacobian determinant and the latter with positive Jacobian determinant. Hence:

- $deg(\mathcal{F}, ([-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}]), 0) = -1$. This (box) region encloses only the first root, so the degree is the sign of its Jacobian determinant, -1.

- $deg(\mathcal{F}, ([\frac{1}{4}, \frac{3}{4}], [\frac{1}{4}, \frac{3}{4}], [\frac{1}{4}, \frac{3}{4}]), 0) = 1$. This region encloses only the second root, so the degree is the sign of its Jacobian determinant, 1.
- $deg(\mathcal{F}, ([1, 2], [1, 2], [1, 2]), 0) = 0$. This region encloses neither of the roots, so the degree is 0.
- $deg(\mathcal{F}, ([-1, 1], [-1, 1], [-1, 1]), 0) = 0$. This region encloses both roots, so the degree is the sum of the signs of the Jacobian determinants, 0. This example neatly illustrates a potential cause for confusion — it can be difficult to discern whether there are no roots, or roots with Jacobian determinants of opposite sign which cancel out.

4.3 Algorithms for Computing Topological Degree

Here we discuss each of the main known algorithms for computing Brouwer degree, covering the recursive method in greater detail in Section 4.4. Very few techniques are devoted to the computation of the topological degree as an objective in itself, although there also exist methods that seek to exploit it indirectly, e.g. [BFLW09], an existence test for systems of equations based on the construction of homotopies (see Definition 5.4) and the application of interval arithmetic.

4.3.1 Integration over the Boundary

Given a box D and component functions f_i, as before, we may consider the function $h(x) = \mathcal{F}(x)/|\mathcal{F}(x)|$ on the boundary of D. We see that h is a mapping onto the unit sphere in $\mathbb{R}^n$, and the degree is given by the number of times which h wraps the boundary around the unit sphere. This is otherwise referred to as the *winding number*.

The method of O'Neal and Thomas [OT75] makes use of a definition of the degree in terms of the Kronecker integral [Kro69]. The integral of the Jacobian of h over the boundary of D is divided by the surface area of the unit sphere to give the degree. A degree computation thus essentially becomes a (fairly hard) problem of numerical integration. In [OT75], a quadrature method was used to compute the integral. It seems that only a very low precision should be required, since an error of less than $\frac{1}{2}$ suffices to determine the correct (integer) answer. However the error analysis performed by the authors is complicated, and they were unable to guarantee results within an error margin, although they showed that the probability of error could be made arbitrarily small, and that the numerical convergence is good. The method was succesfully employed for some examples in $\mathbb{R}^3$ and $\mathbb{R}^6$. Using this basic idea, one is of course dependent upon the performance of the numerical method chosen.

4.3.2 Triangulation of the Boundary

Very similar to the integration over the boundary method, a technique has been proposed which also considers the function h which wraps the boundary of D around the unit sphere, but which does not rely upon numerical integration. Instead, the boundary is triangulated into a set of simplices $s_1, \ldots, s_i$ for which each simplex does not wrap the unit sphere. More specifically, for x_1 and x_2 in a simplex, $|h(x_1) - h(x_2)| < 1$. This is based on the work of A. Eiger et al. [ESS84], who used a simplex-bisection scheme in the construction of a nonlinear solver.

This method can work by projecting the vertices of each simplex under the mapping h, and comparing the components of the results with a sample point on the unit sphere. With the restriction on the variance of $h(x)$ over the simplices, we are sure that none wholly wraps the unit sphere. In essence, the sample point is a marker on the sphere and we count the winding number according to the orientations of those simplices with which it intersects. This could be regarded as a symbolic analogue of the integration method. Given a set of suitable simplices, it appears that the rest of the algorithm is fairly trivial, but the question of how to appropriately determine the simplices is hard.

4.4 The Recursive Method

This section describes in detail the algorithm which is explored further in Chapter 7. A formulation of the algorithm was proposed by O. Aberth [Abe94], which is in turn an elaboration of the method of R. B. Kearfott [Kea79b]. Unlike the two methods outlined above, this method does not consider the winding number of a mapping to the unit sphere at all. Instead, it uses interval arithmetic with the component functions f_i on the boundary of D. It only considers the case where D is a co-ordinate aligned box (see Def. 2.2), and recurses by taking faces of D of succesively lower dimension.

The broad principle is that the boundary of the box is projected under $\mathcal{F}$, and then there exists a solution to $\mathcal{F}(x) = y$ in D if and only if y is contained within the projection of the box (this is a region of $\mathbb{R}^n$ which itself is not in general a box). This is determined by taking any ray emanating from y and counting the number of times which it crosses this projection. If, for instance, it crosses once then y must lie within the projection, assuming that the projection is bounded — which will be the case for a continuous $\mathcal{F}$ over a finite box.

4.4.1 Faces of a Box

Let us suppose we have a starting box in $\mathbb{R}^n$ as follows:

$$\mathbf{X} \;=\; [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]. \tag{4.4}$$

A *face* is obtained by restricting one of the component intervals to an endpoint value (for some i, either $\underline{x}_i$ or $\overline{x}_i$). Lower-dimensional faces are obtained by further restricting intervals to an endpoint. These are referred to as *child* faces derived from a *parent*. A face is therefore a lower-dimensional subset of a box, and can be considered as a box of lower dimension, together with one or more point values. A face is subdivided into two or more *sub-faces* by dividing one or more of the intervals into two. Unlike a child face, a sub-face therefore has the same dimension as the parent face, but a smaller width in one or more directions.

4.4.2 Scope

This is a recursive technique for the computation of Brouwer degree, using interval arithmetic. Given a box $\mathbf{X}$ in $\mathbb{R}^n$ and a continuous function $\mathcal{F} : \mathbf{X} \to \mathbb{R}^n$ given by the component real-valued functions

$$f_i(x_1, \ldots, x_n), \quad i = 1, \ldots n,$$

the Brouwer degree of $\mathcal{F}$ over $\mathbf{X}$ at 0, $deg(\mathcal{F}, \mathbf{X}, 0)$, is computed. The degree is calculated at 0 without loss of generality — it can be adapted to calculate the degree at a general point $y \in \mathbb{R}^n$ with a trivial modification to the functions f_i.

A set of interval arithmetic routines (see Chapter 2) are required which cater for the types of functions under consideration. Elementary functions, for example, require a set of routines to deal with exponential and trigonometric operators, as well as the usual arithmetic operators. Due to the nature of the algorithm, variable-precision interval arithmetic is needed to make a degree-finder completely infallible.

4.4.3 Overview

Consider a simple example in $\mathbb{R}^2$, illustrated by Figure 4.2.

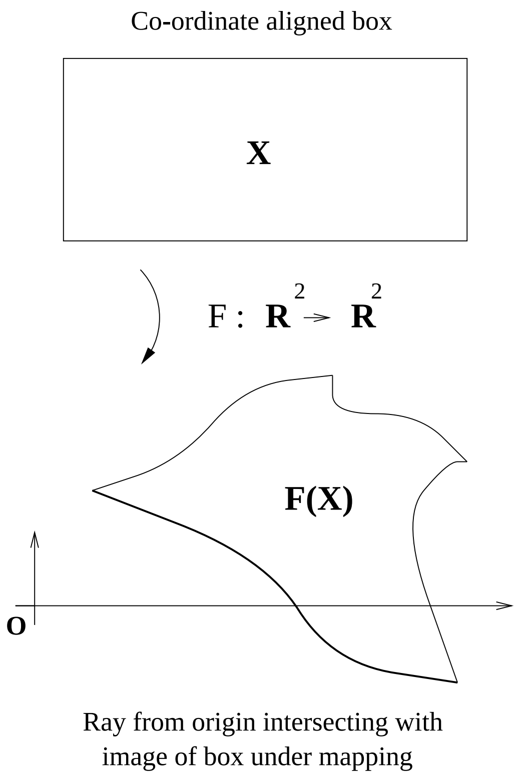

Co-ordinate aligned box
X
F : R^2 -> R^2
F(X)
O
Ray from origin intersecting with image of box under mapping

Figure 4.2: Computation of topological degree: example in $\mathbb{R}^2$.

In this case we test to see whether the image of $\mathbf{X}$ under the mapping contains the origin. A ray is drawn from the origin (for the sake of simplicity it is taken in the positive direction of the primary variable). The question is answered by determining the number of times the ray crosses the boundary of $\mathcal{F}(\mathbf{X})$; twice in this case. Clearly, the origin is only contained within $\mathcal{F}(\mathbf{X})$ if the ray crosses an odd number of times.

To determine which segments of $\mathcal{F}(\mathbf{X})$ intersect with the ray, the boundary of the box $\mathbf{X}$ is split into its 4 component faces s_1, s_2, s_3, s_4 and the image of each face $f(s_i)$ is considered in turn. Deciding whether a given segment $f(s_i)$ intersects with the ray is perhaps not quite as trivial as it might first seem. Using interval arithmetic, two interval enclosures are computed:

$$\begin{aligned}\mathbf{y}_1 &:= f_1(s_i),\\ \mathbf{y}_2 &:= f_2(s_i).\end{aligned}$$

If $\mathbf{y}_1$ is always positive and $\mathbf{y}_2$ contains zero, then the ray must intersect this segment. Similarly, if $\mathbf{y}_1$ is always negative or if $\mathbf{y}_2$ does not contain the zero point, there cannot be an intersection. This seems simple enough, but the problematic case is when both $\mathbf{y}_1$ and $\mathbf{y}_2$ contain zero. It is not immediately clear how to discern the two different cases illustrated in Figure 4.3.

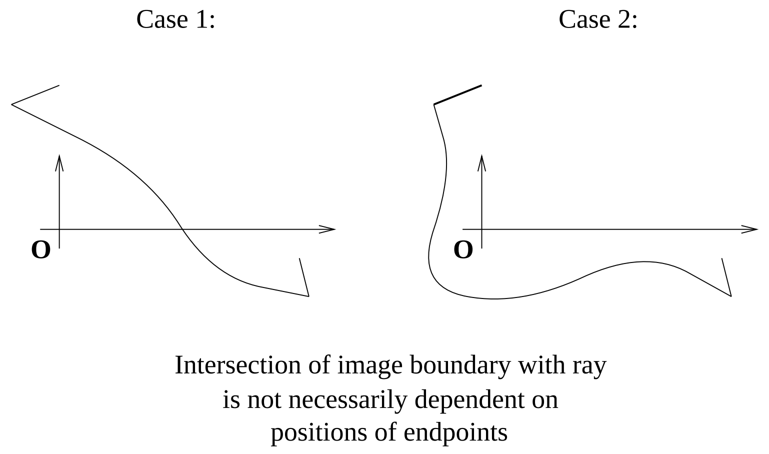

Case 1:
Case 2:
O
O
Intersection of image boundary with ray
is not necessarily dependent on
positions of endpoints

Figure 4.3: Computation of topological degree: examples with $\mathbf{y}_1$ and $\mathbf{y}_2$ containing zero.

In fact in this case the troublesome face must be subdivided and each fragment (sub-face) must be analysed in turn. Further subdivisions must be performed until each fragment satisfies one of the above criteria. When this is done for all the faces s_i, the *crossing parity* is equal to the number of intersections modulo two. Alternatively, the Brouwer degree can be found by assigning orientations of ± 1 to the faces, and taking the sum of the orientations of all the faces that intersect. The crossing parity is slightly cruder in that it only indicates the presence of one or more roots, rather than incorporating information about the signs of

Jacobian determinants. That is to say, the degree is an integer whereas the parity is just a binary result.

This example in $\mathbb{R}^2$ does not illustrate the recursive nature of the algorithm, however an example in $\mathbb{R}^3$ (see Figure 4.4) — beyond the base two-dimensional case — shows how the recursion arises and illustrates the algorithm more fully.

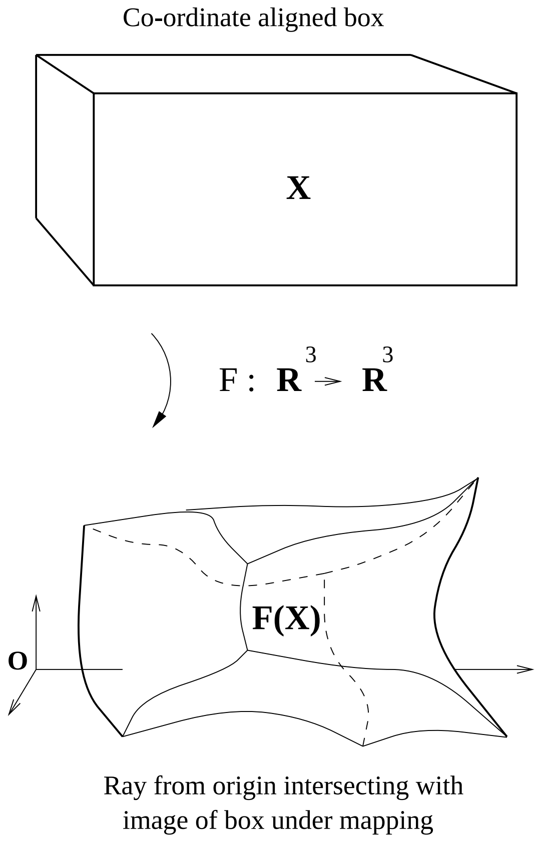

Co-ordinate aligned box
X
F : R^3 → R^3
F(X)
O
Ray from origin intersecting with image of box under mapping

Figure 4.4: Computation of topological degree: example in $\mathbb{R}^3$.

Here a list of 6 faces in two dimensions is analysed in a similar fashion. As before, the functions f_1, f_2, f_3 are used to generate interval enclosures $\mathbf{y}_1$, $\mathbf{y}_2$, and $\mathbf{y}_3$. This time, those faces which are positively checked (those for which $\mathbf{y}_1$ is wholly positive and for which $\mathbf{y}_2$ and $\mathbf{y}_3$ contain zero) are tested further, since it is not guaranteed that the ray from the origin intersects the images of these faces. An intersection here seems *likely*, but it is not possible (without further computation) to discern the two cases as illustrated in Figure 4.5

It can be seen that the question of whether or not a given face contributes to the degree resolves to another Brouwer degree computation. This sub-problem has one fewer dimension, with the primary variable (x_1) having been eliminated. The ray is this time taken in the positive direction of x_2.

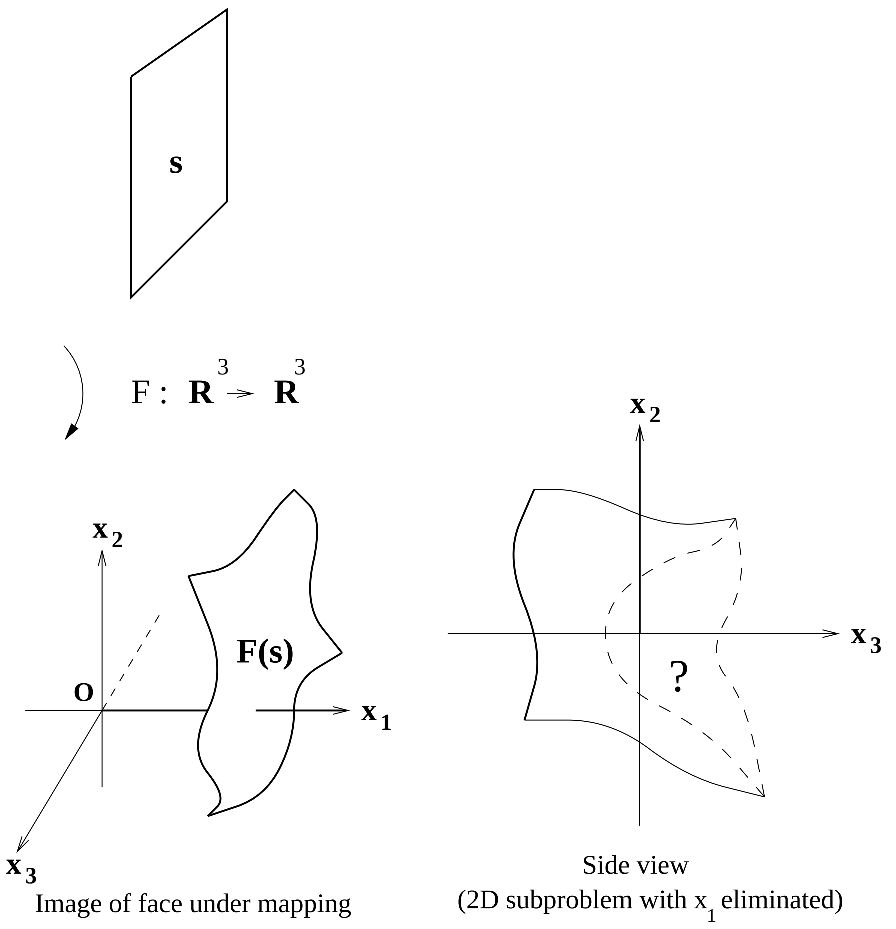

S
F : R^3 → R^3
x2
O
F(S)
x1
x3
Image of face under mapping
x2
x3
?
Side view
(2D subproblem with x1 eliminated)

Figure 4.5: Computation of topological degree: sub-problem generated in $\mathbb{R}^2$.

Any given degree computation therefore recursively generates a succession of sub-problems of lower dimension, so that the boundaries of the initial box are effectively broken down into a large number of one-dimensional line segments, with the computation ultimately resolving to a large number of function evaluations over these intervals.

A more formal expression of the algorithm formulation is to say that

$$deg(\mathcal{F}, \mathbf{X}, 0) \; = \; i(C^n, 0),$$

where $i(C^n, 0)$ is the incidence number of C^n with the origin in $\mathbb{R}^n$, and C^n is a polyhedral complex approximating $\mathcal{F}(\mathbf{X})$ to sufficient accuracy. The next section provides the algorithm in full.

4.4.4 Detailed Algorithm

Face Generation

The initial stage generates an array L of the $2n$ faces of $\mathbf{X}$ (4.4). For $j = 1, \ldots, n$ add the list elements whose components are given by

$$s_{2j-1_i} = \begin{cases} [\underline{x}_i, \overline{x}_i] & \text{if } i \neq j, \\ \underline{x}_i & \text{if } i = j, \end{cases}$$

$$s_{2j_i} = \begin{cases} [\underline{x}_i, \overline{x}_i] & \text{if } i \neq j, \\ \overline{x}_i & \text{if } i = j. \end{cases}$$

Each face therefore comprises the product of $n-1$ intervals taken from $\mathbf{X}$, plus either the left-most or right-most point value from the remaining interval. The dimension of each face is $n-1$.

There is also an orientation field which is assigned to each face. For each of the $2n$ faces this is assigned as follows:

- The face which has $x_j = \underline{x}_j$ is given orientation $\sigma = (-1)^j$.
- The (opposite) face which has $x_j = \overline{x}_j$ is given orientation $\sigma = (-1)^{j+1}$.

Selection of Faces

This process takes the array of faces L generated above and performs a series of tests, using interval arithmetic. Some faces (which are tested positively) are placed into a new array L_1. Others are subdivided into smaller sub-faces (of the same dimension) which are appended to L for repeated analysis. Others are discarded. The amount of subdivision required may depend in part on the extent of the overestimation due to the use of interval arithmetic. Note that the array L grows dynamically with the addition of smaller fragments of larger faces — there may be an issue concerning the best order in which to search the elements of L, which is discussed in Chapter 7.

Eventually the array L is exhausted with the generation of L_1. Clearly, many of the entries of L_1 will be faces that were also present in L, and all entries will have the same dimension. This second array represents those portions of the boundary for which it is *suspected* that the ray from the origin intersects, destined for further examination. It can only be a suspicion, since for problems of non-trivial dimension the polyhedral complex formed by applying the function to the boundary is only an approximation to $\mathcal{F}(\mathbf{X})$; in some cases not a very good one.

More formally, for each face s_i in L, do the following:

1. For $j = 1, \ldots, n$, use interval arithmetic to generate interval values, $\mathbf{y}_j$, from applying f_j to the interval s_i.

2. Do one of the following:
    - If any of $\mathbf{y}_2, \dots, \mathbf{y}_n$ do not contain zero, then discard s_i.
    - If $\mathbf{y}_1$ is to the left of zero, i.e. $\overline{\mathbf{y}}_1 < 0$, then discard s_i.
    - If $\mathbf{y}_1$ is to the right of zero, i.e. $\underline{\mathbf{y}}_1 > 0$, then add s_i to L_1.
    - Otherwise generate the 2^{n-1} sub-faces that can be formed from s_i by bisecting all component intervals and taking either the left-hand or right-hand half of each. Copy point values and orientations σ. Append these sub-faces to L.
3. Process next face s_{i+1}.

This procedure is iterated until the array L is exhausted.

Reduction of Faces

This procedure takes the array of faces L_1 selected above and generates their component child faces of one lower dimension. For example, if the initial box $\mathbf{X}$ was a cuboid in $\mathbb{R}^3$, then the input array would comprise faces of dimension two — rectangles with one variable fixed. These would now be reduced to a list of one-dimensional line segments (with two variables fixed), obtained from the 4 edges of the rectangles.

For each element of L_1, in the first iteration $2(n-1)$ child faces are generated, in the same fashion as for the initial generation of the faces. The orientation of each child face is determined by multipying σ, the orientation of the parent face, by $(-1)^j$ or $(-1)^{j+1}$, depending on whether the left-most or right-most point, respectively, of the jth *non-degenerate* interval component of the parent was taken. All of the child faces are placed into a new array L.

Unfortunately, this process is greatly complicated by a seemingly minor consideration. One of the limitations of the algorithm is that it cannot work in the case of a solution occuring on the boundary of the box. It is taken as given that this scenario is excluded for the initial computation, but we need to ensure that this is also the case for the automatically generated sub-problems. Since the new system of equations upon recursion will have had the primary variable fixed (and the first function eliminated) it *is* nevertheless possible that $f_2, \dots, f_n$ are simultaneously zero somewhere on a child face. Such a scenario would cause an infinite subdivision of faces and would result in failure to terminate. However, such problematic child faces would have been generated *twice*, from different parents — where this boundary condition occurs, it must by construction also occur for a neighbouring selected parent face. For example, with an initial cuboid in $\mathbb{R}^3$, two adjacent rectangular faces have a common edge (child face). It can be seen that double selection of child faces is a necessary condition for the failure case, furthermore in such cases the two copies of the child face would, by construction, have opposite orientations. So two duplicate child faces cancel out in the computation, making contributions to the result that are equal but opposite in sign, and can safely be eliminated. This fail-safe mechanism is required in order to guarantee that the computation succeeds, although it does introduce new overhead costs.

After the generation of child faces, an exhaustive search is therefore performed, firstly to find, and then to eliminate any duplicates. The situation is complicated further by the fact that faces may not only be strict duplicates, but may also partially overlap one another. For instance one child face may be the result of a subdivided parent face, and thus form a subset of the larger copy.

In $\mathbb{R}$, it is straightforward to resolve overlap. There are two line segments, and the ovelapping portion can easily be removed by comparing endpoints, and one (or two) new line segments can be computed which represent the symmetric difference. In $\mathbb{R}^3$, however, more work is needed. Consider two cuboid faces that overlap in one corner. In removing the duplicated region from one cuboid, we are left with a cuboid that has a block cut out of one corner. This is no longer a primitive shape, and cannot therefore be represented as a simple product of intervals. The simplest approach is to decompose this into 7 primitives (cuboid faces), although this could be done with as few as 3. So to remove the duplication from two overlapping faces in three dimensions may typically require the generation of up to 14 new faces. In the rare case where one face is contained entirely within the other, 26 new faces would result. It seems reasonable to assume that the elimination process may increase the number of faces in L by a theoretical maximum factor of $3^m - 1$, where m is the dimension of the faces (in the first iteration, $m = n - 1$); see Subsection 7.1.2 for an explanation.

Recursion

At this stage we have a new array L of faces with dimension $n - 2$ (in the first recursion), which are analysed in a similar fashion. Essentially we now have to resolve a number of Brouwer degree problems with lower dimension. For the second cycle, a ray from the origin in the direction of x_2 is drawn in each case, so that only $\mathbf{y}_2, \ldots, \mathbf{y}_n$ need to be determined with interval arithmetic, with f_2 being the deterministic function this time. The recursive property of the degree is given by:

$$deg((f_1, \ldots, f_n), \mathbf{X}, 0) = \sum_{s \in L_1} \sigma(s) deg((f_2, \ldots, f_n), s, 0). \tag{4.5}$$

In total, $n - 1$ cycles have to be performed (including the first), at the end of which the result is an array L containing faces of zero dimension (points in $\mathbb{R}^n$). These points s_i need to be filtered to only select those for which $f_n(s_i) > 0$. The degree is given by taking the sum of the orientations of these selected points:

$$deg(\mathcal{F}, \mathbf{X}, 0) = \sum_{i, f_n(s_i) > 0} \sigma(s_i). \tag{4.6}$$

In the case of calculating only the crossing parity, the result is merely the size of L modulo two (again, after selecting points with positive image).

4.4.5 Example

We revisit the straightforward example from Section 4.2.2. $\mathcal{F} : \mathbb{R}^3 \to \mathbb{R}^3$ is given by

$$\begin{aligned} f_1 &= x_1^2 + x_2^2 - x_3, \\ f_2 &= x_2^2 + x_3^2 - x_1, \\ f_3 &= x_3^2 + x_1^2 - x_2. \end{aligned}$$

We wish to compute the degree over the box $\mathbf{X} = ([-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}])$.

Face Generation

The 6 faces initially placed into the array L are:

$$s_1 = (-\frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}]), \sigma = -1$$

$$s_2 = (\frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}]), \sigma = 1$$

$$s_3 = ([-\frac{1}{4}, \frac{1}{4}], -\frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}]), \sigma = 1$$

$$s_4 = ([-\frac{1}{4}, \frac{1}{4}], \frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}]), \sigma = -1$$

$$s_5 = ([-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}], -\frac{1}{4}), \sigma = -1$$

$$s_6 = ([-\frac{1}{4}, \frac{1}{4}], [-\frac{1}{4}, \frac{1}{4}], \frac{1}{4}), \sigma = 1$$

First Iteration

In the testing of L, there is no subdivision of faces (assuming exact interval arithmetic). Only s_5 is passed onto L_1, since

$$f_1(s_5) = [\frac{1}{4}, \frac{3}{8}] > 0$$

and

$$f_2(s_5) = f_3(s_5) = [-\frac{3}{16}, \frac{3}{8}] \ni 0.$$

None of the other s_i satisfy these conditions. Now the 4 sub-faces of s_5 are generated:

$$s_1' = (-\frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}], -\frac{1}{4}), \sigma = 1$$

$$s_2' = (\frac{1}{4}, [-\frac{1}{4}, \frac{1}{4}], -\frac{1}{4}), \sigma = -1$$

$$s_3' = ([-\frac{1}{4}, \frac{1}{4}], -\frac{1}{4}, -\frac{1}{4}), \sigma = -1$$

$$s_4' = ([-\frac{1}{4}, \frac{1}{4}], \frac{1}{4}, -\frac{1}{4}), \sigma = 1$$

Second Iteration

As before, there is no face subdivision required, and only one element, s'_1, is passed to L_1. Then the 2 final sub-faces are generated (from s'_1):

$$s''_1 = (-\frac{1}{4}, -\frac{1}{4}, -\frac{1}{4}), \sigma = -1$$

$$s''_2 = (-\frac{1}{4}, \frac{1}{4}, -\frac{1}{4}), \sigma = 1$$

Final Computation

The final array L now contains just two points: s''_1 and s''_2. Only $f_3(s''_1)$ is positive, and so $deg(\mathcal{F}, \mathbf{X}, 0) = -1$, since that is the orientation of the only remaining face. This is the sign of the Jacobian determinant of the only root of $\mathcal{F}$ in $\mathbf{X}$, $(0, 0, 0)$.

This simple example serves as a basic illustration. Here, however, no face subdivision or overlap resolution is required; in practice these requirements complicate things greatly, so that a solution by hand becomes excessively longwinded and a solution by computer may require non-trivial time.

5 Systems of Polynomial Equations

In this chapter we mainly address systems of nonlinear equations (briefly introduced in Section 1.1) where the component functions are polynomials, which constitute a wide subset of such problems. We consider what it means to 'solve' such a system, and we survey existing methods of solution. An overview of real-world applications in which polynomial (and nonlinear) systems commonly arise is also given.

A system of polynomial equations can be written in the form $\mathcal{P} = 0$, where

$$\mathcal{P} : \mathbb{R}^n \to \mathbb{R}^n$$

comprises n polynomials p_i, $i = 1, \ldots, n$, in the real variables $x_1, \ldots, x_n$. In the most general case, it is desired to determine the *solution set*, i.e. the set consisting of all solutions to the equations, viz.

$$\{x \in \mathbb{C}^n \mid p_i(x) = 0,\ \forall i = 1, \ldots, n\}. \tag{5.1}$$

In many (but by no means all) cases, complex solutions do not correspond to real-world solutions; here, we can restrict the search space for the solution set to $\mathbb{R}^n$. Similarly, solutions where some or all of the variables have negative (or zero) values may not correspond to solutions of interest. In many cases, we seek only one real-world solution, which has to be distinguished from the one or more spurious solutions accompanying it, which may have negative or complex components (see Subsection 5.1.1 for such an example).

We may often further restrict the search for solutions to a particular box $\mathbf{X}$ in $\mathbb{R}^n$. Here the solution set is

$$\{x \in \mathbf{X} \mid p_i(x) = 0,\ \forall i = 1, \ldots, n\}. \tag{5.2}$$

There are several cases where non-polynomial systems can be converted to polynomial systems (in $\mathbb{R}^n$) with a relatively straightforward variable substitution. Systems with complex-valued polynomials can be rewritten simply by separating each variable and each polynomial into their real and imaginary parts, doubling the number of variables. Functions where powers of trigonometric functions appear can often satisfactorily be converted to polynomials by employing a simple substitution for those variables which represent angles, as can trigonometric polynomials. In this case the relationship between the sine and the cosine of a quantity ($\sin^2(\theta) + \cos^2(\theta) \equiv 1$) can be expressed as a quadratic equation.

These problems are subject to attack by various means. A solution by hand may involve clever manipulations, intuitive leaps, or even guesswork. A computer program or algorithm (commonly called a *solver*) is more likely to rely on set strategies and repetetive calculation. The classic means of solving systems of equations by hand involves variable substitution and algebraic manipulation. This is only feasible when the total degrees of the component polynomials are low. In fact, due to the classic result of the insolubility of the quintic (the impossibility theorem of Abel), we know that, in general, algebraic solutions to polynomial equations of degree five or greater are impossible.

5.1 Applications

To date, fields of application where problems may be presented as systems of nonlinear equations include (but are not limited to) the following. Some of these applications only involve systems of polynomial equations. Further examples can be found in the monograph [Mor87].

The number of equations, n, is typically small — less than 10 — although this can vary widely for certain problems.

- **Astronomical Calculations**: The behaviour of an orbiting system of astronomical bodies is governed by a system of equations. Typical problems include intercept planning or scheduling, satellite tracking, and collision checking. Under the classical theory of Laplace, the determination of an orbit can result in a degree 8 polynomial [MG$^+$08].
- **CAGD (Computer-Aided Geometric Design)**: CAGD involves the modelling of a real or virtual three-dimensional scene on a computer, cf. [Far02]. The objects in the environment are typically constructed from a set of *primitives* — simple geometric objects such as spheres, cones, etc. An example of such a CAGD package is *SvLis* [Bow99]. Surface intersection or shape interrogation problems in the model thus present themselves as a system of equations, where each primitive is governed by one equation. Systems of polynomial equations appearing in geometric intersection computations are considered in [HMSP96]. Ray-tracing problems are of a similar type. Such problems are usually comprised of polynomial equations of low degree. In a model, two surfaces will very commonly just touch, which is an inherent cause of numerical ill-conditioning. This makes them unsuitable for solution by non-robust methods.
- **Chemical Engineering**: Process design problems in chemical engineering can be formulated as systems of nonlinear equations, and interval Newton methods can tackle some of them successfully [BS95].
- **Chemical Equilibrium Problems**: Complex chemical systems may comprise a number of different substances which inter-react according to given formulae, where each variable represents the amount of each substance. A point of equilibrium of the chemical system is found as a solution to a system of equations [MM90]. These problems are typically polynomial systems of low degree. Combustion problems to determine flash points are similar.
- **Dynamics**: Problems in dynamics are often presented as systems of equations. The types of equations may vary, e.g. polynomials for motion under gravity, sinusoidal

functions for simple harmonic motion, etc. In [DS09], polynomial dynamic systems are considered and the bound function from Section 10.2.4 based upon the Bernstein expansion is employed.

- **Economical Sciences**: Economic systems can be modelled as a series of interacting variables and numerous problems can then be posed as systems of equations to solve. Cost functions are often polynomial and many cost analysis problems therefore consist of systems of polynomial equations. An elementary example of a polynomial arising in an investment decision problem is given in [Jud85]. Several further examples can be found in [Mor87].
- **Electrical Engineering**: Electric current network equations arise in circuit design [NR08]. It is desired solve the network equations to find the electrical current through every component (equivalently, the voltage at every connection). Many circuit elements exhibit linear behaviour, in which case the network equations comprise a linear system, but often there are nonlinear components such as semiconductors. Where there is uncertainty in component tolerances, interval analysis may be applied [Dre05].
- **Mechanical Systems**: The configuration of a mechanical device consisting of connected components is described by a system of (often polynomial) equations. An example from [Mor87] is the six-revolute-joint problem; a valid configuration may be sought as a solution to the system. Truss and frame structures, consisting of structural elements connected at nodes, may be placed under loading, and it is desired to compute the resultant displacements and stress/strain forces. Application of the *finite element method* results in a (usually large) system of linear equations; where uncertainty in loading forces and positions is modelled, polynomial parametric dependencies may be added [GPS12].
- **Neurophysiology / Neural Networks**: Simple systems of connected and interdependent firing neurons can be modelled by a system of equations with low-degree polynomials, typically large and sparse. There may be many neurons, but they are not connected to all others, and the behaviour of each is simple. Neural networks can be modelled similarly [Noo89].
- **Robot Motion Planning / Kinematics**: The motion of a free-standing or wheeled robot, or of a robot arm with a fixed base, may be constrained by the presence of obstacles or other physical limitations. The robot may move in a number of dimensions and the set of all valid positions is called the *configuration space*. Each constraint can be expressed as an inequality which must be satisfied. To plan a motion from a given starting point to a desired end point, we must seek a valid path in the configuration space. In some cases, we may seek an optimal path (i.e. the shortest/quickest route), or we may question the existence of a valid path. This is also called the *piano movers problem*. In cases such as these, the problem formulation can be modified to a system of equations. One example problem concerns the inverse kinematics of an elbow manipulator [HS95b].

5.1.1 Example System

The amounts $x_1, \ldots, x_5$ of 5 substances at the chemical equilibrium point of a particular hydrocarbon combustion [MM90] occur as roots of the following system of equations:

$$\begin{aligned}
p_1 &= x_1x_2 + x_1 - 3x_5, \\
p_2 &= 2x_1x_2 + x_1 + 0.0000019230x_2^2 + x_2x_3^2 + 0.00054518x_2x_3 \\
&\quad +0.000034074x_2x_4 + 0.00000044975x_2 - 10x_5, \\
p_3 &= 2x_2x_3^2 + 0.00054518x_2x_3 + 0.386x_3^2 + 0.00041062x_3 - 8x_5, \\
p_4 &= 0.000034074x_2x_4 + 2x_4^2 - 40x_5, \\
p_5 &= x_1x_2 + x_1 + 0.00000096150x_2^2 + x_2x_3^2 + 0.00054518x_2x_3 \\
&\quad +0.000034074x_2x_4 + 0.00000044975x_2 + 0.1930x_3^2 + 0.00041062x_3 \\
&\quad +x_4^2 - 1.
\end{aligned}$$

There are 4 real and 12 complex roots. The real roots are:

$$\begin{gathered}
(0.0027572, 39.242, -0.061388, 0.85972, 0.036985), \\
(0.0021533, 50.550, -0.054145, -0.86067, 0.037001), \\
(0.0031141, 34.598, 0.065042, 0.85938, 0.036952), \\
(0.0024710, 43.879, 0.057784, -0.86021, 0.036966).
\end{gathered}$$

The root with wholly positive components corresponds to the real-world solution to this problem.

5.2 Types of Solutions

We first need to consider the *dimensionality* of the solution set. In underdetermined or degenerate systems of equations, this dimension is either greater than zero (i.e. there are infinitely many solutions) or else the system is inconsistent (i.e. there are no solutions). A solution set of non-zero dimensionality can be given an algebraic or geometric description (or approximation), but this is usually rather different to the procedure of finding point solutions. Hereafter, we do not consider the case where the number of variables and the number of equations are unequal, and assume that the dimension of the solution set is zero, in other words, that there are a finite number of unique solutions.

In this case, there may be one or more values x in the domain set which satisfy $\mathcal{P}(x) = 0$; these are the solutions and we wish to find them. This simple description, however, is incomplete in several details — we need to be more specific in order to fully describe what we mean by 'solution'.

The presentation of solutions may be characterised by two main attributes:

- **Form of solutions**

  In increasing order of desirability, a solution may either be given implicitly, as a floating-point approximation within a specified tolerance, or, ideally, as a closed-form expression or an exact number. A solution expressed implicitly might be regarded as merely a refinement of the problem, and not a proper solution; however if the implicit form is simpler than the system itself, and perhaps coupled with an existence proof or bounds, it would go a fair way towards satisfying the demands of 'solution'. Alternatively, if floating-point results are given, what level of precision is acceptable?

- **Choice of solutions**

  Do we wish to find all solutions that exist within the domain, all solutions within a specified region of interest, only one solution from within a specified region, or will any solution at all suffice? If only one solution is required from several, does it matter how the chosen solution is designated? For example, in some cases, we may seek the solution nearest to a specified point.

The answers to these questions will depend largely on the application. A safety-critical engineering problem will likely require a numerical approximation provably within tightly-constrained bounds. All solutions within a specified region may be required. Algebraic problems may require exact algebraic answers, which can only be provided by symbolic computation (by machine or hand), not by numerical computation. To satisfy a mathematical proof, a guarantee of existence of a single solution (or a guarantee of non-existence of any solutions) within a specified region may suffice.

All these variations do occur, but the most common category of problem that arises in practical applications seems to be that of finding good numerical approximations to all the solutions within a specified region of interest. The required precision is typically between 3 or 4 significant figures up to as accurate as the machine arithmetic will allow, although some problems require arbitrary precision arithmetic (also sometimes called *range arithmetic*, cf. [AS92]).

It may be that there exist no solutions to the system within the region in question; in this case a procedure which verifies this fact and terminates may be considered to have satisfied the requirements of the solution process.

5.3 Methods of Solution

The classic numerical technique for solving a system of equations is the Newton method (see Subsection 5.3.2). This yields a single solution and requires a sufficiently good initial approximation and a numerically well-conditioned problem. In such cases it usually works well. Today, there are many more methods available; amongst numerical methods there are many refinements of Newton's method, there are techniques based upon symbolic manipulation, and interval and homotopy methods. A general-purpose, high-performance infallible solver for nonlinear systems remains an elusive goal, although robust polynomial system solvers are becoming increasingly successful.

Most authors, e.g. [SP93], place methods for the solution of polynomial systems into three categories: *elimination methods*, *continuation methods*, and *subdivision methods*, where the latter category includes interval-based techniques. In the remainder of this chapter we review each of these approaches and some specific solvers from the literature. A discussion of techniques for the solution of systems of nonlinear equations is given in the monograph [Sik97].

Firstly we outline a scheme for categorisation in Subsection 5.3.1, and then we broadly classify and consider the existing solvers in Subsections 5.3.2 to 5.3.7. Unless otherwise noted, we assume that we have a polynomial system $\mathcal{P} = 0$ where the number of equations and the number of variables is both n, and where we consider the solution set in the form of (5.1) or (5.2).

5.3.1 Categorisation of Methods

One problem encountered in attempting a thorough review of existing polynomial and nonlinear solvers is that there is no universal standard for nomenclature and classification. It is sensible first to examine some common features of solvers and equip ourselves with some terminology. For such a classification we can identify (at least) five key distinguishing attributes:

Scope

A *global* method will find all solutions within a designated region of interest, but a *local* method will only find one such solution, which may be designated in a seemingly arbitrary fashion. Some solvers are only designed for polynomial systems; others may handle any types of nonlinear function for which a computer implementation of their evaluation can be achieved.

Robustness

A solver is *robust* if it is not subject to numerical instability. Robustness might be achieved, for example, by the use of interval arithmetic or if the solver operates on an exact symbolic representation, although these approaches have their own computational disadvantages.

Theoretical Basis

Of course, an acceptable solver must have a theoretical basis to support it, i.e. a proof that it works. Existing methods tend to be grounded either in the fields of numerical analysis or algebraic geometry.

Internal Data Type

The basic data type(s) upon which the necessary manipulations are performed may be floating-point numbers, rational numbers, intervals, symbolic data, or a mixture thereof.

Solution Process Structure

What computational strategies may be adopted to find the solution(s)? Commonly observed are *iteration* (repetition of a computation upon an initial approximation to achieve a sequence of converging values), *recursion* (obtaining a result as the combination of a number of similar problems of lower degree or dimension), *subdivision* (repetition of a computation on successively smaller portions of the domain to achieve greater accuracy and isolate solutions), and *continuation* (construction of a homotopy to a known function, which exhibits branching and path-tracing). These categories are not mutually exclusive, as any given method may utilise a number of different computational techniques.

5.3.2 Newton Methods

The obvious starting point in this review is the classic Newton method. Given our classification above, we may describe this as an iterative non-robust local numerical solver.

Given $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$, an initial approximation x_0 to a solution of $\mathcal{F}(x) = 0$ is refined by the iteration

$$x_{i+1} \; = \; x_i - J^{-1}(x_i)\mathcal{F}(x_i), \tag{5.3}$$

where J is the Jacobian matrix of $\mathcal{F}$.

In many cases this method can be used as a simple and effective solver. If successful, a quadratic rate of convergence is achieved. There are, however, a number of limitations:

- As a local method, root designation is arbitrary. This is a big problem if no information concerning the placement of all solutions is available. The iteration may converge to a solution other than the one desired, which might be unanticipated and unwanted.
- The method will fail for numerically ill-conditioned problems, being sensitive to the behaviour of the partial derivatives of the component functions of $\mathcal{F}$ in the vicinity of a solution. Singular solutions cannot be found.
- The choice of the initial approximation to a desired solution must be sufficiently good, otherwise the sequence of approximations will either converge to an alternative solution, not converge at all (oscillating or diverging to give no useful result), or exhibit extremely slow convergence.

Many categories of problems (e.g. CAGD) are thus inappropriate for solution by Newton's method. If the actual existence of a solution is in question, as is the case for many abstract problems (e.g. geometric intersection), there may well be no choice of the initial value possible other than a guess, so that the sequence of iterates generated is unlikely to be satisfactory.

To apply Newton's method with rigour, one requires a verification that a solution exists and is isolated within a given region. It is possible to determine a (possibly small) neighbourhood within which any choice of initial approximation is guaranteed to yield a correctly-converging sequence. This neighbourhood, and convergence bounds, can be computed (for example by application of Kantorovich's theorem), based on the values of the partial derivatives.

There exist numerous algorithmic improvements and modifications to Newton's method, which improve robustness or convergence properties, but the fundamental limitations remain. Most of these methods have been in use for some time, and well-established suites of software exist; the Numerical Algorithms Group (NAG) library [NAG] claims to be the largest. It contains a large selection of routines which will cater for problems with various numerical difficulties. However, precise knowledge in advance of the properties of the system is needed in order to choose the correct routines. In practice, categories of well-understood and numerically well-behaved problems where a single specified solution is required can generally be solved by an expert user.

5.3.3 Interval Newton Methods

Broadly speaking, the interval Newton method is an adaption of the Newton method obtained by replacing the real (floating-point) variables in (5.3) by intervals (see Section 2.1). This requires tight interval extensions for the component functions and their partial derivatives. The initial approximation is replaced by a *starting interval* (in the univariate case) or a *starting box* (in the multivariate case), and each iteration delivers a new interval or box, which is intersected with it. The central result here is that if the new iterate is contained within the previous approximation, i.e. the new interval or box is a proper subset of its predecessor, then the existence of a unique solution inside is guaranteed. In this case a successively narrowing sequence of bounding intervals or boxes for this solution is obtained. An iterative non-robust local or global interval solver can be thus constructed.

It is not quite this straightforward, however. One may be confounded by the fact that, now using interval arithmetic, convergence is not achieved. Convergence now depends on the non-ideal interval arithmetic used, and the dependency problem (see Subsection 2.1.4) will likely arise. Especially since the interval computation involves the partial derivatives as well as the function itself, in some cases it may be rather hard to achieve a narrowing interval sequence, without a sufficiently tight interval or box in the first place. It may be appropriate to employ subdivision to tighten these bounds, albeit at greater computational cost. Where convergence is achieved, however, it is quadratic, as per the Newton method.

The interval Newton method is subject to the same limitations in scope as before, i.e. it requires a numerically well-conditioned problem with a good-quality starting approximation. The efficiency (or overestimation) of the underlying interval arithmetic is a further limitation. However it does possess some of the advantages of other subdivision-based methods described in Subsection 5.3.6. Versions of the interval Newton method are able to solve some categories of problems with speed. It is also quite suitable merely as a *verifier*, offering existence proofs that are superior to those based on the traditional Newton method.

Improvements to the method include the use of the Krawczyk operator and the Hansen-Sengupta operator, amongst others. For a more detailed treatment of types of interval Newton method and these operators, see [RR88, Neu90, Sta95, Gra07].

5.3.4 Elimination (Symbolic) Methods

These are global methods with symbolic data which are often recursive in nature. By operating symbolically on the component equations, they are not defeated by numerically ill-conditioned problems. The hurdles which symbolic methods must overcome are *intermediate expression swell* and restrictively high orders of complexity. The former is the commonly-encountered phenomenon in which complicated intermediate data (such as high-degree polynomials or very large integers) are formed from relatively simple input data, which can drastically slow a computation or halt it altogether. The latter often arises from the recursive expression of the solution in terms of a number of smaller (with respect to either degree or dimension) problems. This typically leads to exponential orders of growth. Elimination methods are usually restricted to the case of polynomials with rational coefficients, since they rely on properties such as factorisation and divisibility.

In the case of systems of polynomial equations, the most common techniques are based upon the computation of a *Gröbner basis* for the system. We begin with a few definitions:

Definition 5.1 (Polynomial Ideal)**.** *The* ideal *of (generated by) a set of polynomials $p_1, \ldots, p_n \in \mathbb{Q}[x_1, \ldots, x_n]$ is the set of all linear combinations of the p_i with polynomial coefficients, i.e.*

$$\langle p_1, \ldots, p_n \rangle \;=\; \{q_1 p_1 + \ldots + q_n p_n \mid q_1, \ldots, q_n \in \mathbb{Q}[x_1, \ldots, x_n]\}\,.$$

The polynomials which comprise a system of equations can be viewed as forming an ideal. An alternative basis for the same ideal will have the same solution set. We then require a basis from which the solution set can be mechanically constructed.

Definition 5.2 (Polynomial Reduction)**.** *A polynomial p is* reduced *with respect to a set of polynomials S if all the leading monomials of polynomials in S do not divide the leading monomial of p.*

Where p is not reduced with respect to S, a suitable multiple of a polynomial from S can be subtracted to eliminate the leading monomial. This definition needs to be furnished with an admissible relational operator for polynomials; a suitable example ordering is the lexicographic order. We omit the details of what constitutes such an ordering here for brevity; for details see [DST93].

Definition 5.3 (Gröbner (Standard) Basis)**.** *A basis (generating set) S of a polynomial ideal I, with respect to a given ordering, is a* Gröbner basis *if every possible reduction of an element of I to a polynomial reduced with respect to S yields zero.*

The essence of the idea is thus to rewrite the system of polynomials into a standard basis, by a process of reduction. B. Buchberger's algorithm [Buc70] is the classic method. Given such a basis, broadly speaking, the solution set is then elaborated by first finding an expression for the zeros of the least (with respect to the ordering) polynomial and a mechanism of successive substitution. A more complete treatment is to be found in [DST93, Chapter 3], cf. [Win96, Chapter 8].

Intermediate expression swell can be observed both in the degrees and number of polynomials; the Gröbner basis may be larger than the starting set of polynomials. The average-case complexity of Buchberger's algorithm is exponential in the number of variables n. At worst, it is doubly exponential.

In the case of more general systems of equations, other known methods of solution include Wu stratification [Wu84, Ric99], with origins in the work of W.-T. Wu, based similarly on *rewriting*, and retract or cylindrical decomposition [Kan96], from algebraic geometry. Intermediate expression swell is known to be a problem for these methods, but experimental software exists. An overview of these methods for solving elementary systems is presented in [Ric96]. More recent work includes the triangular decomposition [CD$^+$10] and J.-C. Faugère's improved algorithms for the computation of Gröbner bases [Fau02]. A recent summary and historical overview of symbolic methods for solving systems of polynomial equations is given in [Laz09].

The collaborative Polynomial System Solving project [POS95] implemented several variants of Buchberger's algorithm as a major component of the *PoSSo* C++ software libarary. This software proved effective for certain categories of industrial applications. Such solvers may succeed where numerical algorithms fail, but due to the high computational overheads associated with symbolic computation, high-dimension problems typically prove prohibitively expensive. Packages for computing Gröbner bases, based both on Buchberger's algorithm and also a variant of Faugère's method, are included in the computer algebra system *Sage* [SJ05]; other computer algebra systems include similar implementations.

5.3.5 Continuation (Homotopy) Methods

These methods, which are generally only applicable for systems of polynomial equations, are a result of the combination of theory from algebraic geometry with numerical analysis techniques. They can be classified as global solvers with numeric data.

Definition 5.4 (Homotopy)**.** *Let $A, B \subset \mathbb{R}^n$ and $\mathcal{F}, \mathcal{G} : A \to B$ be continuous. A* homotopy *from $\mathcal{F}$ to $\mathcal{G}$ is a continuous function*

$$H : [0,1] \times A \to B$$

(jointly continuous in both arguments) with $H(0,x) = \mathcal{F}(x)$ and $H(1,x) = \mathcal{G}(x)$, $\forall x \in A$. Two functions are said to be homotopic *if there exists a homotopy between them.*

In other words, $H(t,x)$ describes a gradual transformation (a change in a continuous fashion) from $\mathcal{F}(x)$ to $\mathcal{G}(x)$ as t transits from 0 to 1.

The process of homotopy continuation is broadly as follows: A given system of n polynomial equations in n variables $\mathcal{P}(z) = 0$, where $z \in \mathbb{C}^n$, is extended to a system of $n+1$ variables by the addition of a homotopy parameter t. It is desired that the new system $H(t,z) = 0$ has the property that $H(1,z) = \mathcal{P}(z)$ and that the solutions to $H(0,z) = 0$ are known. The two (non-trivial) stages are then firstly defining the homotopy function H, and then applying an appropriate numerical method to track the paths defined by $H(t,z) = 0$ as t varies from 0 to 1.

The major problems which seem to be encountered involve the large number of paths to trace, and paths which diverge to infinity. This method also cannot be used to determine singular solutions, and other numerical difficulties may arise. The number of paths to be traced (and hence the complexity) is sensitive to the total degree of the system.

An overview of early continuation techniques was given in [Mor87]; detailed treatment may also be found in [AG90, AG97]. A continuation scheme utilising the topological degree for existence tests is proposed in [BFLW09].

HOMPACK [MSW89], probably the first major implementation of a polynomial continuation method, had difficulties in solving even small systems. It has since been refined, benefiting from both algorithmic improvements and advancements in available computing power. Thereafter, *PHCpack* [Ver99], met with more success, having solved a catalogue of certain problems with n upto 12. A more recent FORTRAN 90 software package is *HOM4PS*; it has been tested in comparison with these predecessors [LLT08].

5.3.6 Subdivision Methods

Where they succeed, elimination and continuation methods often deliver more information than is needed, since they typically determine all complex solutions of the system, whereas for many applications only the solutions within a given area of interest are required. With a subdivision, or branch-and-bound, method (see Subsection 2.3.1), we may ignore spurious solutions outside the specified region without wasting computational effort. The starting box is successively partitioned into smaller sub-boxes; those sub-boxes which are infeasible (i.e. they cannot contain a solution) may be eliminated by an exclusion test, e.g. using bounds for the ranges of the polynomials under consideration over them; where at least one component polynomial has a range which excludes zero, infeasibility is assured. Such a method ends by producing zero or more sub-boxes of a suitably small size which contain all solutions to the system within the specified starting box. They are thus also suited to the determination of isolating regions for solutions.

Solvers based upon subdivision methods are generally global and robust. The main challenge lies in achieving a subdivision sequence which terminates, and in a reasonable time. Without an effective pruning step, the search tree of sub-boxes generated can become excessively large. Here, tools such as interval computation techniques and the Bernstein expansion may be employed, exploiting their beneficial bound convergence properties. Straightforward interval evaluation, however, tends not to perform well, often yielding bounds that are too wide or which do not converge sufficiently quickly. Some manipulations can be beneficial for achieving tighter bounds, for example a conversion to Horner (nested) form in the case of polynomial systems, or various kinds of interval enclosures (introduced in Section 2.2).

Of course, methods for solving general systems of nonlinear equations can be applied to polynomial systems, too. The method of A. Eiger et al. [ESS84], cf. Subsection 4.3.2, was based on a pseudo-computation of the topological degree upon simplices. Having a high order of complexity, it was only applied succesfully for some small-dimension systems. Interval computation schemes for general systems, cf. [RR88, Neu90, Kea96b], can likewise be applied to polynomial systems. Some variants of the interval Newton method can also be classified as subdivision methods. V. Stahl investigated a number of interval arithmetic methods [Sta95], based on linear and nonlinear tightening, and has applied them to inverse kinematics problems in the field of robotics. A further approach using intervals is given in [VHMK97].

There seem to be relatively few branch-and-bound methods which are specific to polynomial systems. A couple of methods which apply the expansion of a multivariate polynomial into Bernstein polynomials (see Chapter 3) have been proposed [SP93, FL00]. In the scheme of Sherbrooke and Patrikalakis [SP93], sequences of bounding boxes for solutions are generated firstly by projecting control polyhedra onto a set of co-ordinate planes and secondly by utilising linear programming. However the relationship between the Bernstein coefficients of neighbouring sub-boxes, cf. Subsection 3.3.2 is not exploited, nor is an existence test employed. The method is further developed with preconditioning and the reduction scheme is augmented with a univariate solver in [MP09].

The development of a subdivision-based solver based on the Bernstein expansion, coupled with existence testing, forms the subject of Chapter 8.

5.3.7 Combined Methods

A given polynomial system whose degree and/or number of equations is relatively modest can nowadays typically be solved, given the right choice of method. Whether or not the polynomials are dense or sparse, or whether the problem is ill-conditioned, for example, are factors which influence the best choice. The task remains to automate the selection of the optimal method, and to improve the speed and extend the scope of such solvers.

The theory of solving polynomial systems is generally segregated into the three main categories (Subsections 5.3.4, 5.3.5, and 5.3.6), with little crossover. However, given the strengths and weaknesses of each type of solver — Newton-based methods don't always work, whereas symbolic methods tend to have an inherently high structural complexity, for example — it seems sensible to combine methods, if in some way possible. There have been a couple of notable attempts: Jäger and Ratz [JR95] combined the method of Gröbner bases with interval computations. A NAG-led collaborative project, *Framework for Integrated Symbolic-Numeric Computation (FRISCO)* [FRI99], aimed to integrate a number of techniques (both symbolic and numeric) in the development of a fast black-box solver suitable for multi-purpose industrial applications.

There are more solvers for general systems of equations, cf. Subsections 5.3.2 and 5.3.3, than for polynomial systems alone. The drawback of general-purpose solvers is precisely their generality: a single method that could succesfully tackle any nonlinear system would seem to be all but impossible to achieve. The Newton and interval Newton methods and their variants remain the most widely used general nonlinear solvers, although it appears there is no consensus on an optimal choice of method. Given the wide diversity of problem types (both polynomial and non-polynomial), it seems doubtful that there *is* a universally good or optimal approach.

6 Problems Involving Polynomial Inequalities

We conclude the first part of this thesis with a brief overview of systems of polynomial inequalities, and problems which may include polynomial inequalities, such as global optimisation problems. The Bernstein expansion (introduced in Chapter 3) has previously been applied to robust stability problems, cf. [Gar00], and to the solution of systems of polynomial inequalities [GG99b]. Constraint satisfaction problems typically consist of both equations and inequalities. In [Gra07], for example, interval analysis (introduced in Chapter 2) is applied within a branch-and-bound scheme for their solution.

6.1 Systems of Polynomial Inequalities

Dynamical systems arising in control theory are commonplace, occurring in application areas such as manufacturing plant control, digital control, guidance systems for rockets and missiles, and automobile cruise control. Numerous problems in control system design and analysis, often problems of robust control and robust stability, may be recast and reduced to systems of inequalities involving multivariate polynomials in real variables. This corresponds broadly to the following problem:

Let $p_1, \ldots, p_m$ be polynomials in $x = (x_1, \ldots, x_n)$ and let a box $\mathbf{X}$ in $\mathbb{R}^n$ be given. We wish to determine the *solution set* $\mathbf{\Sigma}$ of the system of polynomial inequalities, given by

$$\mathbf{\Sigma} := \{x \in \mathbf{X} \mid p_i(x) > 0,\ i = 1, \ldots, m\}. \tag{6.1}$$

In general, it is not possible to describe the solution set $\mathbf{\Sigma}$ exactly; instead a good approximation to it is sought. A subdivision scheme (i.e. branch-and-bound scheme, cf. Subsection 2.3.1), in which the box $\mathbf{X}$ is repeatedly partitioned into sub-boxes and bounds for the ranges of the component polynomials thereupon are sought, may typically be employed. A number of techniques from the theory of interval analysis (cf. Chapter 2), or else the Bernstein expansion (cf. Chapter 3), may be employed. In the latter case, the Bernstein coefficients of each p_i, $i = 1, \ldots, m$, can be computed for each sub-box, and the range enclosing property (3.17) used to determine whether each polynomial is strictly negative, strictly positive, or neither over the box. Sub-boxes should be subdivided until either strict positivity for all polynomials or non-positivity for at least one polynomial is satisfied, or until the sub-box volume or maximum sub-box edge length falls below a certain minimum $\varepsilon > 0$, which specifies a satisfactory level of 'resolution' for the final box-union approximation of the solution set.

An inner approximation (underestimation) of $\mathbf{\Sigma}$, and for its complement, and an approximation of the boundary of $\mathbf{\Sigma}$, respectively, may be labelled and defined as follows:

- $\mathbf{\Sigma}_i$: an inner approximation (underestimation) of $\mathbf{\Sigma}$, consisting of the union of sub-boxes of $\mathbf{X}$ on which all polynomials p_i are positive.
- $\mathbf{\Sigma}_e$: an inner approximation of the exterior (complement) of $\mathbf{\Sigma}$, given by the union of sub-boxes of $\mathbf{X}$ with the property that on each there is at least one non-positive polynomial p_{i^*}.
- $\mathbf{\Sigma}_b$: an outer approximation (overestimation) of the boundary $\partial\mathbf{\Sigma}$ of $\mathbf{\Sigma}$, consisting of the union of sub-boxes of $\mathbf{X}$ on which all polynomials p_i attain positive values, but on which at least one polynomial also attains non-positive values.

The union of $\mathbf{\Sigma}_i$ and $\mathbf{\Sigma}_b$ forms an outer approximation (an overestimation, or guaranteed enclosure) for $\mathbf{\Sigma}$. An example of these three sets and corresponding sub-boxes is depicted in Figure 6.1.

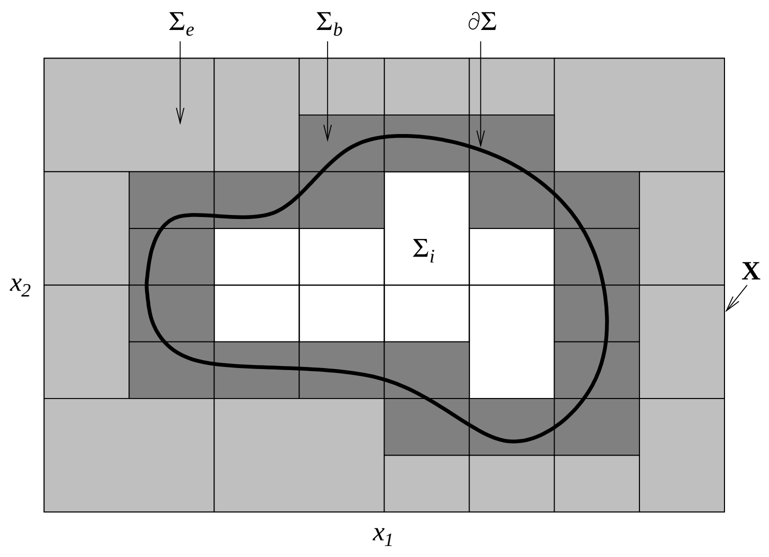


Figure 6.1: Approximations of the solution set $\mathbf{\Sigma}$ and its boundary $\partial\mathbf{\Sigma}$ over a box $\mathbf{X}$ in $\mathbb{R}^2$.

Details of a branch-and-bound algorithm utilising the Bernstein expansion and its application to the computation of $\mathcal{D}$-stability regions (see below) are given in [GG99b] and [Gar00].

In the context of robust control problems, a common formulation is as follows: we are presented with the problem of determining the $\mathcal{D}$-stability region of a family of polynomials within a given *parameter* box $\mathbf{Q}$. For a given polynomial family p in x with a tuple of parameters q (6.3), this is the set

$$\{q \in \mathbf{Q} \mid p(x,q) \neq 0,\ \forall x \notin \mathcal{D}\}\,. \tag{6.2}$$

This problem can be reformulated as a system of polynomial inequalities. We may consider a family of polynomials

$$p(x,q) \;=\; a_0(q)x^m + \ldots + a_{m-1}(q)x + a_m(q), \tag{6.3}$$

where the coefficients depend polynomially on n parameters $q = (q_1, \ldots, q_n)$, i.e., for $k = 0, \ldots, m$,

$$a_k(q) \;=\; \sum_{i_1,\ldots,i_n=0}^{l} a^{(k)}_{i_1\ldots i_n} q_1^{i_1} \cdot \ldots \cdot q_n^{i_n}. \tag{6.4}$$

The uncertain parameters q_i belong to specified intervals

$$q_i \in [\underline{q}_i, \overline{q}_i], \;\; i = 1, \ldots, n. \tag{6.5}$$

These parameter intervals are represented in the form of a box $\mathbf{Q} := [\underline{q}_1, \overline{q}_1] \times \ldots \times [\underline{q}_l, \overline{q}_l]$.

A polynomial p may be termed *$\mathcal{D}$-stable*, where $\mathcal{D}$ is a set in the complex plane, if all the zeros of p are inside $\mathcal{D}$. The *robust $\mathcal{D}$-stability problem* consists of determining whether the family of polynomials $p(q)$ are robustly $\mathcal{D}$-stable for $\mathbf{Q}$, i.e. whether the polynomials $p(q)$ are $\mathcal{D}$-stable for all $q \in \mathbf{Q}$. Three problems of interest of this type are

- Hurwitz stability, where $\mathcal{D}$ is the open left half of the complex plane,
- Schur stability, where $\mathcal{D}$ is the open unit disc, and
- damping, where $\mathcal{D}$ is a sector centered around the negative real axis with its vertex at the origin.

Testing of determinants for positivity (determinantal criteria) may be used in order to reduce the problem to one of strict inequalities involving multivariate polynomials, which may then be solved with a subdivision-based scheme, as outlined above. The use of determinants is generally restricted to systems depending on only a small number of parameters. Several other types of control problems, e.g. static output feedback stabilisation, can also be reduced to the solution of systems of polynomial inequalities. Apart from stability, other performance specifications of control systems may also be modelled in the frequency domain as polynomial inequalities. In [ZG98] a method based on Bernstein expansion is developed which is capable of treating robust Hurwitz stability problems with a larger number of parameters; an application to robust Schur stability is given in [GG99a].

6.2 Constrained Global Optimisation

Many real-world problems have a natural mathematical formulation where their solution is defined as the minimisation or maximisation of a particular function, known as the *objective function*, usually subject to a number of constraints which can be specified by inequalities and equalities involving constraint functions. Physical problems include manufacturing optimisation, logistics optimisation (e.g. the travelling salesman problem), chemical engineering, the engineering verification of mechanical structures, and many more. Numerous abstract mathematical problems, such as curve-fitting or the Kepler conjecture [Hal05], can be formulated as optimisation problems. Some further examples are given in [Zum08].

A constrained global optimisation problem is canonically presented as

$$\min_{x \in F} f(x), \tag{6.6}$$

where the set of *feasible solutions* F is defined by

$$F := \left\{ x \in S \;\middle|\; \begin{array}{l} g_i(x) \leq 0 \text{ for } i = 1, \ldots, t \\ h_j(x) = 0 \text{ for } j = 1, \ldots, s \\ x \in \mathbf{X} \end{array} \right\},$$

and where $S \subseteq \mathbb{R}^n$, $\mathbf{X}$ is a box contained in S, and f, g_i, h_j are real-valued functions defined on S. The function f is referred to as the objective function, the g_i define inequality constraints, and the h_j equality constraints. We have to determine the global minimum over the entire feasible set, not just a local minimum, as is the case for a local optimisation problem.

A frequently used approach is the generation of *relaxations* and their use in a branch and bound framework. Generally speaking, a relaxation of the given problem has the properties that

1. each feasible point of the given problem is feasible for the relaxation,
2. the relaxation is easier to solve than the original problem, and
3. the solutions of the relaxation converge to the solutions of the original problem, provided the maximal width of the set of feasible points converges to zero.

We obtain a relaxation for the global optimisation problem (6.6) by proceeding as follows:

1. Replace the objective function f and the functions g_i by valid lower bounding functions $\underline{f}$ and $\underline{g}_i$, respectively.
2. If the function h_j defining the jth equality constraint is affine, then this equation is added to the constraints that define the relaxation. The remaining equations are rewritten as a pair of inequalities and are treated in the same way as above.

The following optimisation problem is then obtained:

$$\min_{x \in F^{rel}} \underline{f}(x) \tag{6.7}$$

with the set F^{rel} of feasible solutions given by

$$\begin{aligned} \underline{g}_i(x) &\leq 0, \qquad i = 1, \ldots, t', \; x \in \mathbf{X}, \\ h_j(x) &= 0, \qquad j = 1, \ldots, s', \; x \in \mathbf{X}, \end{aligned}$$

where $t' \geq t$ and $s' \leq s$.

The optimisation problem (6.7) is a valid relaxation of (6.6) if its set of feasible solutions F^{rel} satisfies $F \subseteq F^{rel}$ and $\underline{f}(x) \leq f(x)$ for all $x \in F$. The relaxed subproblem with its set of feasible solutions constitutes a simpler type of problem (for example, a linear programming problem) whose solution provides a lower bound for the solution of (6.6).

These relaxations are commonly used for solving constrained global optimisation problems, e.g. [AMF95, AF96]. The computation of a good quality convex lower bounding function for a given function is thus of significant importance when a branch-and-bound approach is employed, such as in the *COCONUT* software package [BSV$^+$01, Sch03]. Tight bounding functions and tight bounds for the ranges of the objective and constraint functions over sub-boxes are crucial for the efficient resolution of the associated subproblems. Convex envelopes, the uniformly tightest underestimating convex functions, exhibit good performance, cf. [AF96, Flo00, TS02].

Because of their simplicity and ease of computation, constant and affine lower bounding functions are especially useful. Constant bound functions are often used when interval computation techniques are applied to global optimisation, cf. [RR88, Kea96b, Han03]. However, when using constant bound functions, all information about the shape of the given function is lost. A compromise between convex envelopes, which in general require significant computational effort, and constant lower bounding functions are affine lower bounding functions.

To generate an affine relaxation for problem (6.6), the functions f, g_i $(i = 1, \ldots, t)$, and h_j $(j = 1, \ldots, s)$ are replaced by affine lower bounding functions $\underline{f}$, $\underline{g}_i$, and $\underline{h}_j$, respectively. Then the relaxed problem (6.7) with the respective set of feasible solutions yields a linear programming problem whose solution provides a lower bound for the solution of (6.6). Affine lower bounding functions are simpler to compute and work with than convex envelopes, preserving basic shape information and yielding linear programming subproblems that are relatively fast to resolve. A sequence of diverse methods for computing such affine bounding functions for polynomials based upon the Bernstein expansion is proposed in Chapter 10.

Apart from relaxations, many branch-and-bound based methods require bounds on the gradients and Hessians of constraint functions, which are preferably computed rigorously. For example, the αBB algorithm, cf. [AMF95, Flo00], relies heavily on such bounds. Rigorous bounds for the partial derivatives of polynomials can also be provided by the Bernstein expansion, see Subsection 3.3.4.

Part II: Contributions

7 Computation of Topological Degree

This chapter consists of a detailed study of the recursive algorithm for the computation of topological degree, an outline of which was proposed by O. Aberth [Abe94]. Topological degree formed the subject of Chapter 4 and this algorithm was introduced in Subsection 4.4, with a simple example. The behavioural attributes in practice (especially the complexity) of Aberth's method are unknown and remain to be investigated. Here we undertake an in-depth analysis of the algorithm and obtain an estimate of its complexity with the use of a geometric model coupled with a probabilistic analysis. The method is to be implemented and tested with a catalogue of examples, and the data is to be compared with the complexity study. Improvements to the algorithm, in particular the face subdivision strategy, will be considered and proposed. The implemented software is described in Appendix A.

To briefly recall from Section 4.4.2, we are given a box $\mathbf{X}$ in $\mathbb{R}^n$ and a continuous function $\mathcal{F} : \mathbf{X} \to \mathbb{R}^n$ given by component real-valued functions

$$f_i(x_1, \ldots, x_n), \quad i = 1, \ldots n, \tag{7.1}$$

for which we assume efficient interval arithmetic implementations are available. Since interval arithmetic is a core low-level component of this method, we do not consider cases where the component functions f_i may have non-standard interval implementations, cf. Subsection 2.1.7. It is desired to compute the Brouwer degree (see Section 4.2) of $\mathcal{F}$ over $\mathbf{X}$ at 0, $deg(\mathcal{F}, \mathbf{X}, 0)$, where the degree is calculated at 0 without loss of generality.

7.1 Open Questions

There are a number of open questions which arise from Aberth's presentation of the algorithm and design issues which need to be addressed, rendering an implementation non-trivial. In some cases a translation from informal algorithm description to precise specification and program code is highly intricate and introduces 'hidden' design choices, for which it is unclear how to proceed. We list these questions here, and explore them further in this chapter.

- **Performance**

  The performance of the algorithm is undocumented; it is only stated that it was implemented. The performance characteristics, range of practical applicability, and complexity are all unknown and require investigation. The dependence of the algorithm upon the performance of the interval arithmetic used is also unclear. For instance, if the interval ranges generated are too wide, a non-terminating subdivision of faces may be a threat.

- **Face Processing Strategy**

  The main workspace of the algorithm consists of an array of faces L, cf. Subsection 4.4.4, which are in the first instance taken from $\mathbf{X}$. There is an issue concerning the ordering of these faces in L, and the manner in which generated sub-faces are added to it. In an abstract sense, it is more accurate to say that the structure of faces which are generated and analysed is a *tree*. There is then a choice of preferential order for face processing. In [Abe94] it is stated that sub-faces are added to the front of L, implying a *depth–first* strategy. It is not clear that this choice is the most efficient. This design choice only affects the non-functional performance, however.

- **Face Subdivision Strategy**

  The given strategy for subdividing faces into sub-faces is that all interval fields of the face are bisected. However, this is not a functional requirement, since any sensible means of partitioning a face into smaller parts might plausibly work just as well. It is possible to subdivide in one or only some dimensions, provided that face edge widths may ultimately reach arbitrary smallness. This generates fewer immediate sub-faces, but increases the likelihood of requiring further subdivisions. Furthermore, it may be that a good choice of subdivision point reduces the number of faces processed overall. A heuristic to choose such a point may be possible. It is unclear if bisection is an optimal choice in this regard. Since we have observed that the sub-faces form a tree, it is clearly desirable to try to reduce both the branching factor and depth of this tree, to minimise the computational effort. The entire computation is based on recursive face generation and processing, thus it seems that the number of faces generated, especially in the early stages, should be a major deterministic factor in (non-functional) algorithm performance.

- **Overlap Resolution Strategy**

  In Subsection 4.4.4 (Reduction of Faces) the necessity to eliminate any overlaps between child faces was explained. In the first instance, there is the question of whether the problem of boundary zeros for sub-problems occurs with sufficient frequency to merit a strategy of resolving overlaps in all cases. If this is indeed necessary, there are issues concerning both how to structure the search for overlaps between face pairs, and how to resolve overlaps once found. We may wish to resolve either by minimising immediate effort, or alternatively by minimising the number of generated faces for later work.

- **Ordering of Variables**

  In the default case we intersect the image of the box with a ray emanating from the origin in the positive direction of x_1 (see Figure 4.4), but this choice is arbitrary. We could choose any coordinate-aligned direction, without essentially changing the algorithm; a choice of initial ray in the direction of a different variable simply corresponds to a variable reordering. It may be possible to make a beneficial choice of ray direction and improve performance. It is simple to provide a pictorial justification for this supposition, at least. In Figure 4.2, if we take a ray emanating in a perpendicular direction to that which is illustrated (i.e. in the direction of the secondary variable), there is no intersection with the box image at all, so it seems that less computational effort would be needed. It may be that, at least in some cases, an optimal reordering of the variables is possible.

We address two of these issues immediately before proceeding further:

7.1.1 Face Processing Strategy

As noted above, in each major iteration of the algorithm, we are presented with an array L of faces which have to be tested (with interval arithmetic function evaluations) and either discarded (negative result), retained and placed in the output array L_1 (positive result), or divided into sub-faces for further testing (ambiguous result). The structure of faces and sub-faces forms a tree. Choices of either prepending or appending new sub-faces to L correspond to processing of the face tree in depth-first or breadth-first order, respectively.

Both breadth-first and depth-first strategies were implemented and compared. A detailed analysis is omitted for the sake of brevity, but this comparison showed clearly that the depth-first choice had a much smaller memory requirement. This is a common observation for many types of tree search algorithms.

The computation time seems to be largely unaffected by this design choice, since the same number of faces need to be processed in either case. The reason for the marked observed difference in memory usage becomes clear once we consider the branching factor of this process; 2^{n-1} sub-faces are generated from each face in the first major iteration of an n-dimensional problem. In the depth-first case, the number of sub-faces which need to be stored (for subsequent analysis) is proportional to the depth of the current sub-face in the tree. In the breadth-first case, however, the number of deferred sub-faces is proportional to the breadth of the tree at the current depth. For given n, the maximum tree depth is the deterministic memory requirement factor, and this requirement grows linearly with depth for the depth-first search, but exponentially with depth for the breadth-first search.

A possible additional advantage of the depth-first strategy is that sub-faces are entered into L_1 in an order which has potential benefit from the viewpoint of searching for overlapping faces (discussed below). In this case, the elements of L_1 are ordered with respect to the original face in L that they are a subset of, i.e. L_1 is constructed as a series of sequential 'blocks' of faces, where all faces within a block were generated by subdivision of the same original face. The probability of two elements of L_1 sharing a common edge is therefore greater if they are in the same block. It is plausible (although highly non-trivial) to imagine that by exploiting this structure, the number of required comparisons when checking the new list L of generated child faces may be lower. In contrast, with the breadth-first strategy the elements of L_1 are ordered with respect to size. There would seem to be no easy way to exploit this structure to optimise the overlap search.

The depth-first face processing strategy is therefore favoured. This is implemented by recursing on all sub-faces before advancing to the next face in the input list L, i.e. by prepending sub-faces to L.

It may also be worth noting that a good choice of face processing strategy also depends on the face subdivision mechanism that is used, since this determines the structure of the sub-face tree. For instance if one only subdivides in a single variable, then the maximum branching factor is drastically reduced (to 2).

7.1.2 Overlap Elimination Strategy

After the subdivision of faces is complete, all those elements positively selected (in L_1) are decomposed into their child faces and entered into a new array L, ready for the next major iteration. Those faces in L_1 that are geometrically adjacent will yield overlapping (or identical) child faces which will, by construction, have opposite orientation.

The first question which is posed is whether a common zero of the reduced subsystem (i.e. $f_i = 0, \ldots, f_n = 0$, where $i > 1$) occurs on the boundary of a face with any sufficient frequency to justify a systematic elimination of all overlapping regions. Although it seems that for categories of randomly-generated problems this phenomenon should occur with zero probability, testing quickly revealed that, for the kinds of simple problems a user may be likely to input (e.g. low degree polynomials with integer coefficients over a unit box), this failure case arises significantly often. A systematic elimination therefore seems fully justified and is implemented. In any case, it may even be that the computational effort required for this elimination is outweighed by the benefits of possibly reducing the total number of faces for later analysis. This is because, although the failure case notionally arises with zero probability, non-critical face overlap occurs more often than not. Whether the overlap elimination reduces the number of faces overall, or generates more, depends on whether an overlap is an exact duplicate (removing two faces) or partial (removing two faces, but generating possibly several more).

In implementing the overlap resolution there are two major design choices. Firstly, how should one structure the search for overlapping faces, and secondly, how should overlaps be best resolved, once found?

Let $\#L$ be the length of the array L to be searched. In the absence of any information concerning the positions of pairs which cannot by construction overlap, a complete check of all pairs of faces, requiring $\frac{\#L(\#L-1)}{2}$ comparisons, is required. We do have some such information; L is composed sequentially of small subsets of child faces with a common parent in L_1, therefore each such face need not be compared with its immediate neighbours. However, eliminating these comparisons would yield only a very minor reduction, which is likely not worth the implementation cost. As discussed in Subsection 7.1.1 above, further information could potentially be derived from the order in which the faces were entered into L_1. Although it certainly seems that a more structured search, using fewer than $\frac{\#L(\#L-1)}{2}$ comparisons, is therefore possible, this is a complex problem; indeed it is not at all clear whether any saving would not be offset by the effort required in generating and processing structural meta-data for L. It was therefore decided to proceed with an unmodified search.

After the search, those faces which are not matched (overlapped) with any other are retained in L, but the others are separated out for the process of removal of duplicates and duplicated regions. In general, new faces must be created to represent the unduplicated portions of any copies. To remove the duplicated portion of a pair of faces, the corresponding component intervals of each must be compared. This is a somewhat involved process which involves determining a number of *split points*: points which are an endpoint of an interval from one face, but which occur inside the corresponding interval of the other. The duplicated region is given by the product of duplicated interval segments, and the new faces are generated as products of intervals formed by breaking up the component intervals for each face at the split points. Due to the intricacies of this process, it occupies a notable portion of the software code, and can take up a fair chunk of the execution time. For brevity, technical coding details are omitted.

There are a number of different possibilities for the number of new faces that are generated. Let m be the common dimension of the overlapping faces. The best case, of course, is when all the interval fields coincide, requiring no new faces. The next-best case is when there is but a single split point, which requires a single new face for the unduplicated region. There are a number of intermediate cases, and the worst case arises when there are 2 split points in all interval fields. Each interval pair here may overlap end-to-end or one may be contained within the other. If all pairs overlap end-to-end, $2(2^m - 1)$ new faces are generated, if one face is entirely inside another, there are $3^m - 1$ new faces. These cases are illustrated in Figure 7.1.

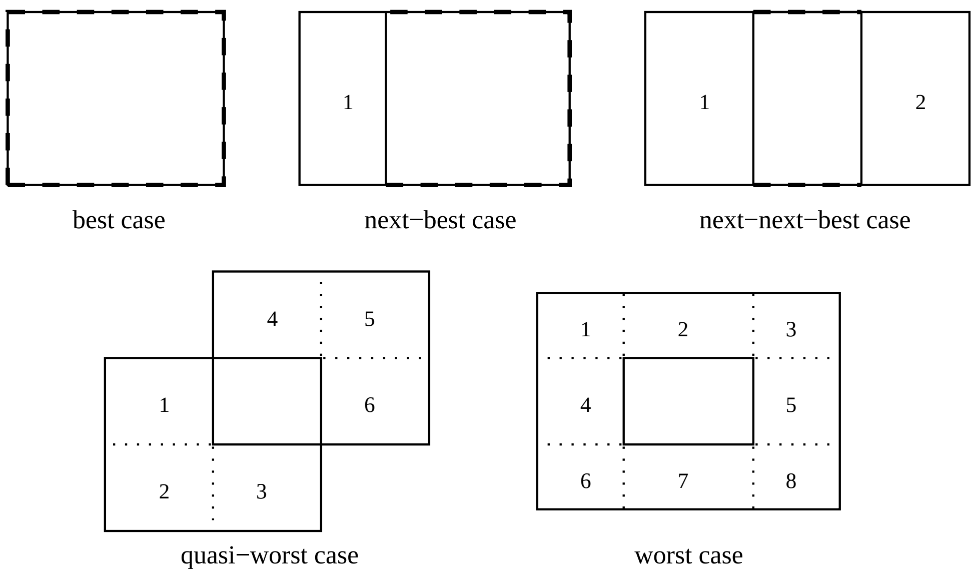


Figure 7.1: Overlapping faces in $\mathbb{R}^2$ and new faces generated (numbered) for the unduplicated regions.

Although this seems the simplest way of generating the new faces, it is certainly possible to reduce the number of these faces, since some of them could be viably merged (i.e. their union is expressible as a product of intervals). Minimising the number of faces is desirable, but this refinement has a significant associated computational cost, so there is a conflict between minimising the effort immediately expended or the number of faces (which may save effort later). Due to the siginificant computational effort already required by the overlap resolution, it was decided not to burden it further.

It is finally worth commenting that this phenomenon of overlap resolution also occurs similarly in geometric modelling systems. For example, the modelling package *SvLis* [Bow99] deals with constructing objects from simple primitives, and can test for overlap of these objects in $\mathbb{R}^3$.

7.2 Analysis of the Algorithm

The analysis and performance of the degree-finding algorithm form the subject material for this section. We take two approaches to the algorithm analysis. Firstly, we perform an abstract analysis, by constructing a simplified geometric model, and secondly we exercise the program with a large number of randomly-generated polynomial problems to construct a data analysis. There are some major questions to be posed:

- What are the factor(s) that determine the computational effort for any particular problem instance?

  There are six candidates identified in answer to this question. We wish to investigate these in order to determine those factors with the greatest degree of influence.

  - Problem dimension
  - Function behaviour within the box
  - Box volume
  - Number of solutions to system within the box
  - Proximity of solution(s) to boundary of box
  - Overestimation in interval arithmetic implementations of functions

- Over what range of problems is the method practically applicable?

  We wish to determine which types of problem are too hard (i.e. for which the time or memory requirement of the computation becomes prohibitive).

- How does this method compare with others?

  A substantive answer here is not possible, given the scarcity of other methods and available data. The numerical quadrature method of O'Neal and Thomas [OT75] is reported to "perform efficiently" for topological degree problems in $\mathbb{R}^6$, although no timing data are provided. In any case, a peformance comparison would be rather unfair, given the relative advantage in computing power which is available today. For their triangulation method, A. Eiger et al. [ESS84] reported only on specific low-degree examples, and discussed the barriers to achieving robustness in their method. We hope that this work will help to contribute to a meaningful comparison with any future topological degree finders that may be developed.

7.2.1 Schematic

Figure 7.2 depicts an overview of the entire degree computation process, starting with an n-dimensional box $\mathbf{X}$, and returning a single integer as the final result.

This illustrates a view of the algorithm as a chain of face processes, where the faces involved are of successively lower dimension after each major iteration. The complexity issue therefore resolves to a question of the number of faces that are generated and processed at each stage. The generation of faces, sub-faces, and child faces, as well as the resolution of overlaps can cause an increase in the number of faces, indicated by the multiplicative factors in Figure 7.2. For a non-trivial computation, the number of faces produced will be significantly larger than the size of the initial array of box sub-faces L, i.e. *intermediate expression swell* is observed.

7.2.2 Metrics and Notation

We wish to measure or estimate each of the following quantities, in order to fully understand in practice the mechanisms of face generation, processing and discard.

- Work–Dimension Distribution

  How many faces are processed in each major iteration? In other words, is the bulk of the computational effort occupied with processing faces of high dimension (early iterations), of low dimension (late iterations), or is the distribution equal?

- Designation Ratio

  The *designation ratio* is defined as the proportion of the terminal (sub-)faces in L that are accepted into L_1. A terminal (sub-)face is one for which no (further) subdivision is performed. We wish to measure this quantity for each major iteration.

- Maximum Subdivision Depth

  The process of face subdivision generates a tree of sub-faces, where those sub-faces at a greater depth have been subdivided more and are therefore smaller. We wish to measure the maximum required subdivision depth for each iteration.

- Branching Factor

  An m-dimensional face that is terminal of course produces 0 sub-faces; otherwise 2^m sub-faces are generated, i.e. there is a fixed branching factor of either 0 or 2^m. We define the average branching factor as the mean number of sub-faces generated per node, averaged over all nodes (including terminal faces, i.e. leaf nodes) in the search

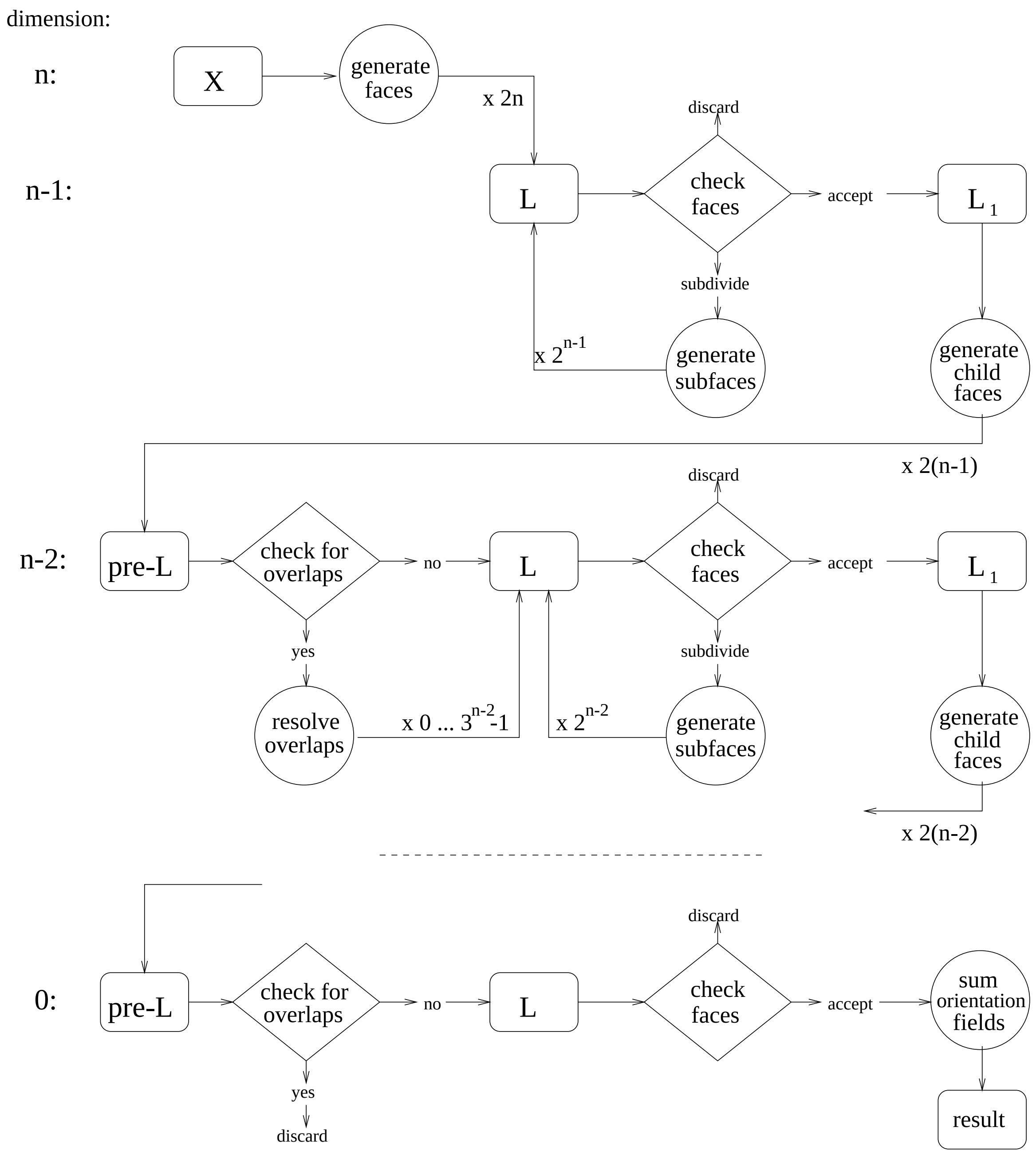


Figure 7.2: Schematic of the recursive topological degree algorithm.

tree. For a given number of initial faces, the average branching factor is indicative of the total number of nodes in the tree (and therefore of the total computational effort required in processing it). In particular, any tree of finite size (i.e. a subdivision process that terminates) must by necessity exhibit an average branching factor less than 1.

Let L_t be the set of terminal faces that are generated from L with an average branching factor b, where $b < 1$. By counting the numbers of nodes at each depth we have

$$\#L_t \; = \; \#L + \#Lb + \#Lb^2 + \ldots \; = \; \#L \sum_{i=0}^{\infty} b^i \; = \; \frac{\#L}{1-b}.$$

An example tree of sub-faces is given in Figure A.1. In the left-most tree, we start with a single face of dimension 2, thus the fixed branching factor is 0 for a terminal face and 4 for a face that is subdivided. The total number of faces (nodes) is 29 and the average branching factor is $\frac{28}{29}$.

- Overlap Ratio

  As indicated in Figure 7.2, child faces of one lower dimension are generated from L_1 and are placed into *pre-L*. The *overlap ratio* is defined as the proportion of these faces that overlap with another. (Note that the number of overlapping faces is exactly twice the number of occurrences of overlap, since all overlaps occur in pairs.)

- Overlap Badness

  In resolving an overlap between two m-dimensional faces, anywhere between 0 and $3^m - 1$ new faces may be generated to represent the disjoint union. The more new faces produced, the worse the outcome, in terms of computational effort. We may therefore quantify the *overlap badness* as the average number of new faces produced per overlap. If the product of this measure with the overlap ratio is above 2, the net effect of the overlap resolution is an increase in the size of L from *pre-L*.

Example 7.1. *Let $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$ where $\mathcal{F}(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$ be given by*

$$\begin{aligned} f_i(x_1, \ldots, x_n) &= x_i^2 - x_{i+1}, \;\; \text{for } i = 1, \ldots, n-1, \\ f_n(x_1, \ldots, x_n) &= x_n^2 - x_1. \end{aligned}$$

This is the extended Kearfott function [Kea79a]; known solutions are $(0, \ldots, 0)$ and $(1, \ldots, 1)$. We wish to compute $deg(\mathcal{F}, \mathbf{X}, 0)$, where

$$\mathbf{X} \; = \; [-10, 10]^n.$$

The recursive algorithm is run for the cases $n = 3$ and $n = 6$; see the diagnostic information in Table 7.1, where i is the iteration number and m is the dimension of faces in L. The algorithm successfully terminates, reporting in both cases that $deg(\mathcal{F}, \mathbf{X}, 0) = 0$ (note that the two solutions have opposite signs of their Jacobian determinants).

i	m	$\#L$	max. search depth	faces checked	avg. branch. factor	$\#L_1$	desig. ratio	overlaps	overlap ratio
				$\mathbb{R}^3$, $\mathbf{X} = [-10, 10]^3$, $deg(\mathcal{F}, \mathbf{X}, 0) = 0$					
1	2	6	0	6	0	1	16%	0	0%
2	1	4	0	4	0	2	50%	0	0%
				$\mathbb{R}^6$, $\mathbf{X} = [-10, 10]^6$, $deg(\mathcal{F}, \mathbf{X}, 0) = 0$					
1	5	12	0	12	0	1	8%	0	0%
2	4	10	0	10	0	2	20%	0	0%
3	3	16	0	16	0	4	25%	0	0%
4	2	24	0	24	0	8	33%	0	0%
5	1	32	0	32	0	16	50%	0	0%

Table 7.1: Diagnostic output for the degree calculations of Example 7.1.

No face subdivision at all is required for this example. This illustrates that a degree computation, even for a large box enclosing one or more solutions, can *potentially* be performed with very little effort.

Example 7.2. *Let $\mathcal{F} : \mathbb{R}^2 \to \mathbb{R}^2$ where $\mathcal{F}(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$ be given by*

$$\begin{aligned} f_1 &= x_2(x_2 + 1) - 4(x_1 + 1), \\ f_2 &= (x_2^3 - 10x_1 - \frac{13}{2})^2 - \frac{1}{4}. \end{aligned}$$

This system of equations represents the intersection of a parabola and a squared cubic. Solutions (specified to a precision of 3 decimal places) are $(-0.910, -1.281)$, $(-0.866, -1.386)$, $(-0.652, 0.781)$, $(-0.5, 1)$, $(2, 3)$ and $(1.804, 2.886)$. The recursive algorithm is run for four different boxes; see the diagnostic information in Table 7.2, with the same notation as before. In each case, the reported degree is equal to the sum of the signs of the Jacobian determinants of all solutions within the box.

Despite the low dimension of this example, a significant amount of face subdivision is required in all cases. These results hint that there is a (possibly weak) correlation between the interval widths of the box and the search depth.

Example 7.3. *Let $\mathcal{F} : \mathbb{R}^6 \to \mathbb{R}^6$ where $\mathcal{F}(x_1, \ldots, x_6) = (f_1(x_1, \ldots, x_6), \ldots, f_6(x_1, \ldots, x_6))$ be given by*

$$\begin{aligned} f_1 &= x_1^2 - x_2^2 + x_3^2 - x_4^2 + x_5^2 - x_6^2 - r^2, \\ f_2 &= 2x_1x_2 + 2x_3x_4 + 2x_5x_6, \\ f_3 &= (x_1 - 1)^2 - x_2^2 + (x_3 - 1)^2 - x_4^2 + (x_5 - 1)^2 - x_6^2 - r^2, \\ f_4 &= 2(x_1 - 1)x_2 + 2(x_3 - 1)x_4 + 2(x_5 - 1)x_6, \\ f_5 &= x_5 - x_3, \\ f_6 &= x_6 - x_4. \end{aligned}$$

i	m	$\#L$	max. search depth	faces checked	avg. branch. factor	$\#L_1$	desig. ratio	overlaps	overlap ratio
				$\mathbf{X} = [-5, 5]^2$, $deg(\mathcal{F}, \mathbf{X}, 0) = 0$					
1	1	4	4	14	0.714	1	7%	0	0%
				$\mathbf{X} = ([-1, 0], [0, 2])$, $deg(\mathcal{F}, \mathbf{X}, 0) = 0$					
1	1	4	3	14	0.714	1	7%	0	0%
				$\mathbf{X} = ([-0.6, -0.4], [0.9, 1.1])$, $deg(\mathcal{F}, \mathbf{X}, 0) = -1$					
1	1	4	2	12	0.667	3	25%	2	66%
				$\mathbf{X} = ([-0.6, 5], [0.9, 5])$, $deg(\mathcal{F}, \mathbf{X}, 0) = -1$					
1	1	4	7	22	0.818	1	4%	0	0%

Table 7.2: Diagnostic output for the degree calculations of Example 7.2.

The solutions to this system represent the extremal points of the intersection of two spheres, where the problem dimension has artificially been increased from 3 to 6. The two spheres are of radius r*, with centres* $(0, 0, 0)$ *and* $(1, 1, 1)$*.*
If $r > \frac{\sqrt{3}}{2}$*, a single pair of real solutions (with* $x_2 = x_4 = x_6 = 0$*) is produced.*
If $r < \frac{\sqrt{3}}{2}$*, complex solutions are generated.*
The recursive algorithm is run for different values of r *and* $\mathbf{X}$*; see the diagnostic information in Table 7.3. As before, the reported degree is equal to the sum of the signs of the Jacobian determinants of all solutions within the box.*

i	m	$\#L$	max. search depth	faces checked	avg. branch. factor	$\#L_1$	desig. ratio	overlaps	overlap ratio
				$r = 1$, $\mathbf{X} = [-1, 1]^6$, $deg(\mathcal{F}, \mathbf{X}, 0) = 2$					
1	5	12	4	10828	0.999	56	0%	132	47%
2	4	296	0	296	0	28	9%	52	46%
3	3	120	0	120	0	16	13%	24	50%
4	2	48	0	48	0	8	16%	8	50%
5	1	16	0	16	0	4	25%	2	50%
				$r = 2$, $\mathbf{X} = [1, 2]^6$, $deg(\mathcal{F}, \mathbf{X}, 0) = 0$					
1	5	12	0	12	0	0	0%		
				$r = 2$, $\mathbf{X} = ([-8, 0], [-2, 2], [-2, 2], [-2, 2], [-2, 2], [-2, 2])$, $deg(\mathcal{F}, \mathbf{X}, 0) = 1$					
1	5	12	4	13612	0.999	52	0%	98	37%
2	4	324	1	708	0.542	26	3%	53	50%
3	3	172	1	236	0.271	22	9%	35	53%
4	2	62	0	62	0	4	6%	4	50%
5	1	8	0	8	0	2	25%	1	50%

Table 7.3: Diagnostic output for the degree calculations of Example 7.3.

Here we can observe that there is a vast difference in the number of faces that need to be processed for boxes that contain solutions versus one that does not. This seems to be often (but is not always) the case. Where a significant amount of subdivision is required for a face of high dimension, many sub-faces are produced.

Example 7.4. *Let $\mathcal{F} : \mathbb{R}^5 \to \mathbb{R}^5$ where $\mathcal{F}(x_1, \ldots, x_5) = (f_1(x_1, \ldots, x_5), \ldots, f_5(x_1, \ldots, x_5))$ be given by*

$$\begin{aligned}
f_1(x_1, \ldots, x_5) &= (x_1 + x_5)^2 - \sin(x_3), \\
f_2(x_1, \ldots, x_5) &= x_1 x_2 + x_3 - x_4^2 - 2x_5, \\
f_3(x_1, \ldots, x_5) &= x_3 x_4 - x_1, \\
f_4(x_1, \ldots, x_5) &= x_2 + x_3 + e^{x_4} - 2, \\
f_5(x_1, \ldots, x_5) &= x_5.
\end{aligned}$$

We wish to compute $deg(\mathcal{F}, \mathbf{X}, 0)$, where

$$\mathbf{X} = [-10, 10] \times [-10, 10] \times [-5, 5] \times [-5, 5] \times [-5, 5].$$

The recursive algorithm is run; see the diagnostic information in Table 7.4. As before, the algorithm successfully terminates, reporting that $deg(\mathcal{F}, \mathbf{X}, 0) = 1$ (thus there is at least one solution to $\mathcal{F} = 0$ within $\mathbf{X}$).

i	m	$\#L$	max. search depth	faces checked	avg. branch. factor	$\#L_1$	desig. ratio	overlaps	overlap ratio
1	4	10	8	1690	0.994	24	1%	27	28%
2	3	251	4	563	0.554	21	3%	32	50%
3	2	103	1	115	0.104	6	5%	3	25%
4	1	18	0	18	0	6	33%	3	50%

Table 7.4: Diagnostic output for the degree calculation of Example 7.4.

Here, some of the component functions are non-polynomial. Note that, as with most of the preceding examples, each variable appears at most once in each component function; this example therefore illustrates the performance without any effect of the dependency problem (cf. Subsection 2.1.4) in interval arithmetic.

The preceding examples are typical of non-trivial degree computations over (relatively) large boxes. The following observations become readily apparent:

- The amount of computational effort is generally unpredictable. Where it is considerable, there is often a very similar example (e.g. the same function but with a different box) which would appear to have the same degree of complexity, but for which the algorithm requires little or no subdivision, or where almost all faces are discarded. This may be to a certain extent dependent on hidden factors such as the choice of ray direction as well as the function behaviour within the box. Nevertheless, certain categories of simple functions such as the identity function or the extended Kearfott function (Example 7.1) generally require no subdivision.

- The work–dimension distribution seems to be heavily weighted towards the faces of highest dimension, i.e. the majority of the work is performed in the first iteration. For large boxes, this may simply be due to the fact that the face selection criteria (i.e. select or discard) are more likely to be satisfied for small faces. The subdivision is mostly performed in the first iteration; thereafter the faces are already mostly small.

A more rigorous analysis of the algorithm with more examples is presented in Section 7.4.

7.3 Abstract Analysis (Face Subdivision)

In this section, we undertake an analysis of the face subdivision process in order to determine the likely average branching factor. The branching factor appears to be crucial in determining the overall computational effort — where it is low, only few faces are processed throughout.

We aim to determine the branching factor in the case of systems of linear equations, and then extrapolate to the general case.

7.3.1 Best and Worse Case Analysis

The best case of the face subdivision process occurs when no subdivision at all is required. Processing of such a face is trivial and the branching factor is 0. This case occurs with non-zero probability (an example is the identity function over the unit box in $\mathbb{R}^n$).

The subdivision process is guaranteed to terminate, provided that there are no solutions to the system of equations on the face (to within machine precision) and that the interval arithmetic used has the property that $w(\mathbf{f}(\mathbf{x})) \to 0$ as $w(\mathbf{x}) \to 0$, where $\mathbf{f}$ is an interval function and w denotes the width operator. The natural interval extension (see Definition 2.13) is inclusion isotone and satisfies this requirement. However the amount of subdivision required may be arbitrarily large; it is unfortunately not possible in general to compute an upper bound on the amount of computational work that may be required for any given problem. Asymptotically speaking, the worst case is a failure to terminate, which can only arise through limitations in interval arithmetic or machine precision which we assume, for reasonable categories of problems, to occur with negligible or very low probability.

Since the best and worse cases are both extreme, the only meaningful alternative is to perform an average case analysis. We firstly exclude all the (best) cases with branching factor 0, therefore this is more accurately described as an average case analysis of all non-trivial instances of face processing.

7.3.2 Basic Operations

We assume the basic operation in this complexity analysis to be an instance of a face check. For an m-dimensional face, this requires (up to) $m+1$ interval function evaluations. The cost of this check is not static, but does vary according to lexical function complexity (i.e. the number of basic operations in a function) and problem dimension in a predictable way. It is also dependent on the efficiency of the interval arithmetic.

7.3.3 Linear Systems in $\mathbb{R}^2$

This case is straightforward. A face s of a 2-dimensional box $\mathbf{X}$ is a 1-dimensional line segment. The zero sets of the two linear functions f_1 and f_2 constitute straight lines which must (since we assume non-triviality) each intersect s. This is the only requirement of the functions, so the two intersection points can be considered to be designated at random. The face s is subdivided into sub-faces s_1 and s_2 by bisecting the line segment. Each sub-face may contain one of the intersection points, or one may contain both; it is clear that each case arises with probability $\frac{1}{2}$. In the former case both sub-faces are terminal; in the latter case the sub-face containing both points is non-terminal. Therefore for non-trivial faces with random linear functions, the average branching factor is $0 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = \frac{1}{2}$.

7.3.4 Linear Systems in $\mathbb{R}^3$

Here we consider a face of a 3-dimensional box $\mathbf{X}$, and component linear functions f_1, f_2, and f_3. Let us assume that the face is non-trivial, i.e. at least some subdivision is required. The interval values for f_1, f_2, and f_3 over the (2-dimensional) face then must all contain zero.

In abstract terms we can envisage this as the solution sets of each $f_i(x) = 0$ (straight lines) intersecting the face (a rectangle), illustrated in Figure 7.3.

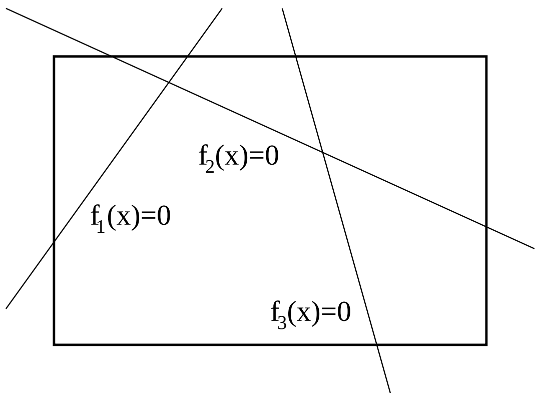


Figure 7.3: Zero sets of f_1, f_2, and f_3 intersecting a rectangular face.

To determine the average branching factor of the search tree here, we need to consider how many of the sub-faces (the four quadrants of the rectangle) are designated after a subdivision. The problem may thus be formulated abstractly as follows:

- Given a rectangle intersected by three 'random' lines, how many of its four quadrants are likely to be intersected by all three lines?

We first need to define exactly what is meant by a 'random' line. The only property which is assumed is that each line intersects the rectangle, so from the set of all possible random lines, we restrict selection to those for which this holds.

There are different conceivable ways of picking random lines, which correspond to different choices of random parameters in the corresponding linear equations. A line in a two-dimensional plane can be defined either by two distinct points (through which the line passes), or as a point and a gradient (alternately specified as an angle of incidence with an axis of reference). It is clear that a random choice of angle, rather than gradient, is sensible, since we wish to select random parameters within finite bounds. Random points could be selected either upon the boundary of, or within the rectangle, or upon a segment of the x- or y-axis.

The set of all intersecting lines has infinitely many distinct members, so choosing a member at random is a non-trivial issue. Deciding upon this issue actually transpires to be a rather subtle means of parameter selection corresponding to slightly different definitions of randomness. For 'true' randomness, we wish to afford no bias to any types of line. This means that any point within the rectangle should be just as likely to be intersected by a line as any other point. Therefore the following approach is devised:

Line Family Model

Given a rectangular face, for a given intersecting line, let θ be the angle of incidence with the horizontal axis. It is assumed that all angles $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ occur with equal probability in the set of all intersecting lines. (Note that the direction of the line is irrelevant.) Fixing θ, we may consider the set of intersecting lines, all with gradient $\arctan(\theta)$, each member of which may intersect 1, 2, or 3 (but not 0) of the quadrants. Such a set is depicted in Figure 7.4.

Conditional upon θ, the general strategy now is to determine the probability of such a line intersecting a single quadrant, $P(1 \,|\, \theta)$, the probability of intersecting exactly two quadrants, $P(2 \,|\, \theta)$, and of intersecting three quadrants, $P(3 \,|\, \theta)$. It should be clear that it is not possible for any one line to wholly intersect all four quadrants, or that lines coincident with either of the bisectors for the rectangle occur with probability zero. Each line in the family can be characterised by a unique point of intersection with any chosen non–parallel line, such as an axis. The conditional probabilities can be determined by integration over all possible intercept values.

The overall probability of intersection with exactly n quadrants (where n must be 1, 2, or 3) is given by

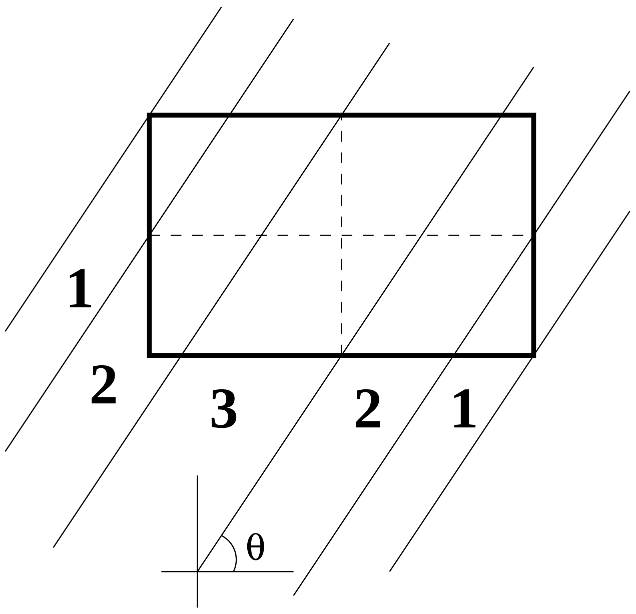


Figure 7.4: Set of line families with angle θ intersecting a rectangle.

$$P(n) \;=\; \frac{1}{\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} P(n\,|\,\theta)\,d\theta.$$

Computation of $P(1)$

We enumerate the (horizontal) length of the rectangle as a, and the (vertical) height as b, and split into two separate cases:

- **Case 1:** $\tan(\theta) < \frac{b}{a}$

  This case is depicted in Figure 7.5. By measuring the range of intercept l of all lines, and the range of intercept d of lines intersecting just one quadrant (the upper left or bottom right) we have

$$P_1(1\,|\,\theta) \;=\; \frac{2d}{l} \;=\; \frac{a\tan(\theta)}{b + a\tan(\theta)}.$$

- **Case 2:** $\tan(\theta) > \frac{b}{a}$**,** $\tan(\phi) < \frac{a}{b}$

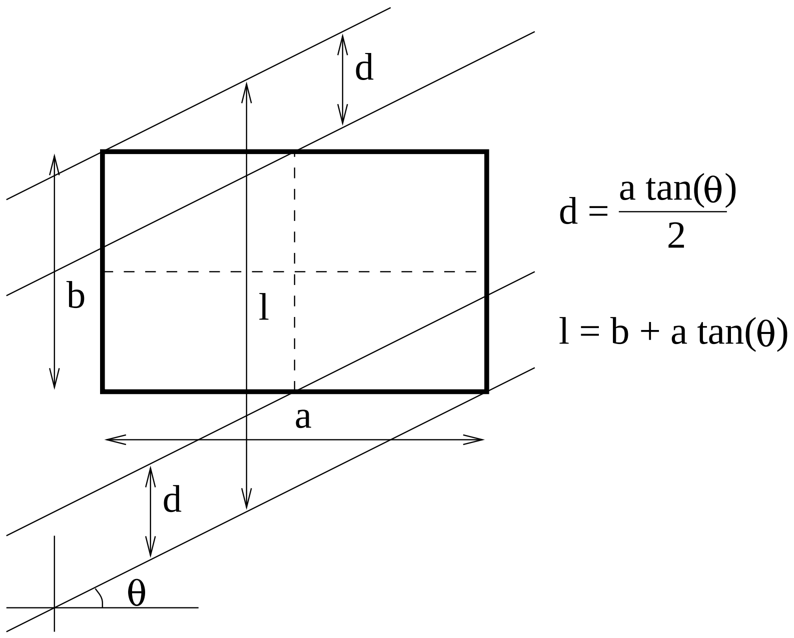


Figure 7.5: Lines intersecting one quadrant of the rectangle (Case 1).

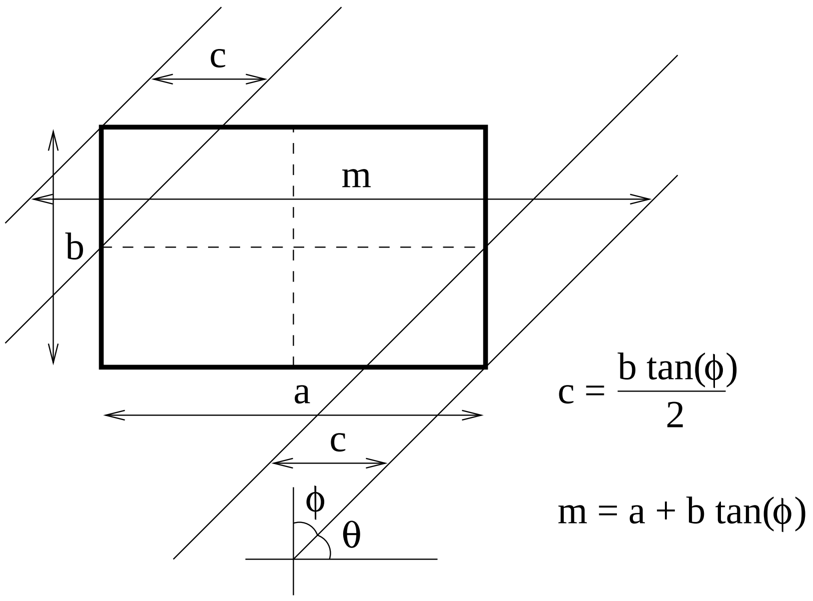


Figure 7.6: Lines intersecting one quadrant of the rectangle (Case 2).

This case is depicted in Figure 7.6. Setting $\phi = \frac{\pi}{2} - \theta$ and measuring the range m of intercept of all lines, and the range of intercept c of lines intersecting just one quadrant we have

$$P_2(1 \mid \phi) \;=\; \frac{2c}{m} \;=\; \frac{b\tan(\phi)}{a + b\tan(\phi)}.$$

Combining the two cases and integrating over all possible θ values (and considering negative gradient lines by symmetry) we have

$$\begin{aligned}
P(1) &= \left(\int_0^{\arctan\frac{b}{a}} \frac{a\tan\theta}{b + a\tan\theta}\,d\theta + \int_0^{\arctan\frac{a}{b}} \frac{b\tan\phi}{a + b\tan\phi}\,d\phi \right) \frac{2}{\pi} \\
&= \frac{2}{\pi} (\left[\frac{a}{2(a^2+b^2)} \big(b\log(\tan^2\theta + 1) - 2b\log(a\tan\theta + b) + 2a\theta \big) \right]_0^{\arctan\frac{b}{a}} \\
&\quad + \left[\frac{b}{2(a^2+b^2)} \big(a\log(\tan^2\phi + 1) - 2a\log(b\tan\phi + a) + 2b\phi \big) \right]_0^{\arctan\frac{a}{b}}) \\
&= \frac{1}{\pi(a^2+b^2)} \left(a \left(b\log\left(\frac{b^2}{a^2} + 1\right) - 2b\log(2b) + 2a\arctan\frac{b}{a} + 2b\log b \right) \right. \\
&\quad \left. + b \left(a\log\left(\frac{a^2}{b^2} + 1\right) - 2a\log(2a) + 2b\arctan\frac{a}{b} + 2a\log a \right) \right) \\
&= \frac{1}{\pi(a^2+b^2)} \left(2ab\log\left(\frac{a^2+b^2}{ab}\right) - 4ab\log 2 + 2a^2\arctan\frac{b}{a} + 2b^2\arctan\frac{a}{b} \right) \\
&= \frac{2ab}{\pi(a^2+b^2)} \left(\log\left(\frac{a^2+b^2}{ab}\right) - \log 4 + \frac{a}{b}\arctan\frac{b}{a} + \frac{b}{a}\arctan\frac{a}{b} \right).
\end{aligned}$$

Setting the ratio of the side lengths, $r = \frac{a}{b}$, we have

$$P(1) \;=\; \frac{2}{\pi}\left(\frac{r}{r^2+1}\right)\left(\log\left(r + \frac{1}{r}\right) - \log 4 + r\arctan\frac{1}{r} + \frac{1}{r}\arctan r\right). \tag{7.2}$$

In the case of a square face (i.e. $r = 1$), $P(1) = \frac{1}{2} - \frac{\log 2}{\pi} \approx 0.27936$.

Computation of $P(1 \vee 2)$

Labelling the rectangle as before, we consider the same two cases:

- **Case 1:** $\tan(\theta) < \frac{b}{a}$

  This case is depicted in Figure 7.7. By measuring the range of intercept l of all lines, and the range of intercept d of lines intersecting either one or two quadrants (here, a horizontally adjacent pair) we have

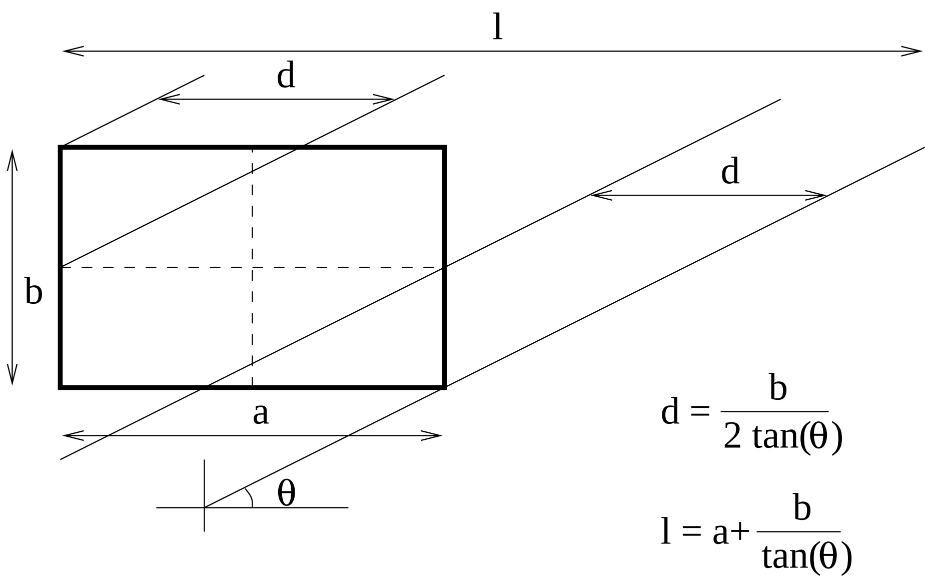

l
d
d
b
a
θ
$d = \frac{b}{2\tan(\theta)}$
$l = a + \frac{b}{\tan(\theta)}$

Figure 7.7: Lines intersecting one or two quadrants of the rectangle (Case 1).

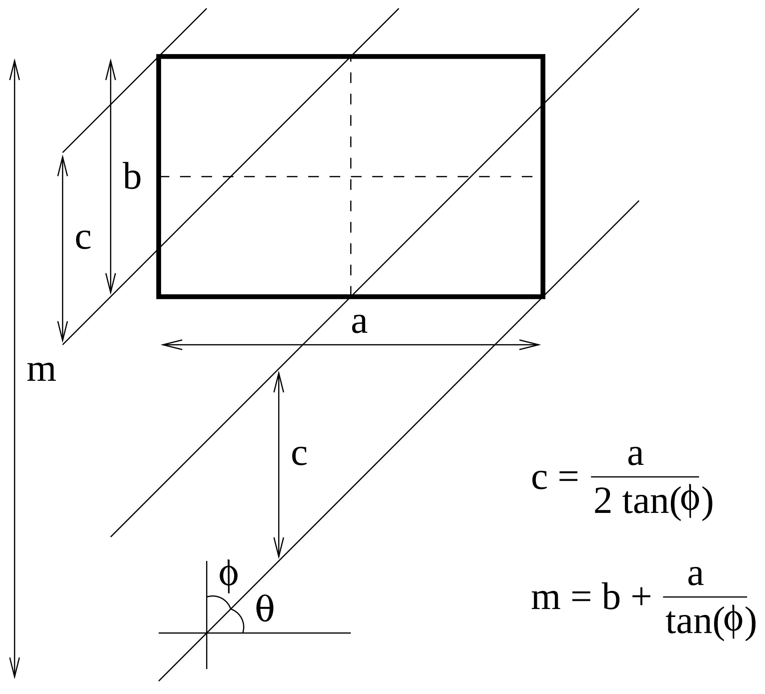


Figure 7.8: Lines intersecting one or two quadrants of the rectangle (Case 2).

$$P_1((1 \vee 2) \mid \theta) = \frac{2d}{l} = \frac{b}{b + a\tan(\theta)}.$$

- **Case 2:** $\tan(\theta) > \frac{b}{a}$, $\tan(\phi) < \frac{a}{b}$

  This case is depicted in Figure 7.8. Setting $\phi = \frac{\pi}{2} - \theta$, as before, and measuring the range of intercept m of all lines, and the range of intercept c of lines intersecting either one or two quadrants (here, a vertically adjacent pair) we have

$$P_2((1 \vee 2) \mid \phi) = \frac{2c}{m} = \frac{a}{a + b\tan(\phi)}.$$

Combining the two cases and integrating over all possible θ values (and considering negative gradient lines by symmetry) we have

$$\begin{aligned}
P(1 \vee 2) &= \left(\int_0^{\arctan \frac{b}{a}} \frac{b}{b + a\tan\theta}\, d\theta + \int_0^{\arctan \frac{a}{b}} \frac{a}{a + b\tan\phi}\, d\phi \right) \frac{2}{\pi} \\
&= \frac{2}{\pi} \Big(\left[\frac{b}{2(a^2+b^2)} \left(-a\log(\tan^2\theta + 1) + 2a\log(a\tan\theta + b) + 2b\theta \right) \right]_0^{\arctan \frac{b}{a}} \\
&\quad + \left[\frac{a}{2(a^2+b^2)} \left(-b\log(\tan^2\phi + 1) + 2b\log(b\tan\phi + a) + 2a\phi \right) \right]_0^{\arctan \frac{a}{b}} \Big) \\
&= \frac{1}{\pi(a^2+b^2)} \left(b \left(-a\log\left(\frac{b^2}{a^2} + 1\right) + 2a\log(2b) + 2b\arctan\frac{b}{a} - 2a\log b \right) \right. \\
&\quad \left. + a \left(-b\log\left(\frac{a^2}{b^2} + 1\right) + 2b\log(2a) + 2a\arctan\frac{a}{b} - 2b\log a \right) \right) \\
&= \frac{1}{\pi(a^2+b^2)} \left(-2ab\log\left(\frac{a^2+b^2}{ab}\right) + 4ab\log 2 + 2b^2\arctan\frac{b}{a} + 2a^2\arctan\frac{a}{b} \right) \\
&= \frac{2ab}{\pi(a^2+b^2)} \left(-\log\left(\frac{a^2+b^2}{ab}\right) + \log 4 + \frac{b}{a}\arctan\frac{b}{a} + \frac{a}{b}\arctan\frac{a}{b} \right).
\end{aligned}$$

With the ratio of the side lengths, $r = \frac{a}{b}$, as before, we have

$$P(1 \vee 2) = \frac{2}{\pi}\left(\frac{r}{r^2+1}\right)\left(-\log\left(r + \frac{1}{r}\right) + \log 4 + \frac{1}{r}\arctan\frac{1}{r} + r\arctan r \right). \tag{7.3}$$

In the case of a square face (i.e. $r = 1$), $P(1 \vee 2) = \frac{1}{2} + \frac{\log 2}{\pi} \approx 0.72064$.

Derivation of Average Branching Factor

Now $P(3) = 1 - P(1 \vee 2)$ and we can observe from (7.2) and (7.3) that $P(1) + P(1 \vee 2) = 1$. Therefore there are only two distinct probabilities that we need to be concerned with, which, for brevity in the material which follows, we label as p and q:

$$\begin{aligned} p &:= P(1) = P(3), \\ q &:= 1 - 2p = P(2). \end{aligned}$$

Let us now consider three random, independent lines (linear functions) crossing the rectangle (two-dimensional face). Each line may intersect 1, 2, or 3 quadrants, with probabilities p, q, and p, respectively. We are interested to know if any of the four quadrants are intersected by all three lines (a sub-face which requires further subdivision) and if so, how many. There may be 0, 1, 2, or 3 of the quadrants (sub-faces) so designated.

There are a number of different cases to consider, depending upon the pattern of intersection of the three lines, listed in Table 7.5. Here, a 'configuration' of (e.g.) 1,1,2 means that two of the lines intersect one quadrant and that one of the lines intersects two quadrants.

Configuration of lines	Probability of configuration	Conditional probability that the number of quadrants containing all 3 lines is:			
		0	**1**	**2**	**3**
1,1,1	p^3	$\frac{15}{16}$	$\frac{1}{16}$	0	0
1,1,2	$3p^2q$	$\frac{7}{8}$	$\frac{1}{8}$	0	0
1,1,3	$3p^3$	$\frac{13}{16}$	$\frac{3}{16}$	0	0
1,2,2	$3pq^2$	$\frac{3}{4}$	$\frac{1}{4}$	0	0
1,2,3	$6p^2q$	$\frac{5}{8}$	$\frac{3}{8}$	0	0
1,3,3	$3p^3$	$\frac{7}{16}$	$\frac{9}{16}$	0	0
2,2,2	q^3	$\frac{3}{4} - \frac{3}{4}\alpha(1-\alpha)$	$\frac{3}{2}\alpha(1-\alpha)$	$\frac{1}{4} - \frac{3}{4}\alpha(1-\alpha)$	0
2,2,3	$3pq^2$	$\frac{1}{2} - \frac{1}{2}\alpha(1-\alpha)$	$\frac{1}{4} + \alpha(1-\alpha)$	$\frac{1}{4} - \frac{1}{2}\alpha(1-\alpha)$	0
2,3,3	$3p^2q$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{1}{4}$	0
3,3,3	p^3	0	$\frac{3}{8}$	$\frac{9}{16}$	$\frac{1}{16}$

Table 7.5: Table of conditional probabilities arising in the calculation of the average branching factor.

The entries for the configurations 2,2,2 and 2,2,3 require a little explanation. Where a line intersects two quadrants, α is defined as the probability that these two quadrants will be vertically adjacent, rather than horizontally adjacent. (Of course, lines intersecting two diametrically opposite quadrants occur with probability zero.) We will see that we do not require an explicit formula for α, but note that it is dependent on the ratio of side lengths r.

By taking the conditional probabilities in the third column, and multiplying by the configuration probabilities and taking the sum we can compute the overall probability that no quadrants are designated (i.e. no quadrant contains all three lines). We can proceed likewise for the overall probabilities that one or more quadrants are designated. Let N be the number of designated quadrants.

$$\begin{aligned}
P(N=0) &= \frac{1}{16}(15p^3+42p^2q+39p^3+36pq^2+60p^2q+21p^3+ \\
&\quad 12q^3-12\alpha(1-\alpha)q^3+24pq^2-24\alpha(1-\alpha)pq^2+6p^2q) \\
&= \frac{1}{16}\left(75p^3+108p^2q+(60-24\alpha(1-\alpha))pq^2+(12-12\alpha(1-\alpha))q^3\right), \\
P(N=1) &= \frac{1}{16}(p^3+6p^2q+9p^3+12pq^2+36p^2q+27p^3+ \\
&\quad 24\alpha(1-\alpha)q^3+12pq^2+48\alpha(1-\alpha)pq^2+30p^2q+6p^3) \\
&= \frac{1}{16}\left(43p^3+72p^2q+(24+48\alpha(1-\alpha))pq^2+24\alpha(1-\alpha)q^3\right), \\
P(N=2) &= \frac{1}{16}\left(4q^3-12\alpha(1-\alpha)q^3+12pq^2-24\alpha(1-\alpha)pq^2+12p^2q+9p^3\right) \\
&= \frac{1}{16}\left(9p^3+12p^2q+(12-24\alpha(1-\alpha))pq^2+(4-12\alpha(1-\alpha))q^3\right), \\
P(N=3) &= \frac{1}{16}\left(p^3\right).
\end{aligned}$$

We can now compute E, the expected number of quadrants designated:

$$\begin{aligned}
E &= \sum_{i=1}^{3} iP(N=i) \\
&= \frac{1}{16}\left(64p^3+96p^2q+48pq^2+8q^3\right) \\
&= 4p^3+6p^2q+3pq^2+\frac{1}{2}q^3 \\
&= \left(p+\frac{1}{2}q\right)\left(4p^2+4pq+q^2\right)
\end{aligned}$$

$$\begin{aligned} &= \left(p + \frac{1}{2}q\right)(2p+q)^2 \\ &= \frac{1}{2}(2p+q)^3 \\ &= \frac{1}{2}. \end{aligned}$$

This satisfying result proves that, for this particular problem type at least, the average–case complexity (i.e. average branching factor) of the face subdivision process is independent of the ratio of side lengths of the face. Also, we can expect the subdivision process for a non-trivial face to be characterised by a branching factor of $\frac{1}{2}$ on average.

That the expression for E resolves to such a simple fraction also hints at the possibility that an easier method for its calculation might exist.

We briefly outline two possible alternatives to the line family model:

Area Integration Model

This is a model in which all points within the rectangle are assumed to be equally likely to be hit by any intersecting line. For any such point, we consider a family of lines passing through it, where the angle of incidence varies. One may then integrate over the x- and y-directions in turn.

Boundary Integration Model

Here we consider that all points on the boundary of the rectangle are equally likely to be hit by any intersecting line. One can integrate over the range of possible boundary intercepts.

The three models proposed here correspond to differing definitions of 'random lines intersecting a rectangle'. Actual results for random examples will depend on the particular choice of random parameter(s).

7.3.5 Linear Systems in $\mathbb{R}^n$

Conjecture 7.1. *Suppose we are given an ($n-1$-dimensional) face s of an n-dimensional box $\mathbf{X}$ and arbitrary component linear functions $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, n$, where each f_i attains zero somewhere within s. Partition s into 2^{n-1} sub-faces by bisecting each component interval of s. Then the expected number of sub-faces intersected by all of the zero sets of f_i, $i = 1, \ldots, n$, (i.e. the average branching factor in the corresponding part of the degree computation for s) is $\frac{1}{2}$.*

This conjecture remains open for general n, but we have proven the cases $n = 2$ and $n = 3$, assuming that the arbitrary functions are distributed randomly according to the line family model. A further discussion on true randomness is beyond the scope of this work.

7.3.6 Nonlinear Systems in $\mathbb{R}^n$

Conjecture 7.2. *Suppose we are given an ($n-1$-dimensional) face s of an n-dimensional box* $\mathbf{X}$ *and arbitrary component functions $f_i : \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, n$, where each f_i is continuously differentiable and attains zero somewhere within s. Partition s into 2^{n-1} sub-faces by bisecting each component interval of s. Then the expected number of sub-faces intersected by all of the zero sets of f_i, $i = 1, \ldots, n$, (i.e. the average branching factor in the corresponding part of the degree computation for s) is greater than $\frac{1}{2}$, but tends towards $\frac{1}{2}$ as the component interval widths tend towards zero.*

Intuitively, it seems more likely that an arbitrary curve intersecting a face will enter more sub-faces than an arbitrary straight line, on average. For continuously differentiable functions, as the size of the face becomes smaller, so the complexity analysis should become increasingly similar to the linear case.

An average-case complexity analysis for arbitrary nonlinear functions is highly non-trivial. One possibility may be to model a nonlinear function (or its derivative, for a continuously differentiable function) as a random walk that intersects the face.

7.4 Data Analysis

To test in practice the characteristics of the face subdivision process in the recursive computation of topological degree, a large number of randomly-generated examples were run. A description of the software is given in Appendix A. The following categories of functions $\mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$ where $\mathcal{F}(x_1, \ldots, x_n) = (f_1(x_1, \ldots, x_n), \ldots, f_n(x_1, \ldots, x_n))$ were tested:

- **Linear functions in $\mathbb{R}^n$**

$$f_i(x_1, \ldots, x_n) \;=\; \sum_{j=0}^{n} b_{ij} x_j + c_i, \quad i = 1, \ldots, n. \tag{7.4}$$

- **Simple quadratics in $\mathbb{R}^n$**

$$f_i(x_1, \ldots, x_n) \;=\; \sum_{j=0, j\neq i}^{n} \left(a_{ij} x_j^2 + b_{ij} x_j \right) + x_i + c_i, \quad i = 1, \ldots, n. \tag{7.5}$$

- **Quadratics with cross terms in $\mathbb{R}^n$**

$$f_i(x_1, \ldots, x_n) \;=\; \sum_{j=0}^{n} \sum_{k=0}^{n} a_{ijk} x_j x_k + \sum_{j=0}^{n} b_{ij} x_j + c_i, \quad i = 1, \ldots, n. \tag{7.6}$$

In every case, each of the coefficients $a_{ij(k)}$, b_{ij}, and c_i were randomly assigned floating-point values in the range $[-1, 1]$. For each category, 1000 such instances were run, in each case to compute the degree at zero, $deg(\mathcal{F}, \mathbf{X}, 0)$, with $\mathbf{X} = [-1, 1]^n$. The maximum required search depth (over all faces) was recorded in each case.

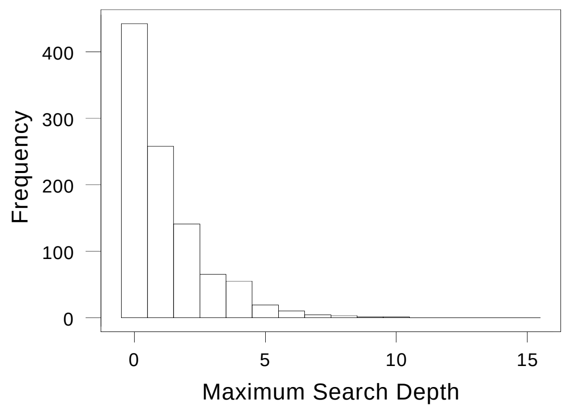
Linear Functions vs. Simple Quadratics

The results for problem types (7.4) and (7.5) with $n = 2$ are given in Figure 7.9.

Were each problem instance to consist of a single starting face, and assuming a fixed average branching factor, we might expect the maximum search depth to exhibit an exponential distribution. Since each problem instance consists of $2n$ starting faces, however, we can instead expect a negative binomial distribution. As the maximum search depth increases, this asymptotically approaches an exponential distribution, where each increment in the maximum search depth should correspond to a multiplication of the frequency by the average branching factor. Both distributions here appear consistent with this model with an average branching factor of about $\frac{1}{2}$.

There is a clear difference in the typical maximum search depth between the two problem types. This appears to be consistent with Conjecture 7.2, but we should be careful about drawing a firm conclusion — the difference may to a greater or lesser extent be attributable to the interval arithmetic dependency problem (cf. Subsection 2.1.4), which is slightly in effect for the simple quadratics (7.5) but not for the linear functions (7.4).

We should also note that the choice of random parameters does not necessarily conform to the line family model of Subsection 7.3.4, nor do we filter out trivial instances where the zero sets of one or more of the component functions do not intersect the box at all. Firstly, this has the consequence that the frequency of cases where no subdivision at all is required is not necessarily significant. Secondly, there may be a small sampling bias (e.g. towards cases where the zero sets of the component functions tend to occur near the centre of the box), which may affect the average branching factor (here, increasing it slightly from the conjectured value of $\frac{1}{2}$).



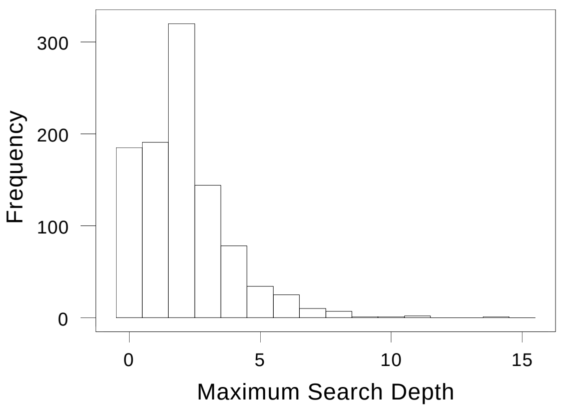


Figure 7.9: 1000 random degree computations: linear functions (left) and simple quadratics (right) in $\mathbb{R}^2$.

Simple Quadratics vs. Quadratics with Cross Terms

The results for problem types (7.5) and (7.6) with $n = 3$ are given in Figure 7.10.

The distributions of the maximum search depth are more clearly recognisable as negative binomial distributions, asymptotically approaching exponential decay corresponding to an average branching factor with a value close to $\frac{1}{2}$.

There is a significant difference in the mean maximum search depth, with the quadratics with cross terms (7.6) requiring more face subdivision (and thus the generation of more sub-faces and more computational effort overall). Again, this may be largely attributable to the dependency problem, which has a stronger effect for this category of problem, where each variable appears more often in the function expressions.

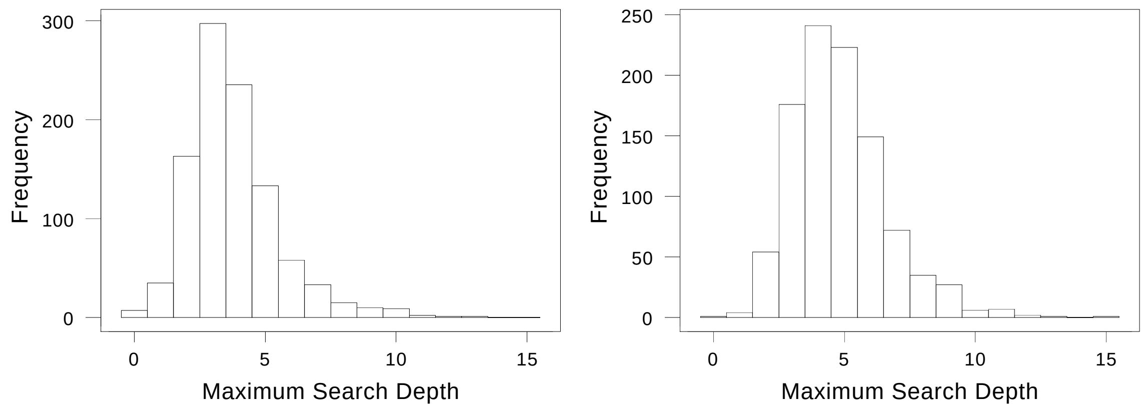


Figure 7.10: 1000 random degree computations: simple quadratics (left) and quadratics with cross terms (right) in $\mathbb{R}^3$.

Simple Quadratics by Dimension

The results for problem types (7.5) with $n = 2, 3, 4, 5$ are given in Figure 7.11.

We can observe a gradual increase of the mean maximum search depth (a rightward shift of the distribution) as n increases, which can be attributable to two factors: Firstly, the overestimation of interval widths due to the dependency problem should increase broadly linearly with n, since each of the n variables contribute additively. Secondly, we have a number of starting faces equal to $2n$. The negative binomial distribution exhibits just such a shift as the number of trials increases. Thus, there is likely still a difference when comparing single face instances of differing dimensions, but less marked.

It therefore seems reasonable to propose that the search depth does not *intrinisically* increase with n, but rather is more linked to the lexical complexity or behaviour of (the interval extensions of) the component functions. The asymptotic behaviour of the distributions remains the same and appears to be unrelated to n. This is strong evidence that the average branching factor is independent of the problem dimension.

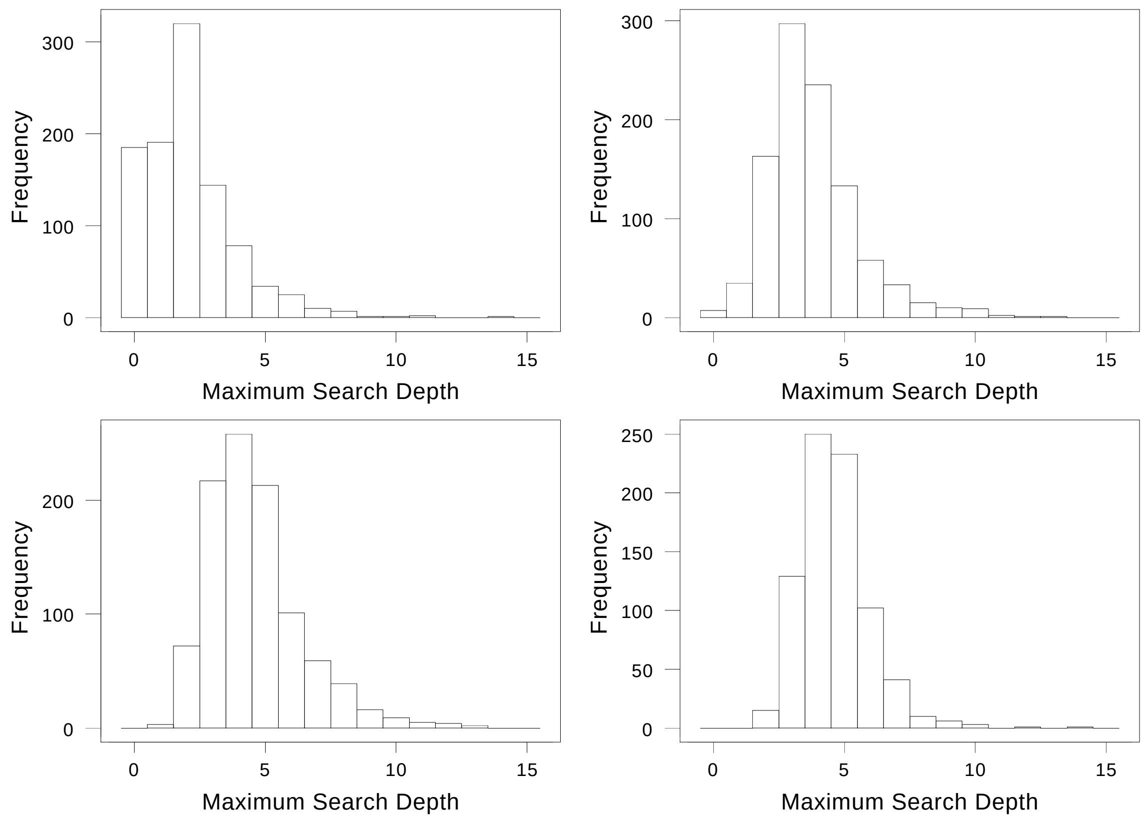


Figure 7.11: 1000 random degree computations: simple quadratics in $\mathbb{R}^2$ (top left), $\mathbb{R}^3$ (top right), $\mathbb{R}^4$ (bottom left), and $\mathbb{R}^5$ (bottom right).

7.5 Subdivision Strategy

The recursive degree-computation algorithm may essentially be regarded as a sequence of interval computations upon faces. Instances leading to minimal computational effort arise where no face subdivisions are incurred. Morever, the effort required for such cases is negligible in comparison to the average, non-trivial case. Considering an individual m-dimensional face (or sub-face), the upper bound on the number of interval function evaluations is m, i.e. there is a small fixed cost for each face. It is then clear that the overall computational effort for a degree calculation is determined to a large extent by the number of faces which are processed in total.

We have also seen that the processing of faces is in general characterised by a large number which are discarded or designated and a relatively small proportion of 'problem' faces. We are usually faced with a large number of sub-faces which descend from these one or few crucial faces. In other words, it is the process of face subdivision which is responsible for the majority of the computational effort. It is therefore reasonable to assume that the choice of face subdivision strategy is a crucial design choice in an attempt to reduce average computational effort.

In this section we present an abstract analysis, based on the assumption of precise and complete knowledge of the solution sets of the relevant equations — knowledge which we will not have in practice. While not therefore directly applicable, certain concepts are elaborated which may be useful for motivating and guiding the design process for new subdivision strategies.

7.5.1 Theoretical Optimal Strategy

Suppose we have a n-dimensional box $\mathbf{X}$ and a continuous function $\mathcal{F} : \mathbf{X} \to \mathbb{R}^n$ given by (7.1). Given an m-dimensional face which arises in a topological degree computation over $\mathbf{X}$, we are required to evaluate some or all of the functions $f_{n-m}, \dots, f_n$ over the face. A subdivision is required precisely when all of the resultant intervals contain zero; in this case the face is partitioned into sub-faces. A sub-face will itself require further subdivision if all of the interval evaluations over it contain zero. Ideally, therefore, we would partition the original face into the smallest number of sub-faces that do not require further subdivision, so that the total number of faces to be processed is minimised.

Abstracted Subdivision Problem

As before, we can abstract the notion of face subdivision away from the broader context of computing the topological degree. Here we will consider the first iteration of the algorithm, without loss of generality. Disregarding the assigned orientation, a face s can be considered to be just an $n-1$-dimensional box in $\mathbb{R}^n$. We have n continuous functions $f_1, \dots, f_n : s \to \mathbb{R}$, not identically zero, with no zero in s common to all functions. We wish to partition s into sub-boxes $s_1, \dots, s_k$ such that for each s_i, $i = 1, \dots, k$, there exists some $j_i \in \{1, \dots, n\}$ for which $0 \notin \mathbf{f}_{j_i}(s_i)$, where the latter denotes the interval extension used for f_{j_i}.

Definition 7.1 (Non-terminal)**.** *A face s is* non-terminal*, with respect to a set of (interval extensions for) functions $\mathbf{f}_1, \dots, \mathbf{f}_n$, if $\forall i = 1, \dots, n,\ 0 \in \mathbf{f}_i(s)$.*

The non-terminal faces are precisely those which require subdivision in the degree computation algorithm.

Definition 7.2 (Subdivision point)**.** *A* subdivision point *is a point chosen inside an $n-1$-dimensional face as the basis for a partitioning of that face into 2^{n-1} sub-faces, by dividing all component intervals about that point. More generally, we may choose not to divide all component intervals of a face s; in this case we just say that it is partitioned into sub-faces $s_1, \dots, s_k$. The term* subdivision *will either refer to the partition itself (as a set of sub-faces) or to the $n-2$-dimensional box which forms the common boundary of any two adjacent sub-faces. The context in which this is used should make the meaning unambiguous.*

Optimal Face Partitioning

Definition 7.3 (Optimal partitioning)**.** *A partitioning of s into terminal sub-faces $s_1, \ldots, s_k$ is* optimal*, with respect to a set of (interval extensions for) functions $\mathbf{f}_1, \ldots, \mathbf{f}_n$, if, for any other partitioning into terminal sub-faces $s'_1, \ldots s'_l$, $k \leq l$, i.e. k is minimal.*

Let us consider the solution set for each of $f_1(x) = 0, \ldots, f_n(x) = 0$ over s. Under the starting assumptions, each solution set must consist of one or more closed subsets of s of dimension $n-2$ or less: curves in $\mathbb{R}^2$, surfaces in $\mathbb{R}^3$, hypersurfaces in $\mathbb{R}^{4+}$. Under the terminology of cylindrical decomposition (cf. Subsection 5.3.4), sign conditions on all the f_i will, collectively, form a stratification of s into manifolds.

Typical Case

Let us assume that the solution set of $f_i(x) = 0$ for at least one $i \in \{1, \ldots, n\}$ does not project fully onto all edges (child faces) of the face, viz. the box enclosure for the intersection of the solution set of $f_i(x) = 0$ with the face s is smaller than s itself.

The following abstract mechanism for determining the minimum number of subdivisions is then proposed: We can arbitrarily select an $n-2$-dimensional cross-section of the face which does not intersect at least one of these solution sets, by restricting one of the interval fields to a point. This subset of the face can then be expanded (back into an $n-1$-dimensional box) by gradually extending the width of the interval field that was restricted, only until all of the solution sets intersect — this now constitutes a non-terminal subface. We are now guaranteed that at least one subdivision must occur within this designated region. Since the last of the solution sets (continuous curves) to be included only intersects minutely, we can be sure that only one subdivision within this region will suffice.

The remainder of the face can be partitioned by repetition of this method, assuming we always remain in the typical case. We then have a partition into $r_1 + r_0$ (box) regions, of which r_1 should contain exactly one (distinct) subdivision, and r_0 of which (possibly $r_0 = 0$) do not require a subdivision. Therefore the minimum possible number of subdivisions is precisely r_1, which results in an optimum partitioning into $r_1 + 1$ sub-faces. With such a partitioning, the maximum number of interval function evaluations over this face is thus $n(r_1 + 1)$.

Example 7.5. *Consider a 2-dimensional face s from a box in $\mathbb{R}^3$, where the solution sets of $f_1(x) = 0$, $f_2(x) = 0$, $f_3(x) = 0$ are as depicted in Figure 7.12.*

- *Starting from the left-hand edge of the face, we must have a sub-face which intersects the $f_2(x) = 0$ curve. Gradually extending this rightward, we can incorporate part of $f_3(x) = 0$ and finally we can just include the smallest part of $f_1 = 0$. This is the first region, R_1, which must contain a subdivision.*

- *Continuing on in the same fashion, we can designate a region R_2 which intersects $f_1(x) = 0$, $f_2(x) = 0$, and the smallest part of $f_3(x) = 0$.*

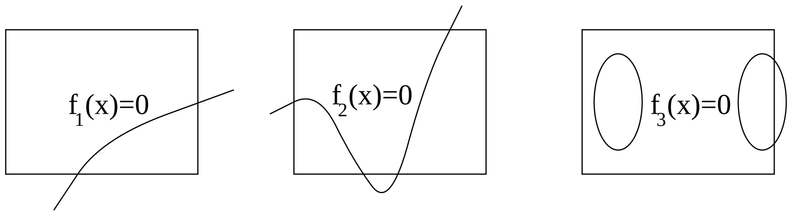


Figure 7.12: Solution sets of $f_i(x) = 0$, $i = 1, 2, 3$, over a face s in Example 7.5.

- *The left-hand edge of the remainder of the face ($s \backslash (R_1 \cup R_2)$) is intersected by all three solution sets, so we can instead expand a new region from the top. This can grow down until it finally includes $f_1(x) = 0$ in its corner. This region is designated R_3.*
- *The remaining region, R_4, is not intersected by $f_2(x) = 0$, so it need not contain a subdivision.*

We thus have a partition into four regions; $R_1, \ldots, R_4$, three of which (R_1, R_2, and R_3) need a subdivision, as depicted in Figure 7.13. An optimal subdivision using precisely three subdivisions, i.e. into four subfaces ($s_1, \ldots, s_4$), is therefore possible, as illustrated in Figure 7.14.

The reader's attention is drawn to the difference between the partition of the face into the regions R_i, each of which must contain a subdivision, and the (example) partition into the actual sub-faces s_i.

The analysis here is predicated upon the assumption of exact interval arithmetic (i.e. ignoring the dependency problem), which is in general not delivered in practice. To cater for interval extensions of functions, the individual solution sets may need to be expanded; also their extent (and the extent of overestimation) depends on the size of the (sub-)face in question. However, the approach is still fundamentally applicable.

Linear Case

In the case where all of the f_i are linear, the solution sets of all $f_i(x) = 0$ can be determined exactly, and it is feasible to actually implement some variant of an optimal subdivision strategy. This is of limited merit in itself, since the computation of topological degree for linear systems is easy in any case. However, if a sufficiently good approximation can be found for each solution set over a face, a method based on this optimal strategy may also be applicable for nonlinear systems.

Maximal Projection Case

In the (presumbly rare) case where the solution sets for all of the $f_i(x) = 0$ project onto all the edges of the face, the optimal partitioning technique outlined above can not be

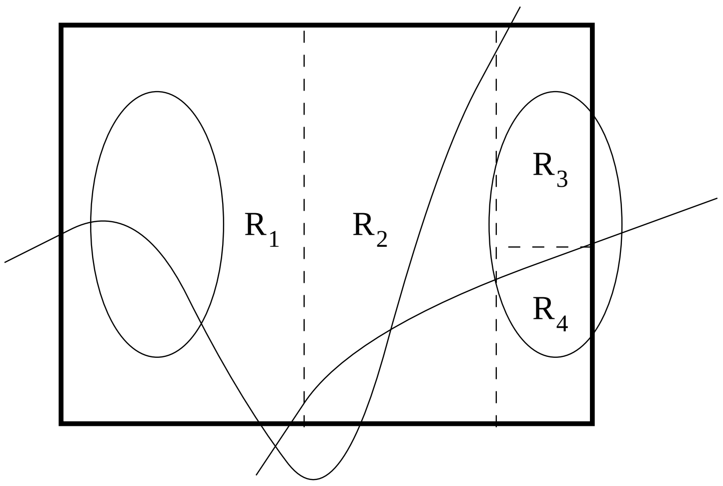


Figure 7.13: Partitioning of the face s into regions $R_1, \ldots, R_4$ in Example 7.5.

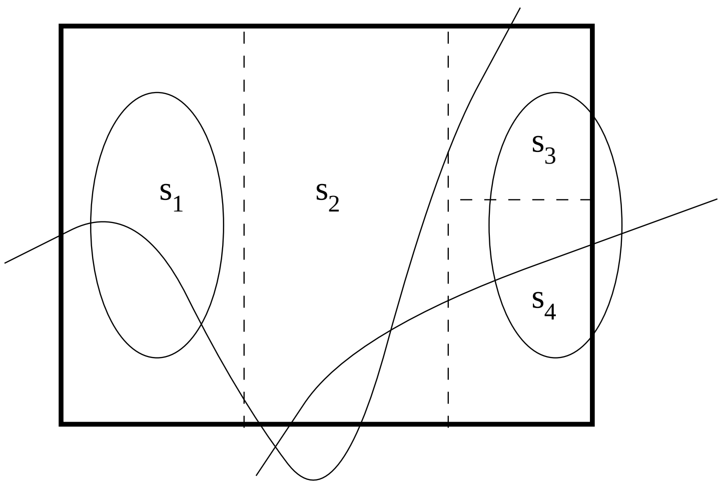


Figure 7.14: Example optimal partitioning of the face s into terminal sub-faces $s_1, \ldots, s_4$ in Example 7.5.

performed, since any $n-2$-dimensional cross-section that is chosen will intersect all solution sets. Such an example is illustrated in Figure 7.15.

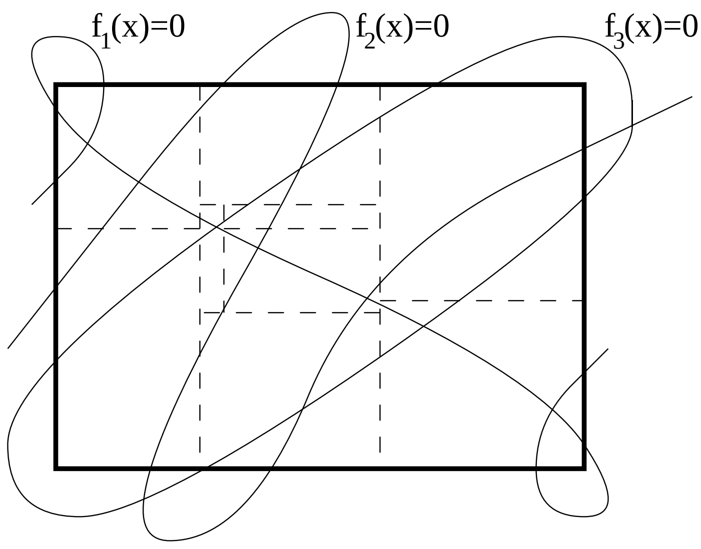


Figure 7.15: Optimal face partitioning in the maximal projection case (where the zero sets of each f_i intersect each edge).

Here, finding a partitioning which is optimal is a significantly harder problem than for the typical case. We may briefly sketch a couple of possible approaches to this. The first is to observe that, although all solution sets project fully onto all edges of the face, it is possible to find sub-faces which do not have this maximal projection property. Any such sub-face, falling into the typical case, can be resolved by the former method. It therefore suffices to find a good way to partition the face into sub-faces with this property — this is in itself an optimal partitioning problem! So we have nested optimal partitioning problems; furthermore it is the case that an optimal partitioning into individually optimally-partitioned sub-faces does not necessarily constitute an overall partitioning that is optimal. This is clearly an extremely hard searching problem.

The other approach that we may mention, like the typical case, relies on enumerating the smallest possible subsets of the face which must contain a subdivision, except that instead of starting with a cross-section of the face, we start with a corner. From each corner, we can expand box subsets (which must initially be terminal) until they only just become non-terminal (where the last solution set enters minutely). As before, the remainder of the face can be handled repetetively. The difficulty here is that we can consider expanding these box subsets in more than one dimension (for faces of dimension 2 or more), so that there are more degrees of freedom in determining such minimally non-terminal regions. (In the typical case, these regions were uniquely determined from any given cross–section, leading to a single partitioning.) The number of remaining divisions that may be required is dependent on the choice of initial region. The choice of each region can be described by $n-2$ variables, and we may need many regions, so we have a nonlinear minimisation problem. Since it is likely that the system of equations and inequalities which governs such an optimal partitioning may well be larger and more complicated than the original system under examination, the effort required to solve it would very likely not be worthwhile.

In conclusion, it is plausible to outline strategies that give very good face partitionings — optimal ones in many cases — but not strategies that give provably optimal ones in all cases. Although the abstract problem of finding optimal partitionings is useful for motivating the design of practical algorithms, we have seen here that the problem of finding an optimal face partitioning can be more difficult to solve than the entire original degree computation. In terms of reducing computational effort, therefore, any attempt at a comprehensive optimal face–partitioning algorithm would be counterproductive — any further analysis here would thus only satisfy an esoteric, not a practical, interest.

7.5.2 Worst Case Analysis

Theorem 7.1. *For any particular instance of a face s and functions f_i given by (7.1) satisfying the preconditions of the topological degree computation algorithm, the number of sub-faces in any optimal partitioning is bounded above.*

Proof: Let

$$M \;=\; \min\left\{ \sum_{i=1}^{n} |f_i(a)| \;\middle|\; a \in s \right\}.$$

Since there are no common zeros of all n functions f_i, $M > 0$.

Choose some $i \in \{1, \ldots, n\}$ and some point $a \in s$. It is assumed that each f_i is continuous over all of s, in particular at a. Therefore $\exists\, \delta > 0$ such that

$$|x - a| < \delta \;\Rightarrow\; |f_i(x) - f_i(a)| < \frac{M}{n}.$$

Let us choose such a δ for each instance, and label it $\delta_{i,a}$.

Let

$$\delta_s \;=\; \min\{\delta_{i,a} \mid a \in s,\; i = 1, \ldots, n\} \;>\; 0.$$

Now let us select any point $a \in s$. $\exists\, j \in \{1, \ldots, n\}$ such that $|f_j(a)| \geq \frac{M}{n}$. (If this were not so, we would have $|f_j(a)| < \frac{M}{n}$ for all j, which would contradict the definition of M.) For all x in the neighbourhood $N_{\delta_s}(a)$, $|f_j(x) - f_j(a)| < \frac{M}{n}$. Therefore for all such x, $|f_j(x)| > 0$. Any sub-face chosen within $N_{\delta_s}(a)$ will therefore be terminal. A sub-face centred on a, say, with each interval width equal to $\sqrt[n]{\delta_s}$ will suffice.

We can place a lattice of points, with lattice width $\sqrt[n]{\delta_s}$ in each dimension, over the whole of s, and partition s by assigning one sub-face to be centred over each lattice point. The size of each sub-face guarantees that it is terminal. If $s = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$, an upper bound on the number of such subfaces is

$$\prod_{i=1}^{n} \left\lceil \frac{\overline{x}_i - \underline{x}_i}{\sqrt[n]{\delta_s}} \right\rceil. \quad \square$$

This (crude) upper bound limits the number of sub-faces required in any optimal (or worthwhile — see below) subdivision for any given problem, and proves that the problem of face subdivision is solvable with finitely many sub-faces.

Can we find a general upper bound on the number of necessary sub-faces for a broad category of problems? To address this, we define the following:

Definition 7.4 (Crucial box)**.** *Given an instance of a face s and functions f_i as before, the* crucial box *of s is the smallest (with respect to the largest component interval width) box subset of s that is non-terminal.*

The crucial box must therefore contain a subdivision, but if it is small, it is a region which is unlikely to contain a subdivision which is chosen at random, which may be problematic.

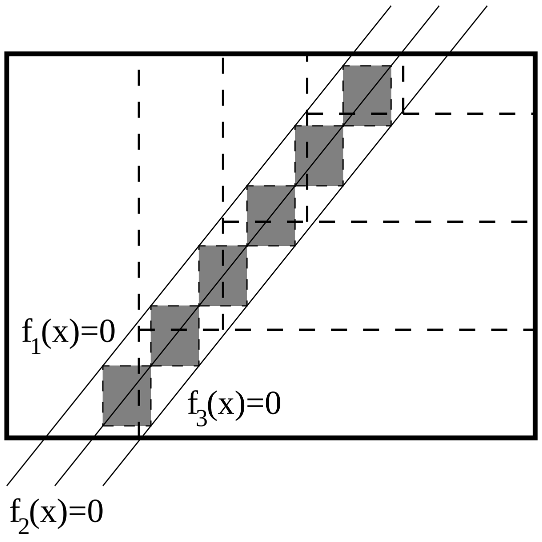


Figure 7.16: Crucial boxes (shaded) and optimal partitioning (dotted lines) in troublesome linear case.

It is possible (although it is unlikely to occur in practice) to construct an example face which has many crucial boxes. For example, a 2-dimensional face in $\mathbb{R}^3$ intersected by the zero sets of three similar linear functions, illustrated in Figure 7.16. If these zero sets (straight lines) are close and parallel, we can have many small crucial boxes whose corresponding component intervals do not intersect at all. In this case, it is not possible to construct a subdivision which enters more than one of the designated crucial boxes. Therefore, the number of subdivisions (and sub-faces) required is bounded below by the number of crucial boxes. Problems can be chosen which have arbitrarily many (arbitrarily small) such crucial regions, for example by making the three straight lines arbitrarily close together. In other words, for any proposed general fixed upper bound on the number of required sub-faces, a problem can be (quite easily) constructed which disproves it. The minimum Hausdorff distance between any two solution sets within the face would seem to be a reasonable metric for the severity of this problem.

In the parlance of the proof for the individual case above, the size of the crucial box is related to the absolute joint function minimum M, and also to the bounds on the partial derivatives of the f_i over the face. For a category of problems with a concrete (non-zero) lower bound on M, and upper bounds on the absolute value of the partial derivates, an upper bound on the number of subdivisions does exist. Where the partial derivatives can be made arbitrarily large, or M arbitrarily small (as in the category of problems with parallel linear solution sets) there is no such bound.

7.5.3 Robustness of Subdivision Strategies

We have seen above that any given face subdivision problem is solvable with finitely many subdivisions and subfaces. This does not guarantee, however, that all subdivision algorithms will terminate with a finite number of sub-faces processed. In the same way that it is possible to have an infinite descending sequence of real numbers with a lower bound, it is possible to subdivide a face in a manner which removes increasingly smaller sub-faces ad infinitum, but which always leaves a certain region undivided. If the untouched region is non-terminal, the process will not terminate.

Definition 7.5 (Robust). *A face subdivision algorithm is* robust *if, after having performed N subdivisions, the maximum component interval width w_N of any distinct remaining non-terminal regions is related to N with the property that $w_N \to 0$ as $N \to \infty$.*

A robust algorithm therefore has the property that any remaining non-terminal 'problem' region can be made arbitrarily small by performing a finite number of subdivisions. Since we can place a limit on the smallness of any crucial box, remaining regions can be made smaller than this, at which point they must be terminal. A robust algorithm is therefore terminating, at least in theory. If we are using an inclusion isotone (see Definition 2.12) interval arithmetic with the property that, for a box $\mathbf{X}$ and well-conditioned function f, the computed value of the interval extension $\mathbf{f}(\mathbf{X})$ tends to the box enclosure of the true value of $f(\mathbf{X})$ as the size of $\mathbf{X}$ tends to zero, the algorithm will be terminating in practice, since we will always be able to find sufficiently small sub-faces which are deemed, even with imperfect interval evaluation, not to require subdivision, in finite time.

It should be possible to construct an example problem which will cause failure to terminate for a non-robust subdivision algorithm. Bisection is an example of a robust subdivision algorithm. An algorithm which only ever divides a face in one dimension would be non-robust.

7.5.4 Worthwhile Subdivision — A Realistic Strategy

The abstract strategies for face partitioning discussed above rely upon precise and detailed knowledge of the solution sets for all the $f_i(x) = 0$. We are not likely to have this information, however, nor be able to obtain it easily — if we did then the whole topological degree calculation could likely be performed without recourse to the recursive algorithm at all. It is clear that in practice we must reckon with not having access to such knowledge. As an aside, such detailed analysis of the function behaviour in the interior of a face also sits rather awkwardly with the principle of boundary determinism, which is fundamental to topological degree (cf. Subsection 4.2.1).

Some form of analysis of the behaviour of the functions within the box (or a face) may still be useful, however. Such analysis has a computational cost, so when trying to minimise the overall computational effort for processing a face, a balance must be drawn between the cost of choosing the point(s) of subdivision, and the cost of the mechanistic subdivision itself. A method of subdivision point selection is only worthwhile if it is at least as cheap as the probable savings made by reducing the number of faces to be processed overall.

As has been illustrated above, it is possible to partition a face in an elaborate fashion, but a simple computational scheme is to pick a single subdivision point, and subdivide the face around that. (Each component interval is divided around the point, so that an $n-1$-dimensional face is partitioned into 2^{n-1} subfaces.) This is also the scheme used in the original outline method of Aberth [Abe94]. It seems reasonable that any heuristic method which selects a single point is likely to be less costly than a method to select a number of points (or some other hybrid structure which can represent a partitioning).

Definition 7.6 (Worthwhile subdivision)**.** *Given an $n-1$-dimensional face s and functions f_i as above, consider the topology of the face where the loci of $f_1(x) = 0, \dots, f_n(x) = 0$ are drawn upon it. A* worthwhile subdivision *of s is a partition of it into 2^{n-1} sub-faces about a point p, all of which are topologically different (in general, they have a simpler topology) than the original face. Therefore a subdivision is only non-worthwhile if there is at least one sub-face which is homeomorphic to the original face with respect to the solution sets inside and on each edge (i.e. the corresponding child faces are likewise homeomorphic).*

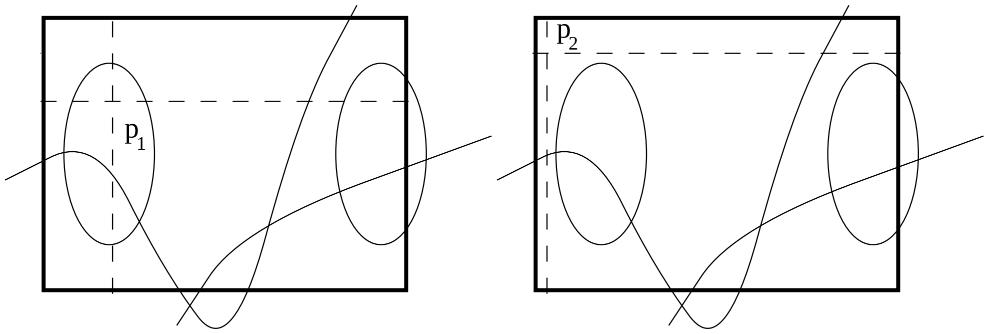


Figure 7.17: Worthwhile (left) and non-worthwhile (right) subdivisions of a face.

An example of a worthwhile and non-worthwhile subdivision of a face is given in Figure 7.17.

Intuitively, therefore, with a worthwhile subdivision, it seems we are making some progress towards splitting the face into a series of manageable chunks, even if further subdivisions may be required. More formally:

Conjecture 7.3. *A worthwhile subdivision can always be performed on a non-terminal face.*

Conjecture 7.4. *A sequence of worthwhile subdivisions on a face will lead to completion of the degree computation (i.e. all final sub-faces are terminal) in a finite number of steps.*

If both of these conjectures are true, then firstly the existence of a worthwhile subdivision algorithm is plausible, and secondly, any such algorithm will be terminating.

7.5.5 Identifying the Fatal Box — A Key Sub-Problem

Definition 7.7. *Considering an $n-1$-dimensional face s arising in a topological degree computation of $\mathcal{F}$ over some box $\mathbf{X}$ in R^n, the* fatal box *of s is the smallest box subset of the face containing all points for which all but one of the functions $f_1, \ldots, f_n$ are simultaneously zero. The fatal box is undefined if there are fewer than two such distinct points.*

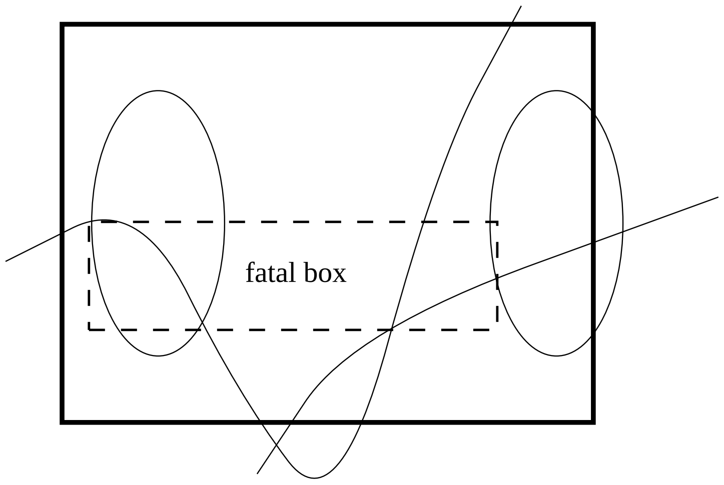


Figure 7.18: Fatal box of a face.

An example fatal box of a face is depicted in Figure 7.18.

Any subdivision point chosen inside the fatal box will lead to a worthwhile subdivision of the face, although the fatal box need not be non-terminal (but probably is). It is proposed that new subdivision methods might rely on either computing an estimate to the fatal box, and selecting a subdivision point inside, or computing a subdivision point directly which is likely to lie within the fatal box.

7.5.6 Is Bisection Optimal?

In the absence of any educated guess as to where might be a good place to put a subdivision point, locating it squarely in the centre of the face would seem to be an eminently sensible strategy. This partitions an m-dimensional face into 2^m sub-faces of equal size. Offering the maximum rate of largest sub-face shrinkage with respect to search depth, it seems a plausible scheme to minimise the search depth, since, in general, the probability of a sub-face requiring further subdivision is proportional to its size.

Bisection is the usual default strategy for most branch-and-bound schemes for solving systems of equations. In some cases, it can be proved to be optimal, e.g. finding the root of a univariate function over an interval, based on function evaluations [Sik82, Sik97].

The likely size (and existence) of the fatal box of a face depends upon the class of functions f_i. For a general analysis, though, let us assume that the fatal box is located arbitrarily within the face, its vertices designated randomly. (This assumes that there is no correlation between the extent of the fatal box and the dimension; it is however possible that such a relationship exists.) For an individual component interval of the face, the probability of the two endpoints of the corresponding component interval of the fatal box occuring in opposite halves of it is $\frac{1}{2}$. This is equivalent to the probability that the interval field of the fatal box includes the midpoint of the interval field for the face. Compounding this probability over all interval variables, we have the probability of the midpoint of an $n-1$-dimensional face occurring within the fatal box as $\frac{1}{2^n-1}$, i.e. it becomes increasingly unlikely with higher dimension.

Bisection of faces does not nearly always produce the minimum number of sub-faces, as illustrated by an example (see Figure 7.19; the three zero sets, two perpendicular lines and a circle, are given by continuous lines and the (successive) subdivisions by dotted lines). In such cases, it seems that perhaps an informed choice of subdivision point would prove more effective.

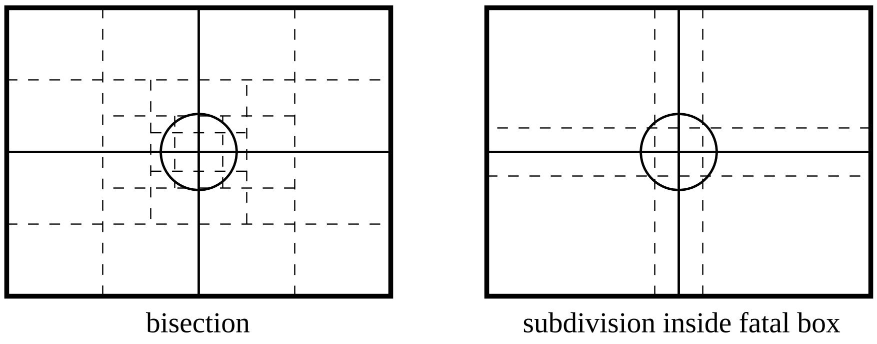


Figure 7.19: Bisection is not always optimal.

Bisection is robust (cf. Definition 7.5). Upon each face subdivision, the width of each interval field of each sub-face is halved — the sub-face size decays exponentially (asymptotically approaching zero) with subdivision depth.

7.5.7 Random Subdivision

It is possible to pick an arbitrary point of subdivision within the face totally at random. While this is likely not a good strategy in itself, it is useful for the purposes of comparing against other variants. If a heuristic subdivision algorithm which picks some subdivision point in a probabilistic way proves to be better in practice than a random choice, then that is evidence of merit. (Although if still worse than bisection, it is perhaps of questionable merit.)

Let us again consider an arbitrarily located fatal box within a face, and calculate the probability of a random subdivision point occurring within it. In the one-dimensional case, let the face be given by the interval $[a, b]$ and the fatal box by the interval $[x, y]$, where x and y are random points within $[a, b]$. What is the probability of a random (subdivision) point in $[a, b]$ being contained within $[x, y]$? The expected width of the fatal box, $w([x, y])$ can be determined as follows:

$$\begin{aligned}
E(w([x,y])) &= \frac{1}{(b-a)^2}\int_a^b\int_a^b |y-x|\,dxdy \\
&= \frac{1}{(b-a)^2}\int_a^b \left(\int_a^y (y-x)dx + \int_y^b (x-y)dx\right) dy \\
&= \frac{1}{(b-a)^2}\int_a^b \left[yx - \frac{1}{2}x^2\right]_a^y + \left[\frac{1}{2}x^2 - yx\right]_y^b dy \\
&= \frac{1}{(b-a)^2}\int_a^b y^2 - (a+b)y + \frac{1}{2}(a^2+b^2)dy \\
&= \frac{1}{(b-a)^2}\left[\frac{1}{3}y^3 - \frac{a+b}{2}y^2 + \frac{1}{2}(a^2+b^2)y\right]_a^b \\
&= \frac{1}{(b-a)^2}\left(\frac{1}{3}b^3 - b^2a + a^2b - \frac{1}{3}a^3\right) \\
&= \frac{1}{3}(b-a).
\end{aligned}$$

The probability of a random point in $[a, b]$ also being contained within $[x, y]$ is therefore $\frac{1}{3}$. Extending to an $n-1$-dimensional face, the probability of a random point being contained within an arbitrary fatal box is $\frac{1}{3^{n-1}}$. This compares poorly to bisection and demonstrates that bisection is a clearly better strategy than choosing a random subdivision point.

Random subdivision is robust, assuming a true random choice of subdivision point. The probability of an infinite sequence of random real numbers chosen inside a given interval all excluding any given non-zero sub-interval is zero. Therefore the size of sub-faces will tend to zero as subdivisions are chosen randomly. A poor pseudo-random choice of subdivision points could however cause the algorithm not to terminate.

7.5.8 A Newton Method

Here we present an outline of a proposed heuristic method for choosing a subdivision point. Given the problem setup as before and an $n-1$-dimensional face s, it is based on the following hypotheses:

- The point in s which minimises the value of $|f_1| + \ldots + |f_n|$ is more likely to yield a worthwhile subdivision than a point chosen at random.
- A projection of the nearest solution to $f(x) = 0$ onto s is a reasonable approximation to such a point.

The method is as follows (and is also illustrated in Figure 7.20):

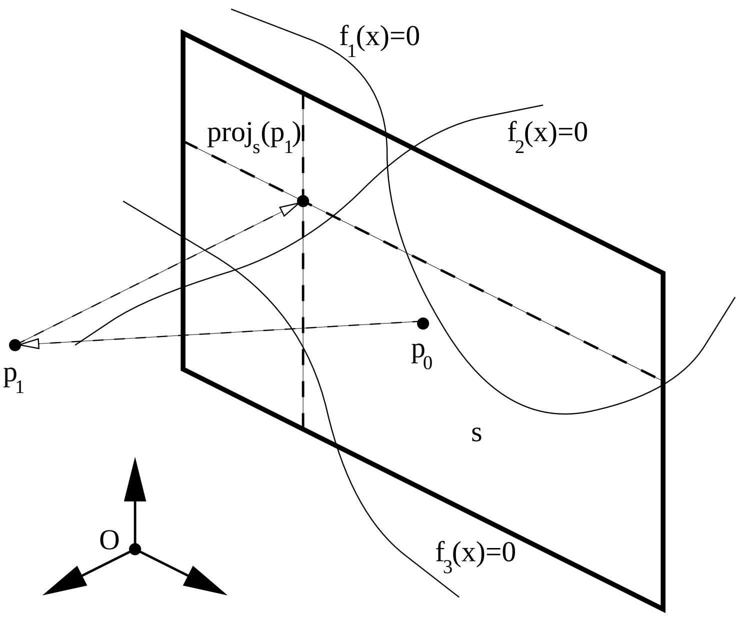


Figure 7.20: Newton method: an example 2-dimensional face in $\mathbb{R}^3$.

1. Set p_0 to be the midpoint of s.
2. Perform one Newton iteration with p_0, giving p_1 as an estimate to a solution of $f(x) = 0$.
3. Project p_1 onto the (hyper-)plane containing s.

4. If this new point lies within s, use it as a subdivision point. Otherwise, resort to using p_0 (the midpoint).

A variant of this strategy would be to use the Newton method on all possible subsystems with one function f_i removed, and by taking the box hull of their projections onto s, thereby compute an estimation for the fatal box, from which the midpoint is taken.

7.6 Conclusions

There may be both a theoretical and a practical interest in topological degree computation; let us deal with both in turn.

In Section 7.2 we listed a number of candidate factors which may determine the computational effort for a problem instance. Before proceeding further, it may be wise to distinguish between trivial and non-trivial instances. In trivial cases at least one component function can be bounded away from zero over the box; in such a case no subdivision is required. Let us therefore consider contributing factors to the complexity for non-trivial instances, i.e. where subdivision is required and/or at least one face or sub-face is designated. Based on the analysis performed in Sections 7.3 and 7.4, we may estimate an ordering for these factors, in order from most deterministic to least deterministic, as follows:

- **Behaviour of enclosures for component functions over the box**: How wide are the enclosures for the ranges of the component functions, how quickly do they contract upon subdivision, and how quickly can zero be excluded? As noted above, for non-trivial cases, all such enclosures are assumed to contain zero. Where all the enclosures still stubbornly include zero even after a number of subdivisions, few of the sub-faces are terminal and the local branching factor is high, requiring a great number of sub-faces to be processed.

  The answers to these questions are actually dependent upon a number of sub-factors (nonlinearity of functions, interval overestimation, and proximity of solutions to the boundary), which are listed below:

- **Nonlinearity of component functions**: Where the component functions are highly nonlinear, the loci of their solution sets have a more complicated structure, meaning that potentially several distinct worthwhile subdivisions are required.

- **Problem dimension (number of variables)**: The dimension n is unavoidably strongly related to the complexity. Although we have posited that the average branching factor is independent of n, and the number of starting faces increases only linearly with n, where subdivision is required the number of sub-faces generated by a subdivision increases exponentially with n.

- **Overestimation in interval arithmetic implementations of functions**: The effect of the dependency problem in interval arithmetic would seem to have a moderate impact (although this factor has not been explicitly investigated), making somewhat more subdivisions necessary before zero can be excluded from a component function enclosure.

- **Proximity of solution(s) to the boundary of the box**: Where a common zero of all component functions is close to a particular face, the face is likely to have a small crucial box. The smaller a crucial box, the more difficult it is to subdivide within it, or the larger the sequence of subdivisions required.
- **Number of solutions to the system within the box**: This has only an indirect effect — the extent of each component function zero set appears to be more important. Indeed, one can construct trivial instances (e.g. Example 7.1) with multiple solutions within the box.
- **Box volume**: If we assume that the problem is non-trivial, then the size of the box appears not to matter much. **However, the box volume and location may play a major role in determining whether a problem is non-trivial or not in the first place.**

From a practical perspective, and given the subject area of this thesis, it is envisioned that the topological degree is principally useful as a component of branch-and-bound methods for solving systems of equations. It is difficult to avoid the conclusion that the generally high and unpredictable complexity of the method remains a fundamental limitation for its use as a low-level component (e.g. a repetetive root isolator or verifier for sub-boxes) within a branch-and-bound scheme. Nevertheless, where the topological degree computation is fast, the root-counting property may be a powerful asset. The issues are not only how to speed up the degree calculation, but also when to apply it judiciously in such a scheme; application to every single sub-box may be excessive. It may be considered as a root-designation, preprocessing, or verification tool for other iterative or subdivision-based methods, which is not designed to be used in every iteration. For example, it may be suitable for application to boxes of terminal width.

Some further remarks on the applicability of the topological degree method are given in Chapter 11.

8 Solution of Systems of Polynomial Equations

In this chapter we present a method for finding all solutions to a system of polynomial equations within a given box [GS01b]. Central to this algorithm is the expansion of the component multivariate polynomials into Bernstein polynomials, which were introduced in Chapter 3; see also Chapter 9. After subdivision, a union of small sub-boxes which enclose the set of solutions is generated by way of an existence test. Some rules for selecting the bisection direction are proposed, and tested with some numerical examples. Additionally, a preconditioning step is developed in order to reduce the computational cost of the existence test [GS01a]. Under the categorisation proposed in Subsection 5.3.1, this is a robust global solver utilising interval arithmetic.

As in Chapter 5, let us write our system of polynomial equations as $\mathcal{P} = 0$, where $\mathcal{P} : \mathbb{R}^n \to \mathbb{R}^n$ is comprised of n polynomials p_i, $i = 1, \ldots, n$, in the real variables $x_1, \ldots, x_n$. Let a box $\mathbf{X}$ in $\mathbb{R}^n$ be given and let the maximum degree of all the p_i be $l = (l_1, \ldots, l_n)$. We wish to compute the set of all solutions (in this case, tight box enclosures for each individual solution) to the system of equations $\mathcal{P} = 0$ within $\mathbf{X}$, cf. (5.2). The existence test and preconditioning may be more broadly applied to systems of continuous nonlinear equations $f_i(x) = 0, i = 1, \ldots, n$, although in this case the Bernstein expansion may no longer be (directly) employed.

8.1 Algorithm

The algorithm (cf. [GS01b]) consists of two main parts. In the first stage the starting box, together with corresponding Bernstein coefficients (cf. Subsection 3.1.2) of each of the polynomials, is successively subdivided into sub-boxes. After each subdivision, a test is performed to prune out those sub-boxes which cannot contain a solution. The second stage begins once all sub-boxes are sufficiently small; it comprises an existence test.

In the case that the number of the equations does not equal the number of the variables, the first stage of the algorithm is still applicable; it is possible to obtain an enclosure for the solution set as a union of boxes. However the existence test could no longer be applied.

8.1.1 Subdivision and Pruning

Firstly, the Bernstein coefficients of each polynomial p_i, $i = 1, \ldots, n$, over the starting box $\mathbf{X}$ are computed and stored (cf. the algorithm in Subsection 3.3.1). The domain is then repeatedly subdivided into sub-boxes, and the Bernstein coefficients of the polynomials over these sub-boxes are computed simultaneously (cf. the algorithm in Subsection 3.3.2 and the description of the sweep procedure, below).

An *exclusion test* is performed on each box: If any one of the polynomials p_i has an entire set of Bernstein coefficients which are either wholly negative or wholly positive (cf. [BCR08]), then we can be sure from the range enclosing property of the Bernstein expansion (3.17) that that polynomial cannot attain zero over the box, and that therefore the box cannot contain a solution to the system. The infeasible boxes which fail this exclusion test are thus discarded and do not need to be subdivided further. In this fashion the total number of boxes to be processed is kept down to a manageable level.

This stage of the algorithm is complete once all of the boxes have side lengths which are smaller than a predetermined width $\varepsilon > 0$. There are now zero or more boxes of very small volume which *may* contain a solution to the system.

Sweep Procedure

The bounds obtained by the inequalities (3.17) are in general tightened if the box $\mathbf{X}$ is bisected into sub-boxes and the Bernstein expansion is applied to each polynomial p_i over these sub-boxes. A *sweep* in the rth direction ($1 \leq r \leq n$) is a bisection of a box perpendicular to this direction and is performed efficiently by recursively applying a linear interpolation according to the algorithm in Subsection 3.3.2, where $\lambda = \frac{1}{2}$. A simple example sweep in the case of two variables is depicted in Figure 3.4. A single sweep requires $O(\hat{l}^{n+1})$ additions and multiplications, where $\hat{l} = \max_{i=1}^{n} l_i$, cf. [ZG98].

Subdivision Direction Selection

A heuristic sweep direction selection rule is employed in an attempt to minimise the total number of sub-boxes which need to be processed. For example, a rule may favour directions either in which the polynomials have large partial derivatives and/or in which the box edge lengths are larger, in order to avoid repetitive sweeps in a single direction. Let the current box be $\mathbf{X}_{cur} = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$, where the number of subdivisions already performed on it (i.e. the current depth in the subdivision tree) is *depth*. Following in part the proposals of Ratz and Csendes [RC95], the method is tested with the following variant rules for the selection of the subdivision direction r:

- **C**: Each direction is chosen in sequence and the sequence is repeated. Each direction is thus afforded an equal bias (in the long run) and this rule is used as a control.

$$r := (\mathit{depth} \bmod n) + 1.$$

- **D1**: We compute an upper bound for the absolute value of the partial derivative for each direction on each polynomial over the box. In each direction, we sum these values over all polynomials, and select the direction for which the product of box edge length and partial derivative sum is maximal. Tight bounds on the partial derivatives are readily computed from the available Bernstein coefficients according to the algorithm in Subsection 3.3.4.

$$r \text{ maximises } \left(\sum_{i=1}^{n} \left| \frac{\partial p_i}{\partial r}(\mathbf{X}_{cur}) \right| \right) (\overline{x}_r - \underline{x}_r) \, .$$

- **D2**: As **D1**, except that we take the maximum of the upper bounds for the absolute value of the partial derivatives over all polynomials for each direction (instead of their sum), and then multiply by the box edge length, as before.

$$r \text{ maximises } \left(\max_{i=1,\ldots,n} \left| \frac{\partial p_i}{\partial r}(\mathbf{X}_{cur}) \right| \right) (\overline{x}_r - \underline{x}_r) \, .$$

8.1.2 Existence Test

At the start of the second stage, we have remaining zero or more (likely several or many) *ε-boxes*, viz. boxes with side lengths smaller than ε, that are of sufficiently small volume. In order to separate those boxes which do not contain a solution from those which do, an *existence test* is performed.

We use the existence test of C. Miranda [Mir41] (see below). Miranda's theorem provides a generalisation of the intermediate value theorem (in particular the fact that if a univariate continuous function f attains values of opposite sign at the two endpoints of an interval, then the interval must contain a zero of f) to many dimensions. The theorem bears a close relationship to Brouwer's fixed point theorem (Theorem 4.1) and can be proven using it.

Theorem 8.1 (Miranda). *Let $\mathbf{X} = [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]$. Denote by $\mathbf{X}_i^- := \{x \in \mathbf{X} \,|\, x_i = \underline{x}_i\}$ and $\mathbf{X}_i^+ := \{x \in \mathbf{X} \,|\, x_i = \overline{x}_i\}$ the pair of opposite parallel faces of $\mathbf{X}$ perpendicular to the ith coordinate direction.*

Let $\mathcal{F} = (f_1, \ldots, f_n) : \mathbf{X} \to \mathbb{R}^n$ be a continuous function. If there is a permutation $(v_1, \ldots, v_n)$ of $(1, \ldots, n)$ such that

$$f_i(x) f_i(y) \leq 0 \text{ for all } x \in \mathbf{X}_{v_i}^-, \, y \in \mathbf{X}_{v_i}^+, \;\; i = 1, \ldots, n, \tag{8.1}$$

then the system of equations $\mathcal{F}(x) = 0$ has a solution in $\mathbf{X}$.

A short proof of Miranda's Theorem was given by M. N. Vrahatis [Vra89], cf. [Vra95].

It should be noted that the result provides a proof of existence, but not a proof of uniqueness. Neither is this a necessary condition for a solution, so boxes for which the test (8.1) fails are not proven to lack one. However if the test is augmented with, e.g., a proof of monotonicity (which can easily be obtained by bounds on the Bernstein enclosure of the partial derivatives), which generally holds for small boxes, then a non-existence proof is achieved.

The test (8.1) requires the computation of the ranges attained by each polynomial over the $2n$ faces of each box, in numerous permutations. This can be achieved very cheaply by utilising the corresponding subarrays of Bernstein coefficients; the Bernstein coefficients of each p_i on the faces of $\mathbf{X}$ are known once the Bernstein coefficients of p_i on $\mathbf{X}$ are computed, cf. Lemma 3.1 in Subsection 3.2.2.

At the end, there are zero or more *ε-boxes* which are thus *guaranteed* to contain a solution to the system.

8.2 Examples

We first begin with a simple illustrative example:

Example 8.1. *Let*

$$\begin{aligned} p_1(x,y) &= 0.0625x^2 + 0.1111y^2 - 1, \\ p_2(x,y) &= 0.06x^2 + 0.12y^2 - 1. \end{aligned}$$

The zero sets of p_1 and p_2 are two very similar ellipses which intersect at four distinct points (see Figure 8.1). At the end of the subdivision and pruning step, after subdividing the starting box of $[-5,5] \times [-5,5]$ eight times in each direction, ε-boxes of width approximately 0.04 remain. Due to the proximity of the zero sets, we have many dozens of such boxes where both p_1 and p_2 cannot be bounded away from zero (see Figure 8.1, bottom-left). However only those four boxes which contain the four solutions pass the Miranda test; the others may be discarded.

We can see here that the Miranda test coupled with the Bernstein enclosure can be very effective in correctly filtering small candidate boxes for solutions.

The method was further tested for some of the sample problems used by Sherbrooke and Patrikalakis [SP93] (**SP**) and Jäger and Ratz [JR95] (**JR**); see Table 8.1. The polynomial in **SP2** is the Wilkinson polynomial.

The maximum subdivision depth is chosen to achieve the same tolerance as used in **SP** and **JR**, respectively. In each case, we record in Table 8.2 the total number of boxes processed (which is equal to twice the number of sweep operations, plus one), the number of Miranda tests performed, and the execution time (averaged over 5 repeat runs). These examples were run on a old PC equipped with a 450MHz Pentium III processor (broadly comparable but up to an order of magnitude faster than used in **SP** and **JR**).

Some categories of problems (e.g. those for which all the component polynomials are similar and have coefficients of the same magnitude) seem to require subdivision in all directions equally; for these cases we observe no appreciable difference in the output data between the control (**C**) and the derivative-based methods (**D1**, **D2**). In other cases we notice that the choice of the sweep direction based on the absolute value of the partial derivatives is effective in reducing the overall number of boxes that are processed and the number of Miranda tests required. There is very little difference between the sum (**D1**) and maximum (**D2**) variants. A choice of sweep direction based on maximal box edge length seems obvious, but this metric is seen to be improved by weighting according to (i.e. multiplying by) the magnitude of the partial derivatives — it is better to contract faster with respect to those variables that have a more dominant effect on the values attained (locally) by each component polynomial.

A greater variance in the number of Miranda tests (and the computation time) between the methods may be observed for a range of differing subdivision depths, corresponding to alternative tolerances not used in **SP** and **JR**. The timings compare mostly favourably to those reported by **SP** and **JR**, but we should note that the processor capability available to us is approximately an order of magnitude faster.

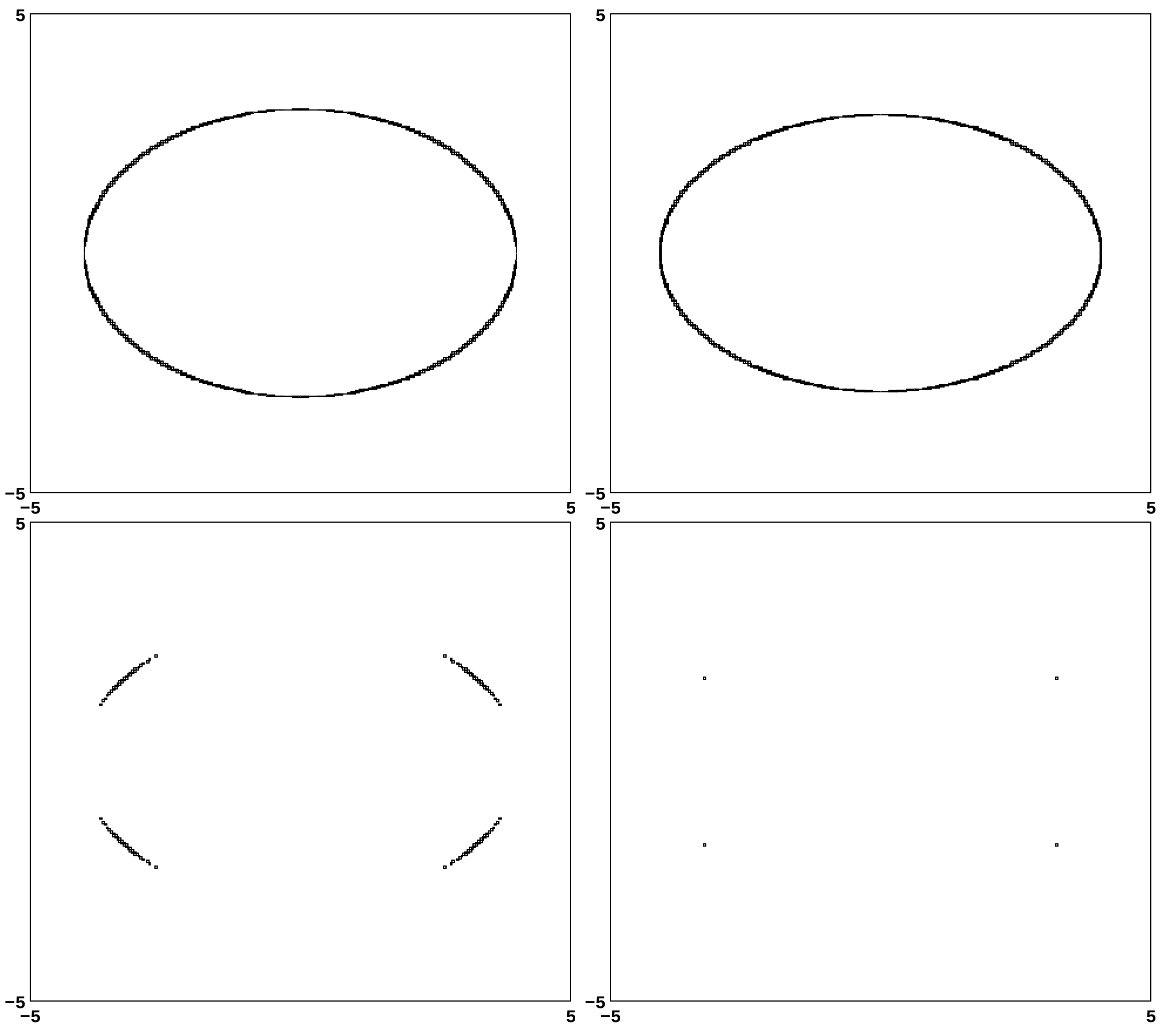


Figure 8.1: Example 8.1: ε-boxes enclosing the zero set of p_1 (top-left), the zero set of p_2 (top-right), the intersection of both zero sets (bottom-left), and those for which the Miranda test succeeds (bottom-right).

Name	**X**	Tolerance	Max. subdivision depth	#Solutions
SP1	$[0,1]^2$	10^{-8}	53	1
SP2	$[0,21]$	10^{-7}	28	20
SP3	$[0,1]^2$	10^{-8}	53	9
SP4	$[0,1]^6$	10^{-8}	159	4
SP5	$[0,1]^2$	10^{-14}	93	1
JR2	$[-1,1]^3$	10^{-12}	123	2
JR4	$[-1,1]^3$	10^{-12}	123	8

Table 8.1: Starting boxes, subdivision depths, and numbers of solutions for the example systems of polynomial equations.

Example	Method	**C**	**D1**	**D2**
SP1	Number of boxes	205	183	183
	Miranda tests	2	1	1
	Time (s)	0.02	0.02	0.02
SP2	Number of boxes	983	identical results	
	Miranda tests	20	(sweep in one direction only)	
	Time (s)	0.12		
SP3	Number of boxes	2493	2245	2245
	Miranda tests	28	20	20
	Time (s)	0.29	0.27	0.29
SP4	Number of boxes	6901	6315	6789
	Miranda tests	15	12	9
	Time (s)	11.99	10.15	8.69
SP5	Number of boxes	5141	5131	5131
	Miranda tests	31	32	32
	Time (s)	0.80	0.80	0.85
JR2	Number of boxes	3705	3571	3655
	Miranda tests	16	16	16
	Time (s)	0.77	0.77	0.82
JR4	Number of boxes	8759	5895	6173
	Miranda tests	27	22	22
	Time (s)	3.05	2.23	2.49

Table 8.2: Total numbers of boxes and Miranda tests for the example systems of polynomial equations.

8.3 Reduction of Computational Cost and Preconditioning

We may improve the method of Section 8.1 by an optimisation of the permutation-checking procedure and the addition of a preconditioning step (cf. [GS01a]). The former reduces the computational complexity and the latter enables the avoidance of failure cases.

The results of this section are valid for general continuous functions.

8.3.1 Permutation Checking

A straightforward implementation of the Miranda test (Theorem 8.1) requires all possible permutations $(v_1, \ldots, v_n)$ of $(1, \ldots, n)$ to be checked; this exhibits complexity $O(n!)$ with respect to each individual component test (8.1), since in the worst case this test has to be performed $n!$ times. As it stands, therefore, the test is only feasible for small n.

Here we propose a more efficient method for checking all permutations. We begin with a partial uniqueness result: Suppose that the ith part of a component Miranda test (8.1), i.e. the test for the component function f_i, for some $i \in \{1, \ldots, n\}$, is successful for some $k \in \{1, \ldots, n\}$, i.e.

$$f_i(x) f_i(y) \leq 0 \quad \forall x \in \mathbf{X}_k^-, \ \forall y \in \mathbf{X}_k^+. \tag{8.2}$$

Assume that there exist $j \in \{1, \ldots, k-1, k+1, \ldots, n\}$ and $\mathbf{X}_k^* \in \{\mathbf{X}_k^-, \mathbf{X}_k^+\}$ such that

$$f_i(x) \neq 0 \quad \forall x \in \mathbf{X}_k^* \cap (\mathbf{X}_j^- \cup \mathbf{X}_j^+). \tag{8.3}$$

Note that the intersection of two differing non-opposite faces of a box is their common child face (using the terminology of Subsection 4.4.1), which is a face of one lower dimension, i.e. dimension $n-2$. In the polynomial case, this condition can easily be checked by inspection of the Bernstein coefficients on the two corresponding $(n-2)$-dimensional faces of the array of Bernstein coefficients of the polynomial over $\mathbf{X}$, cf. Lemma 3.1. It follows then from (8.2), noting that f_i cannot attain both positive and negative values over $\mathbf{X}_k^*$, that

$$\exists v \in \mathbf{X}_k^* \cap \mathbf{X}_j^-, \, w \in \mathbf{X}_k^* \cap \mathbf{X}_j^+ : \ f_i(v) f_i(w) > 0,$$

so that the component Miranda test is sure to fail for f_i on the face pair $(\mathbf{X}_j^-, \mathbf{X}_j^+)$. Therefore, if condition (8.3) is fulfilled for all other face pairs $(\mathbf{X}_j^-, \mathbf{X}_j^+)$, with $j \neq k$, then the pair $(\mathbf{X}_k^-, \mathbf{X}_k^+)$ is the only one for which the component Miranda test will be successful. However, the condition (8.3) may not be satisfied, cf. Example 8.2. We will consider below how to ensure that it may be fulfilled, by applying preconditioning.

Where (8.3) is satisfied, the computational cost of the overall Miranda test may be reduced, with $O(n^2)$ complexity, as follows: Define an $n \times n$ matrix $M = (m_{ij})$ by

$$m_{ij} := \begin{cases} 1 & \text{if the test of } f_i \text{ on } (\mathbf{X}_j^-, \mathbf{X}_j^+) \text{ succeeds} \\ 0 & \text{fails.} \end{cases}$$

As soon as we find any row or column in M consisting wholly of zeroes, we may terminate since no successful permutation is possible. If each row and column of M contain a 1, we may terminate since there is a successful permutation, and we can conclude that the box under consideration contains a solution of the system.

8.3.2 Preconditioning

It is proposed, amongst others, by J. B. Kioustelidis [Kio78], cf. [MK80, ZN88], that a system of nonlinear equations $\mathcal{F}(x) = 0$ should be preconditioned, i.e., it should be substituted by a system $A\mathcal{F}(x) = 0$, where A is a suitably chosen $n \times n$ matrix. Let $\mathcal{F}$ be differentiable on the box in question $\mathbf{X}$ and suppose its Jacobian $\mathcal{F}' := J(\mathcal{F})$ is nonsingular at $\check{x}$, where

$$\check{x}_i := (\underline{x}_i + \overline{x}_i)/2, \quad i = 1, \dots, n,$$

i.e. $\check{x}$ is the midpoint of $\mathbf{X}$. Then a good choice for A is an approximation to $\mathcal{F}'(\check{x})^{-1}$.

If we have the explicit functions describing all the partial derivatives, then we may compute $\mathcal{F}'(\check{x})^{-1}$ exactly. Otherwise, for polynomials, we may again exploit the easy calculation of the Bernstein coefficients of their partial derivatives, cf. Subsection 3.3.4, in order to compute an interval matrix enclosing the Jacobian over the box. The preconditioning matrix A can be set to the inverse of the midpoint matrix (the matrix obtained by setting all entries of the interval matrix to their midpoints).

The following example illustrates a failure case of the improved permutation checking in Miranda test and the application of preconditioning.

Example 8.2. *Consider the system of polynomial equations $\mathcal{P} = 0$ where $\mathcal{P}(x_1, x_2) = (p_1(x_1, x_2), p_2(x_1, x_2))$ is given by*

$$\begin{aligned} p_1(x_1, x_2) &= x_1^2 + x_2^2 - 1, \\ p_2(x_1, x_2) &= x_1 - x_2. \end{aligned}$$

We wish to determine if there is a solution to the system in the box $\mathbf{X} := \mathbf{I}$. The zero sets of both polynomials are displayed in Figure 8.2. Both p_1 and p_2 exhibit sign changes on both pairs of opposite faces, but either p_1 or p_2 attains the value zero on all four vertices of the box (face intersections), thus (8.3) is not satisfied and the permutation optimisation cannot be applied. Furthermore, the slightest overestimation in the computed interval enclosures for p_1 and p_2 over the faces would negate (8.2) and thus cause failure of the unmodified Miranda test.

The preconditioning matrix is

$$A = \mathcal{P}'(\check{x})^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{pmatrix}.$$

This yields the preconditioned system

$$q_1(x_1, x_2) = \frac{1}{2}\left\{(x_1 + \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 - \frac{3}{2}\right\},$$

$$q_2(x_1, x_2) = \frac{1}{2}\left\{(x_1 - \frac{1}{2})^2 + (x_2 + \frac{1}{2})^2 - \frac{3}{2}\right\}.$$

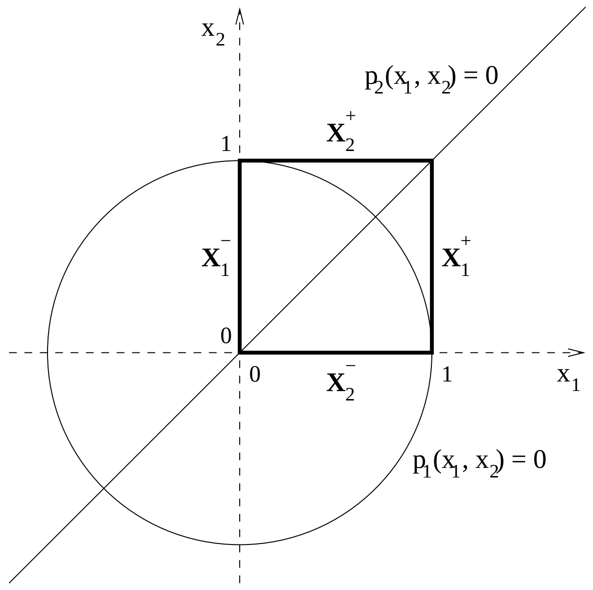


Figure 8.2: An example of failure of the requirement (8.3) for the improved permutation checking scheme and potential failure of the Miranda test.

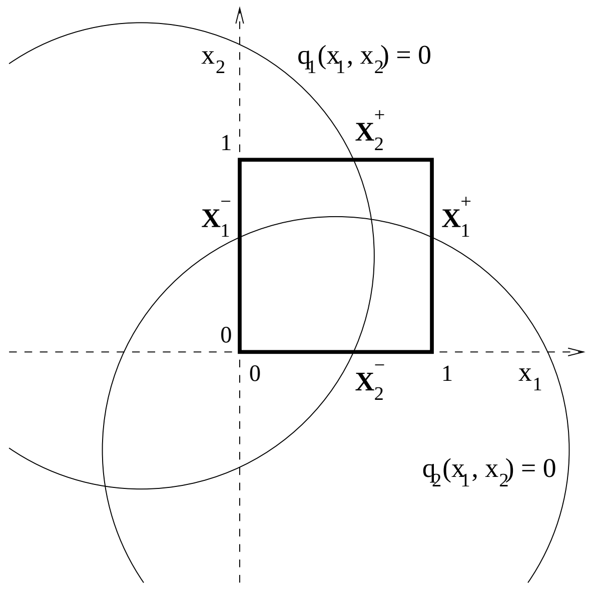


Figure 8.3: The same example after preconditioning.

The zero sets of the two polynomials q_1 and q_2 are displayed in Figure 8.3. Both polynomials have a sign change on one pair of opposite faces instead of at the vertices and condition (8.3) is now satisfied for both. The two solutions of the preconditioned system (both the solution sought inside the unit box and the spurious solution outside it) are of course identical to the solutions of the original system.

Let us consider a preconditioned system of functions $\mathcal{G}$, i.e. $\mathcal{G}(x) = A\mathcal{F}(x)$, where $A := \mathcal{F}'(\tilde{x})^{-1}$ and $\tilde{x}$ is an approximation to x^*, a regular solution to $\mathcal{F}(x) = 0$ and therefore also to $\mathcal{G}(x) = 0$. Zuhe and Neumaier [ZN88] have shown that $\mathcal{G}$ is given by

$$\mathcal{G}(x) \;=\; x - x^* + o(\varepsilon),$$

wherever $x \in \mathbb{R}^n$ with $||x - x^*|| \leq \varepsilon$, for sufficiently small ε. Where this holds, and where $g_i : \mathbf{X} \to \mathbb{R}$ is the ith component function of $\mathcal{G}$, we thus have

$$g_i(x) < 0 \;\; \forall x \in \mathbf{X}_i^- \;\text{ and }\; g_i(x) > 0 \;\; \forall x \in \mathbf{X}_i^+, \;\; i = 1, \dots, n, \tag{8.4}$$

for any box $\mathbf{X}$ with $[\underline{x}_i, \overline{x}_i] = [\check{x}_i - d_i, \check{x}_i + d_i]$, $i = 1, \dots, n$, where d is a vector with

$$|\check{x}_i - x_i^*| < \frac{d}{2}, \;\; i = 1, \dots, n, \tag{8.5}$$

and where $||d||$ is sufficiently small.

Condition (8.5) means that we require the solution x^* to be quite centrally located within the box (i.e. away from its edges), more specifically

$$x^* \in \mathbf{X}_{centre} := [\frac{3\underline{x}_1 + \overline{x}_1}{4}, \frac{\underline{x}_1 + 3\overline{x}_1}{4}] \times \ldots \times [\frac{3\underline{x}_n + \overline{x}_n}{4}, \frac{\underline{x}_n + 3\overline{x}_n}{4}].$$

This condition is not usually satisfied, in fact for an arbitrary example the probability of $x^* \in \mathbf{X}_{centre}$ is $\frac{1}{2^n}$. It would hypothetically be possible, given knowledge of x^* in advance, to perform a sequence of subdivisions which eventually yields a box $\mathbf{X}$ with $x^* \in \mathbf{X}_{centre}$, but it seems difficult to guarantee satisfaction of this condition in practice, without artificially expanding the size of the final box by a volume factor of 2^n.

Nevertheless, if (8.4) holds, then the (problematic) case that $\mathcal{G}$ vanishes (i.e. all g_i simultaneously attain zero) on the boundary of $\mathbf{X}$ is avoided, and condition (8.3) is fulfilled. In this case not only will the Miranda test suceed with the improved permutation checking, but it will succeed for the identity permutation, removing the need for any further permutation checking at all. However the overall complexity is not necessarily reduced, since we replace the permutation checking with a matrix inversion.

9 Improved Bernstein Expansion

At a typical point during the execution of a branch-and-bound method to solve a system of equations and/or inequalities (e.g. a global optimisation problem) involving polynomial functions, we have a polynomial

$$p(x) \ = \ \sum_{i=0}^{l} a_i x^i, \quad x = (x_1, \ldots, x_n), \tag{9.1}$$

in n variables, $x_1, \ldots, x_n$, of degree $l = (l_1, \ldots, l_n)$, and a box

$$\mathbf{X} \ := \ [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]. \tag{9.2}$$

In this chapter we address the question of how to determine a tight outer approximation for $p(\mathbf{X})$, the range of p over $\mathbf{X}$, in a timely fashion. Such bounds can be determined by utilising the coefficients of the expansion of the given polynomial into Bernstein polynomials, cf. Chapter 3. As we have seen (cf. Subsection 3.2.4), the coefficients of the Bernstein expansion of a given polynomial over a specified box of interest tightly bound the range of the polynomial over the box; in Chapter 8 this property was exploited in a solver for systems of polynomial equations.

The traditional approach (see, for example, [Gar86], [ZG98], Chapter 8) assumes that *all* of the Bernstein coefficients are computed, in order to determine their minimum and maximum. By use of the algorithm in Subsection 3.3.1, this computation can be made somewhat more efficient, with time complexity $\mathrm{O}(n\hat{l}^{n+1})$ and space complexity (equal to the number of Bernstein coefficients) $\mathrm{O}((\hat{l}+1)^n)$, where

$$\hat{l} \ = \ \max_{j=1}^{n} l_j. \tag{9.3}$$

Nevertheless, this usual approach still requires that all the Bernstein coefficients are computed, and their number is often very large for polynomials with moderately-many variables.

This exponential complexity is a major drawback of the existing method, rendering it infeasible for polynomials with moderately many (typically, 10 or more) variables. The main motivation of the work in this chapter is therefore to exploit the range enclosing property of the Bernstein expansion without recourse to the exhaustive computation of all the Bernstein coefficients.

An improved representation and computation scheme is developed [Smi09]. The faster performance of the new method is demonstrated with some numerical examples. The software which was developed is introduced in Appendix A.

9.1 Overview

A new method for the representation and computation of Bernstein coefficients of multivariate polynomials has been derived. This new technique represents the coefficients implicitly and uses lazy evaluation so as to render the approach practical for many types of non-trivial sparse polynomials typically encountered in global optimisation problems and systems of polynomial equations; the computational complexity becomes nearly linear with respect to the number of terms t in the polynomial, instead of exponential with respect to the number of variables n.

A brief overview of this approach is as follows:

- It is firstly necessary to derive some fundamental properties of the Bernstein coefficients of multivariate monomials. It is proven that such coefficients of a multivariate monomial can be computed as a simple product of the Bernstein coefficients of its component univariate monomials.
- It is proven that monotonicity of the Bernstein coefficients of monomials holds over a single orthant of $\mathbb{R}^n$.
- A new method of storing and representing the Bernstein coefficients of multivariate polynomials is proposed, which is called the *implicit Bernstein form.* Only the coefficients of the component univariate monomials are stored; the Bernstein coefficients of the whole polynomial itself are stored implicitly and are computed upon demand as a sum of t products. Computing and storing the whole set of Bernstein coefficients is not generally required for many types of sparse polynomial.

  The implicit Bernstein form consists of computing and storing sets of univariate Bernstein coefficients for each of the terms of the polynomial. Computing this form in general requires much less effort than for the explicit form.
- We consider the determination of the minimum or maximum Bernstein coefficient. The Bernstein enclosure (3.17) is given as the interval spanned by the minimum and maximum Bernstein coefficients; we are usually not interested in the intermediate coefficients.

  If the box $\mathbf{X}$ (9.2) spans multiple orthants of $\mathbb{R}^n$, then it should be subdivided around the origin into two or more sub-boxes, and the Bernstein enclosure for each sub-box computed separately. It should be noted that, for most branch-and-bound methods, the vast majority of the computational effort is typically occupied with small sub-boxes which lie within a single orthant.

  Finding the minimum Bernstein coefficient is essentially a problem of searching the array of the Bernstein coefficients. The search space can be dramatically reduced by employing three tests: uniqueness, monotonicity, and dominance.

9.2 Bernstein Coefficients of Monomials

We begin by deriving some fundamental properties of the Bernstein coefficients of multivariate monomials. Let us consider the case of a polynomial comprising a single term

$$q(x) \;=\; a_k x^k, \quad x \;=\; (x_1, \ldots, x_n), \quad \text{for some} \quad 0 \le k \le l, \tag{9.4}$$

where l is the degree of the Bernstein expansion to be computed. Let a box $\mathbf{X}$ (9.2) be given. Without loss of generality, we can take $a_k = 1$, since the Bernstein form is linear, cf. Subsection 3.2.3, i.e. here the Bernstein coefficients for general a_k may be obtained by multiplying these Bernstein coefficients by a_k. Using the compact notation for multipowers and generalised binomial coefficients, for $0 \le i \le l$ we have:

$$\begin{aligned} b_i &= \sum_{j=0}^{min\{i,k\}} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x} - \underline{x})^j \binom{k}{j} \underline{x}^{k-j} a_k \\ &= a_k \sum_{j=0}^{min\{i,k\}} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x} - \underline{x})^j \binom{k}{j} \underline{x}^{k-j}. \end{aligned}$$

Theorem 9.1. *Let*

$$q(x) = x^k, \quad x = (x_1, \ldots, x_n), \quad \text{for some} \quad 0 \le k \le l, \tag{9.5}$$

where $l = (l_1, \ldots, l_n)$. The Bernstein coefficients of q (of degree l) over a box $\mathbf{X}$ (9.2) are given by

$$b_i = \prod_{m=1}^{n} b_{i_m}^{(m)}, \tag{9.6}$$

where $b_{i_m}^{(m)}$ is the i_mth Bernstein coefficient (of degree l_m) of the (univariate) monomial x^{k_m} over the interval $[\underline{x}_m, \overline{x}_m]$.

Proof: The Bernstein coefficients of q (of degree l) over $\mathbf{X}$ are given by

$$\begin{aligned} b_i &= \sum_{j=0}^{min\{i,k\}} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x} - \underline{x})^j \binom{k}{j} \underline{x}^{k-j} \\ &= \sum_{j_1=0}^{min\{i_1,k_1\}} \cdots \sum_{j_n=0}^{min\{i_n,k_n\}} \left(\frac{\binom{i_1}{j_1} \cdots \binom{i_n}{j_n}}{\binom{l_1}{j_1} \cdots \binom{l_n}{j_n}} (\overline{x}_1 - \underline{x}_1)^{j_1} \cdots (\overline{x}_n - \underline{x}_n)^{j_n} \right. \\ &\qquad\qquad \left. \binom{k_1}{j_1} \cdots \binom{k_n}{j_n} \underline{x}_1^{k_1 - j_1} \cdots \underline{x}_n^{k_n - j_n} \right) \end{aligned}$$

$$= \sum_{j_1=0}^{min\{i_1,k_1\}} \frac{\binom{i_1}{j_1}}{\binom{l_1}{j_1}} (\overline{x}_1 - \underline{x}_1)^{j_1} \binom{k_1}{j_1} \underline{x}_1^{k_1-j_1} \cdots \sum_{j_n=0}^{min\{i_n,k_n\}} \frac{\binom{i_n}{j_n}}{\binom{l_n}{j_n}} (\overline{x}_n - \underline{x}_n)^{j_n} \binom{k_n}{j_n} \underline{x}_n^{k_n-j_n}$$

$$= \prod_{m=1}^{n} \sum_{j_m=0}^{min\{i_m,k_m\}} \frac{\binom{i_m}{j_m}}{\binom{l_m}{j_m}} (\overline{x}_m - \underline{x}_m)^{j_m} \binom{k_m}{j_m} \underline{x}_m^{k_m-j_m}$$

$$= \prod_{m=1}^{n} b_{i_m}^{(m)}. \ \square$$

Example 9.1. *Let $n := 2$, $q(x) := x_1^3 x_2^2$, $l := (3,2)$, and the box $\mathbf{X} := [1,2] \times [2,4]$. The Bernstein coefficients are*

$$\{b_i\} = \begin{pmatrix} 4 & 8 & 16 \\ 8 & 16 & 32 \\ 16 & 32 & 64 \\ 32 & 64 & 128 \end{pmatrix}. \tag{9.7}$$

Instead of calculating and storing all 12 Bernstein coefficients, we might instead represent them by

$$\begin{array}{cc} \mathbf{1} & \begin{pmatrix} 4 & 8 & 16 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \end{pmatrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \end{array}, \tag{9.8}$$

i.e. the coefficient $a_k = 1$, plus the Bernstein coefficients of x^3 over $[1,2]$, plus the Bernstein coefficients of x^2 over $[2,4]$. Any Bernstein coefficient of q over $\mathbf{X}$ can be computed as required, as a simple product of a_k and two of these respective coefficients.

9.2.1 Bernstein Coefficients of Univariate Monomials

In this section we consider the Bernstein coefficients $b_i^{(m)}$, $0 \le i \le l_m$, of degree l_m, of the univariate monomial x^{k_m} over the interval $[\underline{x}_m, \overline{x}_m]$, for some $1 \le m \le n$. For the remainder of the section, we omit the subscript m and the superscript (m), for simplicity.

The Bernstein coefficients are given by

$$b_i = \sum_{j=0}^{min\{i,k\}} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x} - \underline{x})^j \binom{k}{j} \underline{x}^{k-j}. \tag{9.9}$$

We wish to simplify this computation as much as possible.

Case $k = l$:

In this case, the formula can be simplified as follows:

$$\begin{aligned}
b_i &= \sum_{j=0}^{min\{i,k\}} \frac{\binom{i}{j}}{\binom{l}{j}} (\overline{x} - \underline{x})^j \binom{k}{j} \underline{x}^{k-j} \\
&= \sum_{j=0}^{i} \binom{i}{j} (\overline{x} - \underline{x})^j \underline{x}^{k-j} \\
&= \underline{x}^{k-i} \sum_{j=0}^{i} \binom{i}{j} (\overline{x} - \underline{x})^j \underline{x}^{i-j} \\
&= \underline{x}^{k-i} (\overline{x} - \underline{x} + \underline{x})^i \\
&= \underline{x}^{k-i} \overline{x}^i.
\end{aligned}$$

Case $k < l$:

In this case, we can start with the Bernstein coefficients of degree k, and use degree elevation. We have seen in Subsection 3.3.3 that the coefficients of higher degree can be expressed as a simple weighted sum of lower degree coefficients. Starting with the formula for $b_i^{[k+1]}$ (3.28), repeated degree elevation yields the following expression (cf. [FR88]) for the coefficients of degree l:

$$\begin{aligned}
b_i &= \sum_{j=\max\{0,i-m\}}^{\min\{k,i\}} \frac{\binom{m}{i-j}\binom{k}{j}}{\binom{k+m}{i}} b_j^{[k]} \\
&= \sum_{j=\max\{0,i-m\}}^{\min\{k,i\}} \frac{\binom{m}{i-j}\binom{k}{j}}{\binom{k+m}{i}} \underline{x}^{k-j} \overline{x}^j,
\end{aligned}$$

where $m = l - k$. This computation can be simplified (cf. [TP96]) by observing that the part of the formula consisting of the two binomial coefficients where m appears can always be expressed as a product of k factors.

9.2.2 Monotonicity of the Bernstein Coefficients of Monomials

We now derive an important property of the Bernstein coefficients for a monomial over an orthant-restricted box:

Theorem 9.2. *Let $q(x) = a_k x^k$, $x = (x_1, \ldots, x_n)$, for some $0 \leq k \leq l$ and let $\mathbf{X}$ be a box (9.2) which is restricted to a single orthant of $\mathbb{R}^n$. Then the Bernstein coefficients b_i of q (of degree l) over $\mathbf{X}$ are monotone with respect to each variable x_j, $j = 1, \ldots, n$.*

Proof: Given that the Bernstein coefficients of q over $\mathbf{X}$ can be expressed as a simple product of Bernstein coefficients of univariate monomials (cf. Theorem 9.1), it suffices to show monotonicity in the univariate case:

Lemma 9.1. *Let $q(x) = x^k$, for some $0 \leq k \leq l$, be a univariate monomial and let $\mathbf{x} = [\underline{x}, \overline{x}]$ be an interval where $\underline{x}$ and $\overline{x}$ do not have opposite signs (i.e. $\mathbf{x}$ does not contain both positive and negative values). Then the Bernstein coefficients b_i of q (of degree l) over $\mathbf{x}$ are monotone with respect to i.*

Proof: We will assume that $0 \leq \underline{x} < \overline{x}$; the negative case is entirely analogous. The result follows by induction on the degree of the Bernstein coefficients.

Case $l = k$:

From the case $k = l$ in the previous section we have

$$b_i^{[k]} = \underline{x}^{k-i}\overline{x}^i, \tag{9.10}$$

from which it is clear that $b_i^{[k]} \leq b_{i+1}^{[k]}$, $i = 0, \ldots, k-1$.

Case $l = k + m$, $m \geq 1$:

Assume that the Bernstein coefficients of degree $k + j$, $b_i^{[k+j]}$, $i = 0, \ldots, k+j$, $0 \leq j < m$, are increasing. The coefficients of degree $k + j + 1$, from (3.28), may be expressed as

$$b_i^{[k+j+1]} = \frac{ib_{i-1}^{[k+j]} + (k+j+1-i)b_i^{[k+j]}}{k+j+1}, \quad i = 0, \ldots, k+j+1, \tag{9.11}$$

with $b_{-1}^{[k+j]} = b_{k+j+1}^{[k+j]} = 0$. We observe that each $b_i^{[k+j+1]}$ is an affine combination of $b_{i-1}^{[k+j]}$ and $b_i^{[k+j]}$, from which

$$b_{i-1}^{[k+j]} \leq b_i^{[k+j+1]} \leq b_i^{[k+j]}, \quad i = 0, \ldots, k+j+1. \tag{9.12}$$

Therefore we have

$$b_i^{[k+j+1]} \leq b_{i+1}^{[k+j+1]}, \quad i = 0, \ldots, k+j, \tag{9.13}$$

and the result follows by induction. □

With this property, for a single-orthant box, the minimum and maximum Bernstein coefficients must occur at two vertices of the array of Bernstein coefficients. This also implies that the bounds provided by these coefficients are sharp (see the sharpness property in Subsection 3.2.5). Finding the minimum and maximum Bernstein coefficients is therefore straightforward; it is not necessary to explicitly compute the whole set of Bernstein coefficients. Computing the component univariate Bernstein coefficients for a multivariate monomial has time complexity $\mathrm{O}(n(\hat{l}+1)^2)$. Of course, one can readily calculate the exact range of a multivariate monomial over a single-orthant box without recourse to any Bernstein coefficients: Given the exponent k and the orthant in question, one can determine whether the monomial (and its Bernstein coefficients) is increasing or decreasing with respect to each coordinate direction. This allows one to determine in advance at which vertex of the box the minimum or maximum is attained; one then merely needs to evaluate the monomial at these two vertices.

Without the single orthant assumption, monotonicity does not necessarily hold, and the problem of determining the minimum and maximum Bernstein coefficients is more complicated. For boxes which intersect two or more orthants of $\mathbb{R}^n$, the box can be bisected, and the Bernstein coefficients of each single-orthant sub-box can be computed separately. The complexity of computing the minimum or maximum Bernstein coefficient will often still be much less than $\mathrm{O}((\hat{l}+1)^n)$. A bisection performed around zero will also yield an improvement of the bounds, unless they are already sharp [Sta95].

9.3 The Implicit Bernstein Form

In this section, a new method of storing and representing the Bernstein coefficients of multivariate polynomials is proposed, which we call the *implicit Bernstein form*.

Firstly, we can observe that since the Bernstein form is linear, cf. Subsection 3.2.3, if a polynomial p consists of t terms, as follows,

$$p(x) \;=\; \sum_{j=1}^{t} a_{i_j} x^{i_j}, \quad 0 \leq i_j \leq l, \quad x = (x_1, \ldots, x_n), \tag{9.14}$$

then each Bernstein coefficient is equal to the sum of the corresponding Bernstein coefficients of each term, as follows:

$$b_i \;=\; \sum_{j=1}^{t} b_i^{(j)}, \quad 0 \leq i \leq l, \tag{9.15}$$

where $b_i^{(j)}$ are the Bernstein coefficients of the jth term of p. (Hereafter, a superscript in brackets specifies a particular term of the polynomial. The use of this notation to indicate a particular coordinate direction, as in the previous section, is no longer required.)

Therefore one may implicitly store the Bernstein coefficients of each term, as in Section 9.2.2, and compute the Bernstein coefficients as a sum of t products, only as needed. Computing and storing the whole set of Bernstein coefficients, should, in general, not be required.

The implicit Bernstein form thus consists of computing and storing the n sets of univariate Bernstein coefficients (one set for each component univariate monomial) for each of t terms. Computing this form has time complexity $\mathrm{O}(nt(\hat{l}+1)^2)$ and space complexity $\mathrm{O}(nt(\hat{l}+1))$, as opposed to $\mathrm{O}((\hat{l}+1)^n)$ for the explicit form. Computing a single Bernstein coefficient from the implicit form requires $(n+1)t-1$ arithmetic operations.

Example 9.2. *We extend Example 9.1. Let* $n := 2$, $p(x) := x_1^3 x_2^2 - 30x_1x_2$, $l := (3,2)$, *and the box* $\mathbf{X} := [1,2] \times [2,4]$. *The sum of the corresponding Bernstein coefficients of each term gives the Bernstein coefficients of* p:

$$\begin{aligned}\{b_i\} &= \begin{pmatrix} 4 & 8 & 16 \\ 8 & 16 & 32 \\ 16 & 32 & 64 \\ 32 & 64 & 128 \end{pmatrix} + \begin{pmatrix} -60 & -90 & -120 \\ -80 & -120 & -160 \\ -100 & -150 & -200 \\ -120 & -180 & -240 \end{pmatrix} \\ &= \begin{pmatrix} -56 & -82 & -104 \\ -72 & -104 & -128 \\ -84 & -118 & -136 \\ -88 & -116 & -112 \end{pmatrix}.\end{aligned} \tag{9.16}$$

The implicit form of these coefficients can be represented as

$$\begin{array}{c}\mathbf{1}\\ \begin{pmatrix}1\\2\\4\\8\end{pmatrix}\end{array} \begin{array}{c}\begin{pmatrix}4 & 8 & 16\end{pmatrix}\\ \begin{pmatrix}\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\end{pmatrix}\end{array} + \begin{array}{c}\mathbf{-30}\\ \begin{pmatrix}1\\ \frac{4}{3}\\ \frac{5}{3}\\ 2\end{pmatrix}\end{array} \begin{array}{c}\begin{pmatrix}2 & 3 & 4\end{pmatrix}\\ \begin{pmatrix}\cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\\ \cdot&\cdot&\cdot\end{pmatrix}\end{array}. \tag{9.17}$$

A desired Bernstein coefficient can now be computed as a sum of two simple products. Note that here the minimum Bernstein coefficient, $b_{(2,2)} = -136$ *is not attained at a vertex of the array.*

9.3.1 Determination of the Bernstein Enclosure for Polynomials

In this section we consider the determination of the minimum Bernstein coefficient; the determination of the maximum Bernstein coefficient is analogous. The Bernstein enclosure (3.17) is the interval spanned by the minimum and maximum Bernstein coefficients.

If the box $\mathbf{X}$ (9.2) spans multiple orthants of $\mathbb{R}^n$, then it should be subdivided around the origin into two or more sub-boxes, and the Bernstein enclosure for each sub-box computed separately. The remainder of this section thus assumes that $\mathbf{X}$ is restricted to a single orthant. It should be noted that, for branch-and-bound methods for constrained global optimisation or systems of equations, the vast majority of the computational effort is typically occupied with small sub-boxes which lie within a single orthant.

Clearly, the determination of the minimum Berstein coefficient is not so simple as for a polynomial comprising a single term; the minimum is not guaranteed to occur at a vertex of the array, although this may often be the case. For polynomials in general, it is doubtful that a *universal* method more efficient than simply computing all of the Bernstein coefficients exists. However, when the number of terms of the polynomial is much less than the number of Bernstein coefficients (which is typically the case for many real-world problems), it is often possible in practice to dramatically reduce the number of coefficients which have to be computed, by reducing the number of Bernstein coefficients which have to be searched.

The minimum Bernstein coefficient is referenced by a multi-index, which we label $i_{\min}$, $0 \le i_{\min} \le l$. We wish to determine the value of the multi-index of the minimum Bernstein coefficient in each direction. In order to reduce the search space (among the $(\hat{l}+1)^n$ Bernstein coefficients) we can exploit Theorem 9.2 and employ the following tests:

- **Uniqueness**: If a variable x_j appears in only one term of the polynomial, then the Bernstein coefficients of the term in which it appears determines $i_{\min_j}$, which is thus either 0 or l_j.
- **Monotonicity**: If the Bernstein coefficients of all terms containing x_j are likewise monotone with respect to x_j, then $i_{\min_j} = 0$ (if all are increasing) or l_j (if decreasing).
- **Dominance**: Otherwise, all the terms containing x_j can be partitioned into two sets, depending on whether they are increasing or decreasing with respect to x_j. If the width of the Bernstein enclosure of one set (treated as the polynomial comprising its terms) is less than the minimum difference between Bernstein coefficients among the terms of the other set, then the first set can make no contribution to the determination of $i_{\min_j}$, and the monotonicity clause applies.

Theorem 9.3 (Location of minimum Bernstein coefficient under uniqueness / monotonicity)**.** *For a polynomial p given as per (9.14), the multi-index of the minimum Bernstein coefficient of p over a single-orthant box $\mathbf{X}$ (9.2), $i_{\min}$, must satisfy*

$$\min_{j=1,\ldots,t}\{i_{\min}^{(j)}\} \;\leq\; i_{\min} \;\leq\; \max_{j=1,\ldots,t}\{i_{\min}^{(j)}\}. \tag{9.18}$$

Proof: Suppose there is some k, $k \in \{1,\ldots,n\}$, for which

$$\min_{j=1,\ldots,t}\{{i_{\min}}_k^{(j)}\} \;>\; i_{\min k}. \tag{9.19}$$

The case of $i_{\min k}$ exceeding the maximum of the ${i_{\min}}_k^{(j)}$ is entirely analogous. Assume $0 \leq \underline{x}_k < \overline{x}_k$; the negative case is analogous. Then there is no m, $m \in \{1,\ldots,t\}$, for which ${i_{\min}}_k^{(m)} = 0$ and therefore the $b_i^{(m)}$ are decreasing with respect to i_k for all $m \in \{1,\ldots,t\}$. Therefore the $b_i = \sum_{j=1}^t b_i^{(j)}$ are decreasing with respect to i_k and so $i_{\min k} = l_k$, which is a contradiction of the initial supposition, and so the result follows. □

Theorem 9.4 (Location of minimum Bernstein coefficient under dominance)**.** *Given a polynomial p as per (9.14) and a single-orthant box $\mathbf{X}$ (9.2), for some $j \in \{1,\ldots,n\}$, let p^{inc} be the polynomial comprising the sum of the terms of p which are increasing with respect to x_j, and let p^{dec} be the polynomial comprising the sum of the terms of p which are decreasing with respect to x_j, with Bernstein coefficients b_i^{inc} and b_i^{dec}, respectively, $0 \leq i \leq l$. If*

$$\forall i = 0,\ldots,l,\; i_j \neq l_j : b^{inc}_{i_1,\ldots,i_j+1,\ldots,i_l} - b^{inc}_{i_1,\ldots,i_j,\ldots,i_l} \;>\; b^{dec}_{i_1,\ldots,0,\ldots,i_l} - b^{dec}_{i_1,\ldots,l_j,\ldots,i_l} \tag{9.20}$$

then $i_{\min j} = 0$. If

$$\forall i = 0,\ldots,l,\; i_j \neq l_j : b^{dec}_{i_1,\ldots,i_j,\ldots,i_l} - b^{dec}_{i_1,\ldots,i_j+1,\ldots,i_l} \;>\; b^{inc}_{i_1,\ldots,l_j,\ldots,i_l} - b^{inc}_{i_1,\ldots,0,\ldots,i_l} \tag{9.21}$$

then $i_{\min j} = l_j$.

Proof: The proof is presented for the first result (9.20); the proof of the second (9.21) is entirely analogous. For all $i = 0, \dots, l, i_j \neq l_j$ we have

$$\begin{aligned} b_{i_1,\dots,i_j+1,\dots,i_l} &= b^{inc}_{i_1,\dots,i_j+1,\dots,i_l} + b^{dec}_{i_1,\dots,i_j+1,\dots,i_l} \\ &\geq b^{inc}_{i_1,\dots,i_j+1,\dots,i_l} + b^{dec}_{i_1,\dots,l_j,\dots,i_l} \\ &> b^{inc}_{i_1,\dots,i_j,\dots,i_l} + b^{dec}_{i_1,\dots,0,\dots,i_l} \\ &\geq b^{inc}_{i_1,\dots,i_j,\dots,i_l} + b^{dec}_{i_1,\dots,i_j,\dots,i_l} \\ &= b_{i_1,\dots,i_j,\dots,i_l}, \end{aligned}$$

namely that the b_i are increasing with respect to x_j, and the result follows. □

Example 9.3. *Consider the polynomial*

$$p(x) = 3x_1x_2^5 + 2x_1^4x_2 - 8x_1^2x_3^6x_4^2 - x_1x_4^8 + 3x_2^3x_5 - 10x_4^5x_5^5x_6^5 + 0.01x_5^2x_6^2 + 4x_5^3x_7^4 \quad (9.22)$$

over the box

$$\mathbf{X} = [1,2]^7. \quad (9.23)$$

The degree, l, is $(4,5,6,8,5,5,4)$ and the number of Bernstein coefficients is thus 340200 ($5 \times 6 \times 7 \times 9 \times 6 \times 6 \times 5$). We can make the following observations:

- *Uniqueness: x_3 appears only in term 3, which is decreasing with respect to it. Therefore $i_{\min 3} = 6$.*
- *Uniqueness: x_7 appears only in term 8, which is increasing with respect to it. Therefore $i_{\min 7} = 0$.*
- *Monotonicity: x_2 appears in terms 1 and 2, both of which are increasing with respect to it. Therefore $i_{\min 2} = 0$.*
- *Monotonicity: x_4 appears in terms 3, 4, and 6, all of which are decreasing with respect to it. Therefore $i_{\min 4} = 8$.*
- *Dominance: x_6 appears in terms 6 and 7, one of which is decreasing and one of which is increasing with respect to it. However, term 6 dominates term 7 to such an extent that term 7 plays no role in determining $i_{\min 6}$. Therefore $i_{\min 6} = 5$, since term 6 is decreasing with respect to x_6.*

Variable x_1 appears in terms 1, 2, 3, and 4, and x_5 appears in terms 6, 7, and 8. A determination of $i_{\min 1}$ and $i_{\min 5}$ thus seems to be non-trivial.

So far, we have determined that $i_{\min} = (?, 0, 6, 8, ?, 5, 0)$. The dimensionality of the search space has thus been reduced from 7 to 2. The number of Bernstein coefficients to compute is consequently reduced from 340200 to 30 (5×6), plus those needed for the implicit Bernstein form, 78 ($8 + 7 + 13 + 11 + 6 + 18 + 6 + 9$), i.e. 108 in total.

9.3.2 Algorithm for the Efficient Calculation of the Bernstein Enclosure of Polynomials

An algorithm for the determination of the minimum Bernstein coefficient is given here; the procedure for the determination of the maximum Bernstein coefficient is analogous.

We are given a polynomial p (9.14) consisting of t terms, whose degree is $l = (l_1, \ldots, l_n)$, and a box $\mathbf{X}$ (9.2), as before. We seek to find a multiindex $i_{\min}$ which references the minimum Bernstein coefficient $b_{i_{\min}}$.

1. If $\mathbf{X}$ is not restricted to a single orthant of $\mathbb{R}^n$ (i.e. there is one or more component intervals of $\mathbf{X}$, $[\underline{x}_m, \overline{x}_m]$, $1 \leq m \leq n$, which contain both positive and negative numbers), then subdivide $\mathbf{X}$ around 0, perform steps 2-5 below for each sub-box, and take the minimum of the minimum Bernstein coefficients for each sub-box.

2. Compute the implicit Bernstein form of p over $\mathbf{X}$, consisting of the Bernstein coefficients of the component univariate monomials of each term.

3. Initialise the search space for $i_{\min}$, S, as the set of all possible multi-indices $\{(0, \ldots, 0), \ldots, (l_1, \ldots, l_n)\}$. (We do not need to store each possible multi-index explicitly; we are only interested in the dimensions of S.)

4. For each variable x_j, $j = 1, \ldots, n$:

   a) Uniqueness test: Count the number of terms for which the Bernstein coefficients are non-constant with respect to x_j. If the number is one, then restrict S so that the jth index corresponds to the minimum Bernstein coefficient of the non-constant term; it is either 0 or l_j.

   b) If the uniqueness test fails, then proceed with the monotonicity test: Sort the terms into those which are increasing, decreasing, and constant with respect to x_j. If the set of increasing terms is empty, then restrict S so that the jth index is l_j. If the set of decreasing terms is empty, then restrict S so that the jth index is 0.

   c) If the uniqueness and monotonicity tests fail, then proceed with the dominance test: Using the two non-empty sets of increasing and decreasing terms from the monotonicity test, compute the width of the Bernstein enclosures of each set, and the minimum absolute difference between Bernstein coefficients of each set. If the minimum absolute difference of the decreasing terms is greater than the width of the increasing terms, then restrict S so that the jth index is l_j. If the minimum absolute difference of the increasing terms is greater than the width of the decreasing terms, then restrict S so that the jth index is 0.

5. Explicitly compute the Bernstein coefficients corresponding to the remaining multi-indices in S, from the implicit form, and determine their minimum.

9.4 Numerical Results

The implicit Bernstein form (cf. Section 9.3) and the algorithm presented above in Subsection 9.3.2 is tested here, and compared to the usual Bernstein form (cf. Chapter 3). For each test problem, consisting of a polynomial p and a starting box $\mathbf{X}$, the Bernstein enclosure is computed, using each method. The box $\mathbf{X}$ is then bisected in each variable direction in turn, providing a basic simulation of a branch-and-bound environment. One of the two resulting sub-boxes is selected at random; it is retained and the other is discarded. After each bisection, the Bernstein enclosure is recomputed over the new box. This process is iterated 100 times, so that the final sub-box is very small.

With normal floating-point arithmetic, inaccuracies may be introduced into the calculation of the Bernstein coefficients and the corresponding bounds, due to rounding errors. As a result, the bounds may not be guaranteed to enclose the range of the polynomial over the box. Therefore interval arithmetic (cf. Section 2.1) has been used; all Bernstein coefficients are computed and stored as intervals. The bounds provided are thus guaranteed, in line with other schemes to provide 'safe' bounds and bounding functions [BVH05, HK04, NS04].

The first test problem consists of the polynomial

$$p(x) \;=\; 3x_1^2x_2^3x_3^4 + 1x_1^3x_2x_3^4 - 5x_1x_2x_4^5 + 1x_3x_4x_5^3 \tag{9.24}$$

over the box

$$\mathbf{X} \;=\; [1,2] \times [2,3] \times [4,6] \times [-5,-2] \times [2,10]. \tag{9.25}$$

The second test problem is the one given in Example 9.3. The remaining test problems are polynomial objective functions drawn from GLOBALLib library [GLO] of test problems for global optimisation. Where otherwise unspecified, a suitable single-orthant starting box of unit width was chosen.

The results are given in Table 9.1; n is the number of variables, t the number of terms, and l the degree, in each case. The number of Bernstein coefficients refers to the number that have to be computed explicitly; for **test2** (Example 9.3), for example, 30 Bernstein coefficients are required to determine the minimum, and 30 to determine the maximum. The timings given in the table are the mean computation times for a single iteration; these numbers should be multiplied by 100 to get the computation times for all iterations. The results were produced with C++ on a single-core 2.4 GHz PC running Linux; the *BeBP* software package described in Section A.2 and the interval library *filib++* [LTWvG01] were employed. The algorithm may alternatively be executed with floating-point arithmetic in place of interval arithmetic; this speeds up the process (by approximately one order of magnitude), but the resultant bounds are no longer guaranteed.

It is clear that the use of the implicit Bernstein form can dramatically reduce the number of Bernstein coefficients that need to be computed explicitly, thereby speeding up the computation of the Bernstein enclosure by up to several orders of magnitude.

It should be noted that almost all of the polynomials in [GLO] are sparse and of low degree, with few or no terms involving more than a single variable. This seems to be typical of the types of polynomials encountered in global optimisation problems. In such cases, the

			Bernstein Form			**Implicit Bernstein Form**		
Name	n	t	Iterations	No. of BCs	time (s)	Iterations	No. of BCs	time (s)
test1	5	4	1-100	1920	0.01	1-3	60	0.0001
$l = (3, 3, 4, 5, 3)$						4-5	12	
						6-100	2	
test2	7	8	1-100	340200	6.05	1-9	60	0.0004
$l = (4, 5, 6, 8, 5, 5, 4)$						10-100	12	
mhw4d	5	17	1-100	1500	0.04	1-3	1000	0.0068
$l = (2, 3, 4, 4, 4)$						4-100	200	
meanvar	7	49	1-100	2187	0.24	1-100	2	0.0008
$l = (2, 2, 2, 2, 2, 2, 2)$								
ex2_1_5	10	16	1-100	17496	0.83	1-100	2	0.0003
$l = (2, 2, 2, 2, 2, 2, 2, 1, 1, 1)$								
harker	20	40	1-100	1.96×10^{11}	$> 10^5$	1-100	2	0.0019
$l = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2)$								

Table 9.1: Number of Bernstein coefficients calculated and computation time for some example multivariate polynomials.

uniqueness, monotonicity, and dominance tests are much more likely to succeed, compared to a polynomial where each variable appears in many terms.

10 Bounding Functions for Polynomials

We return in this chapter to the question of determining a guaranteed outer approximation for a multivariate polynomial over a given box, but now we also aim to preserve the local shape of the polynomial within the box to a certain extent — we wish to compute bounding *functions*, rather than just a pair of bounding values defining an interval enclosure, as was the case in Chapters 3, 8, and 9.

As before, we have a polynomial

$$p(x) \;=\; \sum_{i=0}^{l} a_i x^i, \quad x = (x_1, \ldots, x_n), \tag{10.1}$$

in n variables, $x_1, \ldots, x_n$, of degree $l = (l_1, \ldots, l_n)$, and a box

$$\mathbf{X} \;:=\; [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_n, \overline{x}_n]. \tag{10.2}$$

Now, rather than tight constant bounds for $p(\mathbf{X})$, we wish to compute either a lower bounding function $\underline{f}_{\mathbf{X}}$ or an upper bounding function $\overline{f}_{\mathbf{X}}$, or both, with the property that

$$\underline{f}_{\mathbf{X}}(x) \;\leq\; p(x) \;\leq\; \overline{f}_{\mathbf{X}}(x), \quad \forall x \in \mathbf{X}. \tag{10.3}$$

Of course, it is relatively easy merely to satisfy these conditions, but we wish in general to minimise $\overline{f}_{\mathbf{X}}(x) - p(x)$ and $p(x) - \underline{f}_{\mathbf{X}}$ for $x \in \mathbf{X}$.

A principal application of such bounding functions is in the generation of relaxations for global optimisation problems within a branch-and-bound framework (cf. Section 6.2). Here, it is important for a good lower bounding function to satisfy three basic properties, apart of course from the definitional requirement (10.3):

- It must approximate the original function as tightly as possible.
- It should preserve (in a basic way) the broad shape of the function over the box.
- It should be simpler to evaluate than the original function, and possess 'easier' numerical properties, such as linearity, monotonicity, or convexity.

In this chapter we consider bounding functions for polynomials which may be derived from the Bernstein expansion (cf. Chapter 3) of the polynomial, i.e. from its Bernstein coefficients. Constant bounds provided by the Bernstein enclosure were utilised in Chapter 8, but they only satisfy the last of the three above points; they do not preserve any shape and they do not, in general, approximate a polynomial as closely as other types of bounding functions, as we shall see. We shall firstly consider convex–concave extensions and then affine bounding functions.

10.1 Convex–Concave Extensions

Convex–concave extensions, introduced in [Jan00], are pairs of functions which generalise the concept of an interval extension (cf. Definition 2.11).

Definition 10.1 (Convex–Concave Extension). *A* convex–concave extension *of a function $f : S \to \mathbb{R}$, where $S \subseteq \mathbb{R}^n$, is a mapping $[\underline{f}, \overline{f}]$ which provides for each nonempty box $\mathbf{X} \subseteq S$ a convex function $\underline{f}_{\mathbf{X}} : \mathbf{X} \to \mathbb{R}$ (a lower bounding function) and a concave function $\overline{f}_{\mathbf{X}} : \mathbf{X} \to \mathbb{R}$ (an upper bounding function) such that*

$$\underline{f}_{\mathbf{X}}(x) \;\leq\; f(x) \;\leq\; \overline{f}_{\mathbf{X}}(x), \quad \forall x \in \mathbf{X}. \tag{10.4}$$

Such bound functions are sometimes called *convex underestimators* and *concave overestimators*, respectively.

Here we define three different convex–concave extensions for univariate polynomials, based on the coefficients of the Bernstein expansion, and consider their inclusion isotonicity [GJS03a]. We will later consider the extension to the multivariate case. All three are based on the control points defined by the Bernstein coefficients (cf. Subsection 3.1.2). The extensions exhibit an increasing order of complexity; the first lower bounding function and upper bounding function are affine functions, the second extension comprises two affine functions for each bounding function, and the third convex–concave extension is derived from the convex hull of all the control points.

We are given a univariate (i.e. $n = 1$) polynomial p (10.1) and an interval $\mathbf{x} = [\underline{x}, \overline{x}]$. Let $b_0, \ldots, b_l$ be the Bernstein coefficients (3.13) of p over $\mathbf{x}$ and let their minimum be referenced by $i_{\min}$, i.e. $b_{i_{\min}} = \min\limits_{i=0,\ldots,l} b_i$.

Let $\mathbf{b}_0, \ldots, \mathbf{b}_l$ be the control points derived from the Bernstein coefficients, i.e.

$$\mathbf{b}_i \;=\; \left(\frac{(l-i)\underline{x} + i\overline{x}}{l}, b_i \right), \quad i = 0, \ldots, l. \tag{10.5}$$

The first two convex–concave extensions rely on a calculation of the slope between control points. The slope between $\mathbf{b}_{i_{\min}}$ and $\mathbf{b}_i$ is given by

$$d_{(i_{\min}, i)} \;=\; \frac{l}{\overline{x} - \underline{x}} \frac{b_i - b_{i_{\min}}}{i - i_{\min}}, \quad i = 0, \ldots, i_{\min} - 1, i_{\min} + 1, \ldots, l. \tag{10.6}$$

10.1.1 Extension 1 (One Affine Function)

Let b_k be a Bernstein coefficient such that the slope between its control point $\mathbf{b}_k$ and $\mathbf{b}_{i_{\min}}$ has minimal absolute value, i.e.

$$\left| \frac{b_k - b_{i_{\min}}}{k - i_{\min}} \right| \;=\; \min_{i=0,\ldots,i_{\min}-1, i_{\min}+1,\ldots,l} \left| \frac{b_i - b_{i_{\min}}}{i - i_{\min}} \right| \tag{10.7}$$

and let

$$d_{i_{\min}} \;=\; \frac{l}{\overline{x} - \underline{x}} \frac{b_k - b_{i_{\min}}}{k - i_{\min}}. \tag{10.8}$$

Then for all $x \in \mathbf{x}$, define an affine lower bounding function $\underline{f}$ as

$$\underline{f}(x) \;=\; b_{i_{\min}} + d_{i_{\min}} \left(x - \frac{(l - i_{\min})\underline{x} + i_{\min}\overline{x}}{l} \right). \tag{10.9}$$

An upper bounding function $\overline{f}$ can be defined in a similar fashion, giving (by construction)

$$\underline{f}(x) \;\leq\; p(x) \;\leq\; \overline{f}(x), \;\text{ for all }\; x \in \mathbf{x}. \tag{10.10}$$

In Subsection 10.2.3 we extend the affine lower bounding function $\underline{f}$ to multivariate polynomials, where its coefficients can now be determined as the optimal solution of a linear programming problem.

10.1.2 Extension 2 (Two Affine Functions)

Following a similar scheme to the above extension, we now wish to achieve a tighter enclosure by the use of two affine functions for each bounding function. The lower bounding function should thus have a negative slope to the left of the chosen minimum control point and a positive slope to the right of it. Let $b_{i_{\min}}$ be the minimum Bernstein coefficient, as above. Note that where $i_{\min} = 0$ or $i_{\min} = l$, we only have one affine function, as above. Otherwise, define two slopes $d^-_{i_{\min}}$ and $d^+_{i_{\min}}$ by

$$d^-_{i_{\min}} \;=\; \frac{l}{\overline{x} - \underline{x}} \max_{i=0,\ldots,i_{\min}-1} \frac{b_i - b_{i_{\min}}}{i - i_{\min}}, \quad d^+_{i_{\min}} \;=\; \frac{l}{\overline{x} - \underline{x}} \min_{i=i_{\min}+1,\ldots,l} \frac{b_i - b_{i_{\min}}}{i - i_{\min}}. \tag{10.11}$$

Then for all $x \in \mathbf{x}$, define a lower bounding function comprised of two affine functions, $\underline{f}$ as

$$\underline{f}(x) \;=\; \begin{cases} b_{i_{\min}} + d^-_{i_{\min}} \left(x - \frac{(l-i_{\min})\underline{x}+i_{\min}\overline{x}}{l} \right), & \text{if } x \leq \frac{(l-i_{\min})\underline{x}+i_{\min}\overline{x}}{l}, \\ b_{i_{\min}} + d^+_{i_{\min}} \left(x - \frac{(l-i_{\min})\underline{x}+i_{\min}\overline{x}}{l} \right), & \text{if } x > \frac{(l-i_{\min})\underline{x}+i_{\min}\overline{x}}{l}. \end{cases} \tag{10.12}$$

An upper bounding function $\overline{f}$ can be defined analogously, such that (10.10) is satisfied.

10.1.3 Extension CH (Convex Hull)

Convex–concave extensions 1 and 2 have a natural extension to the use of (up to) l upper and lower affine functions, which are provided by the convex hull of the control points associated with the Bernstein coefficients. Starting from the left-most Bernstein coefficient $b_0 = \underline{f}(\underline{x}) = \overline{f}(\underline{x})$ and its associated control point, we construct the lower bound function $\underline{f}$ by using the facets of the convex hull lying below (or on) the straight line connecting $\mathbf{b}_0$ with the control point $\mathbf{b}_l$ associated with the right-most Bernstein coefficient $b_l = \underline{f}(\overline{x}) = \overline{f}(\overline{x})$. Analogously, the upper bounding function $\overline{f}$ is determined by the facets of the convex hull lying above (or on) this straight line. An algorithm for the computation of the convex hull is relatively straightforward in the one-dimensional case, but highly non-trivial in higher dimensions. An example of the convex hull extension is presented in the following section.

10.1.4 Inclusion Isotonicity

Following Definition 2.12, a convex–concave extension $[\underline{f}, \overline{f}]$ is *inclusion isotone* if, for all intervals $\mathbf{x}$ and $\mathbf{y}$ with $\mathbf{x} \subseteq \mathbf{y}$,

$$\underline{f}_{\mathbf{y}}(x) \ \leq \ \underline{f}_{\mathbf{x}}(x) \ \leq \ \overline{f}_{\mathbf{x}}(x) \ \leq \ \overline{f}_{\mathbf{y}}(x), \quad \forall x \in \mathbf{x}. \tag{10.13}$$

In Theorem 3.5, we have seen that the convex hull of the Bernstein control points is inclusion isotone. The following counterexample shows, however, that the convex–concave extensions based upon either one or two affine upper and lower bounding functions are, in general, not inclusion isotone.

Example 10.1 ([GJS03a])**.** *Let $p(x) = 32x^4 - 112x^3 + 118x^2 - 47x + 6$. Table 10.1 gives the convex–concave extensions (of each type) over the intervals $\mathbf{x} = [0, 0.5]$, $[0, 0.6]$, $[0, 0.7]$, and $[0, 1]$, specified by their upper and lower vertices (rounded to 3 decimal places). For example, the first entry for Extension 1 states that the affine upper bounding function $\overline{f}$ has the values $\overline{f}(0) = 6$, $\overline{f}(0.5) = 0$, and the affine lower bounding function $\underline{f}$ satisfies $\underline{f}(0) = -1.667$, $\underline{f}(0.5) = 0$. The corresponding convex–concave extension is not inclusion isotone, since $\underline{f}_{[0,0.5]}(0) = -1.667 < \underline{f}_{[0,0.6]}(0) = -1.08$. A similar situation occurs on the same intervals for Extension 2. The three extension types for all intervals are depicted in Figure 10.1.*

$\mathbf{x}$	**Extension 1**	**Extension 2**
$[0, 0.5]$	(0,6) (0.5,0) (0,-1.667) (0.5,0)	(0,6) (0.5,0) (0,1.083) (0.25,-0.833) (0.5,0)
$[0, 0.6]$	(0,6) (0.6,0.235) (0,-1.08) (0.6,-0.96)	(0,6) (0.6,0.235) (0,6) (0.15,-1.05) (0.6,-0.96)
$[0, 0.7]$	(0,6) (0.7,0.187) (0,-3.029) (0.7,0.187)	(0,6) (0.7,0.187) (0,6) (0.175,-2.225) (0.7,0.187)
$[0, 1]$	(0,6) (1,0.333) (0,-6.667) (1,-3)	(0,6) (1,0.333) (0,6) (0.25,-5.75) (1,-3)
	Extension CH	
$[0, 0.5]$	(0,6) (0.5,0) (0,6) (0.125,0.125) (0.25,-0.833) (0.5,0)	
$[0, 0.6]$	(0,6) (0.6,0.235) (0,6) (0.15,-1.05) (0.3,-1.02) (0.6,0.235)	
$[0, 0.7]$	(0,6) (0.7,0.187) (0,6) (0.175,-2.225) (0.7,0.187)	
$[0, 1]$	(0,6) (0.75,1.75) (1,-3) (0,6) (0.25,-5.75) (1,-3)	

Table 10.1: Vertices of the convex–concave extensions for Example 10.1.

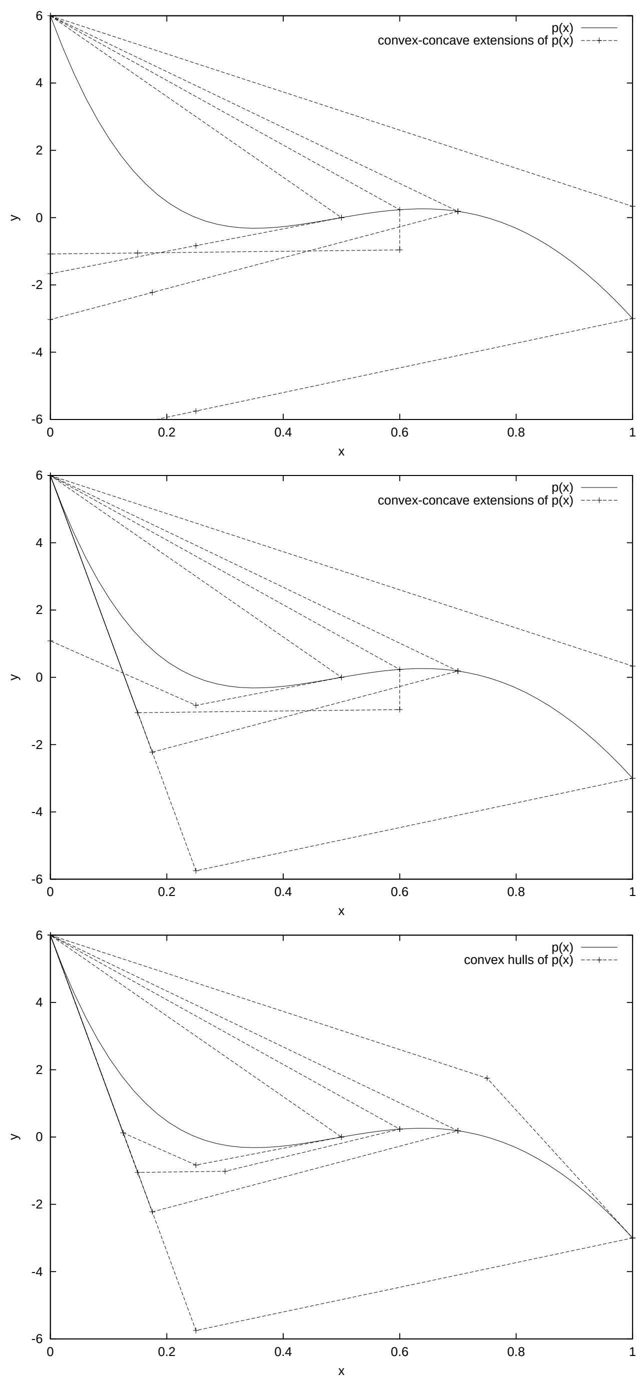


Figure 10.1: Extension 1 (top), Extension 2 (centre), and Extension CH (bottom) over the intervals $\mathbf{x} = [0, 0.5]$, $[0, 0.6]$, $[0, 0.7]$, and $[0, 1]$ for Example 10.1.

10.2 Affine Bounding Functions

We now restrict our attention to affine lower bounding functions. We begin with the simple convex–concave extension from Subsection 10.1.1, Extension 1, and firstly extend it to the multivariate case. Thereafter we consider a sequence of potentially improved methods for the computation of such bounding functions, and compare them.

Suppose, as usual, that we have an n-variate polynomial p (10.1) of degree $l = (l_1, \ldots, l_n)$ and a box $\mathbf{X}$ (10.2). We wish to find an affine lower bounding function

$$c(x_1, \ldots, x_n) \;=\; a_0 + a_1 x_1 + \ldots + a_n x_n \tag{10.14}$$

such that

$$c(x) \;\leq\; p(x), \;\; \forall x \in \mathbf{X}. \tag{10.15}$$

There are many possible such functions, but we wish to define a construction in such a way as to tightly approximate the polynomial from below, so that the maximum discrepancy between the values attained by the bounding function and the polynomial over the box is small. Given a method for a lower bounding function, an upper bounding function for p over $\mathbf{X}$ can usually be obtained in a completely analogous way; an upper bounding function c^u for p can be defined by $c^u(x) := -c(x)$, where c is a valid lower bounding function for $p^-(x) := -p(x)$.

During the course of this research, several possible approaches [GJS03b, GS04, GS07, GS08] for deriving a tight affine lower bounding function from the Bernstein control points (coefficients) of a given polynomial have been developed. An initial comparison of the first two of these methods was undertaken in [GS05] and a more complete comparison is given in Subsection 10.2.8.

As noted at the beginning of this chapter, the likely main application of such bounding functions is in relaxations for global optimisation problems with a branch-and-bound approach (cf. Section 6.2). When affine bounding functions are used, local relaxed subproblems become linear programming problems, which satisfy the requirement of being easier to solve than the original local problem. Good bounding functions also need to be tight and shape-preserving.

10.2.1 Overview

We first present a brief summary of the six methods presented, before going into detail.

The simplest type of affine bounding function is a constant bound function obtained by choosing the minimum Bernstein coefficient (cf. Subsection 10.2.2). This is equivalent to merely using the Bernstein enclosure (e.g. the approach in Chapter 8), i.e. point bounds, rather than an affine underestimator.

Most of the other methods rely on a control point associated with the minimum Bernstein coefficient, $\mathbf{b}_{i_{\min}}$, and a determination of n other control points. These are generally chosen in such a way that the linear interpolant of these points coincides with one of the lower facets of the convex hull of the control points and therefore constitutes a valid lower bounding function for the given polynomial. Such a bounding function can be obtained as the solution of a linear programming problem (cf. Subsection 10.2.3, [GJS03b]). An alternative (cf. Subsection 10.2.4, [GS04]), which, as we shall see, requires less computational effort, is the construction of a sequence of hyperplanes passing through $\mathbf{b}_{i_{\min}}$ which approximate from below the lower part of the convex hull increasingly well. In this case, only the solution of a system of linear equations together with a sequence of back substitutions is needed.

We shall also consider a straightforward approach based upon the designation of control points corresponding to the $n+1$ smallest Bernstein coefficients (cf. Subsection 10.2.5), or to the minimum Bernstein coefficient $b_{i_{\min}}$ and n others for which the absolute value of the slope between them (considered as control points) is minimal (cf. Subsection 10.2.6).

The final approach considered consists of deriving an affine approximation to the entire set of control points (and thereby the graph of the polynomial) over the box (cf. Subsection 10.2.7, [GS07, GS08]). It requires the use of the linear least squares approximation of the set of control points, which yields an affine function closely approximating the graph of the polynomial over the whole of the box. It must be adjusted by a downward shift so that it passes under all the control points, yielding a valid lower bounding function.

These six methods are to be compared, both with respect to the tightness of the bounding functions and with respect to computational complexity (computation time).

10.2.2 Method C (Constant Bound Function)

The easiest type of affine lower bounding function is one equal to a constant; we can simply assign it the value of the minimum Bernstein coefficient, $b_{i_{\min}}$:

$$c(x) \;:=\; b_{i_{\min}} \;:=\; \min_{i=0,\ldots,l} \{b_i\}.$$

Then, according to the range enclosing property (3.17)

$$c(x) \;\leq\; p(x), \quad \forall x \in \mathbf{X}.$$

This method is used here principally as a control — any method which does not deliver a noticably tighter approximation than it is likely to be of no benefit.

10.2.3 Method LP (Linear Programming Problems)

The next method tested is an expansion of Extension 1 (cf. Subsection 10.1.1) into the multivariate case, devised by C. Jansson [GJS03b], which we summarise here. It computes a bounding function passing through $\mathbf{b}_{\mathbf{i}_{\min}}$ and relies upon the computation of slopes and the solution of a linear programming (LP) problem (a brief description of which was given in Chapter 1).

Theorem 10.1 (Jansson [GJS03b]). *Let $J \subseteq \{j \mid 0 \leq j \leq l,\ j \neq i_{\min}\}$ be a set of at least n multi-indices such that*

$$\frac{b_j - b_{i_{\min}}}{\|j/l - i_{\min}/l\|} \;\leq\; \frac{b_i - b_{i_{\min}}}{\|i/l - i_{\min}/l\|} \;\; \forall\ j \in J,\ \ 0 \leq i \leq l,\ \ i \neq i_{\min},\ \ i \notin J, \qquad (10.16)$$

where $\|\cdot\|$ *denotes the Euclidean norm. Then the LP problem*

$$\text{minimise} \quad \left(\sum_{j\in J}(j/l - i_{\min}/l)\right)^T \cdot s \quad \text{such that}$$

$$(i/l - i_{\min}/l)^T \cdot s \;\geq\; b_{i_{\min}} - b_i \quad \text{for} \quad 0 \leq i \leq l,\;\; i \neq i_{\min}$$

has the following properties:

1. *It has an optimal solution* $\hat{s}$.
2. *The affine function*

$$c(x) \;:=\; -\hat{s}^T \cdot x + \hat{s}^T \cdot (i_{\min}/l) + b_{i_{\min}}$$

is a lower bound function for p *over the unit box* $\mathbf{I}$.

The proof is given in [GJS03b]; we omit it for brevity. An error bound is also given therein. An affine transformation (3.23, inverse) is applied to c to obtain the lower bounding function over $\mathbf{X}$. The quantities in (10.16, left-hand side) are absolute values of slopes in the direction $j/l - i_{\min}/l$. Aside from $b_{i_{\min}}$, n control points are chosen (corresponding to the multi-indices in J) for which these quantities are minimal. The result is an affine lower bounding function comprising in a weighted form these n smallest slopes, which passes through a facet of the convex hull of the control points with a minimal weighted slope.

We illustrate the construction of these affine lower bounding functions with a couple of examples:

Example 10.2 ([GJS03b]). *For* $l = 3, 8, 13, 17$, *let* $p(x) = \sum\limits_{i=0}^{l} \frac{(-1)^{i+1}}{i+1} x^i$, $x \in [0,1]$. *The graphs of the four polynomials, their control points, and the computed affine lower bounding functions are given in Figure 10.2.*

Example 10.3 ([GJS03b]). *A model for a hydrogen combustion with excess fuel [MS87] is described by equations with two bivariate polynomials, given by*

$$\begin{aligned} p_1(x_1,x_2) &= \alpha_1 x_1^2 x_2 + \alpha_2 x_1^2 + \alpha_3 x_1 x_2 + \alpha_4 x_1 + \alpha_5 x_2, \\ p_2(x_1,x_2) &= \alpha_6 x_1^2 x_2 + \alpha_7 x_1 x_2^2 + \alpha_8 x_1 x_2 + \alpha_9 x_2^3 + \alpha_{10} x_2^2 + \alpha_{11} x_2 + \alpha_{12}, \end{aligned}$$

where

$$\begin{array}{ll} \alpha_1 = -1.697 \times 10^7 & \alpha_7 = 4.126 \times 10^7 \\ \alpha_2 = 2.177 \times 10^7 & \alpha_8 = -8.285 \times 10^6 \\ \alpha_3 = 0.5500 & \alpha_9 = 2.284 \times 10^7 \\ \alpha_4 = 0.4500 \times 10^7 & \alpha_{10} = 1.918 \times 10^7 \\ \alpha_5 = -1.0000 \times 10^7 & \alpha_{11} = 48.40 \\ \alpha_6 = 1.585 \times 10^{14} & \alpha_{12} = -27.73. \end{array}$$

The graphs of the two polynomials, their control points, and the computed affine lower bounding functions over the unit box $\mathbf{I}^2$ *are given in Figure 10.3.*

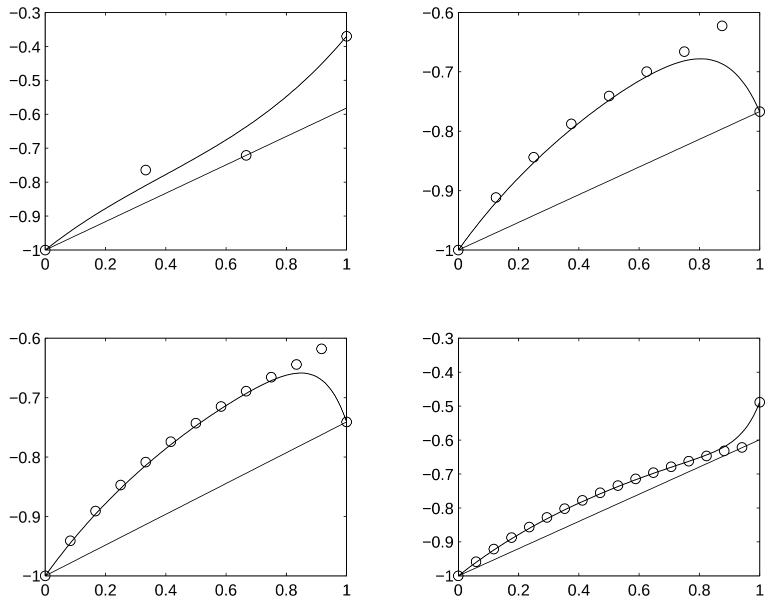


Figure 10.2: The four polynomials from Example 10.2, their control points, and computed affine lower bounding functions.

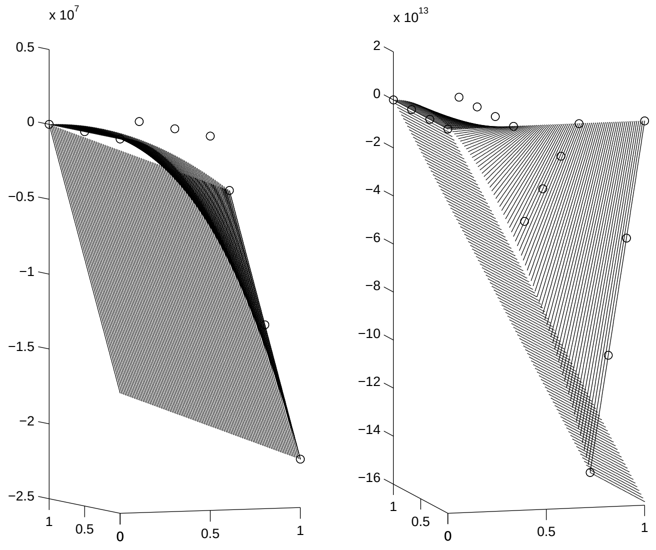


Figure 10.3: The two bivariate polynomials from Example 10.3, their control points, and computed affine lower bounding functions.

10.2.4 Method LE (Linear Equations)

With this method, we construct an affine lower bounding function by only solving a system of linear equations together with a sequence of back substitutions [GS04]. We aim to find hyperplanes passing through the chosen control point $\mathbf{b}^0 = \mathbf{b}_{\mathbf{i}_{\min}}$ (associated with the minimum Bernstein coefficient $b_{i_{\min}}$) which approximate from below the lower part of the convex hull of the control points increasingly well. In addition to $\mathbf{b}^0$, we will designate n additional control points $\mathbf{b}^1, \ldots, \mathbf{b}^n$. Starting with $c_0(x) := b_{i_{\min}}$, we will construct from these control points a sequence of affine lower bounding functions $c_1, \ldots, c_n$. We end up with c_n, a hyperplane which passes through a lower facet of the convex hull spanned by the control points $\mathbf{b}^0, \ldots, \mathbf{b}^n$. In the course of this construction, we will generate a set of linearly independent vectors $\{u^1, \ldots, u^n\}$ and we will compute slopes from $\mathbf{b}^0$ to $\mathbf{b}^j$ in direction u^j. Also, w^j will denote the vector connecting $\mathbf{b}^0$ and $\mathbf{b}^j$.

For ease of exposition, the bounding functions (hyperplanes) are specified in the case of the unit box $\mathbf{I}$. As above, an affine transformation (3.23, inverse) can be applied to c_n to obtain the lower bounding function over a general box $\mathbf{X}$.

Algorithm, First Iteration

$$\text{Let } u^1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Compute slopes g_i^1 from the control point $\mathbf{b}_i$ to $\mathbf{b}^0$ in direction u^1:

$$g_i^1 = \frac{b_i - b_{i_{\min}}}{\frac{i_1}{l_1} - \frac{i_1^0}{l_1}} \quad \text{for all } i \text{ with } i_1 \neq i_{\min_1}.$$

Let i^1 be a multi-index with smallest absolute value of associated slope g_i^1. Designate the control point $\mathbf{b}^1 = \left(\frac{i^1}{l}, b_{i^1}\right)$ and the vector $w^1 = \frac{i^1 - i^0}{l}$.

Define the lower bounding function

$$c_1(x) = b^0 + g_{i^1}^1 u^1 \cdot \left(x - \frac{i_{\min}}{l}\right).$$

jth Iteration, $j = 2, \ldots, n$

$$\text{Let } \tilde{u}^j = \begin{pmatrix} \beta_1^j \\ \vdots \\ \beta_{j-1}^j \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\text{such that } \tilde{u}^j \cdot w^k \;=\; 0, \;\; k = 1, \ldots, j-1. \tag{10.17}$$

Solving (10.17) for the coefficients $\beta_1^j, \ldots, \beta_{j-1}^j$ requires the solution of a system of $j-1$ linear equations in $j-1$ unknowns. This system has a unique solution due to the linear independence amongst the vectors $w^1, \ldots, w^n$, which is guaranteed by Theorem 10.2.

Normalise this vector thusly:

$$u^j \;=\; \frac{\tilde{u}^j}{\|\tilde{u}^j\|}. \tag{10.18}$$

Compute slopes g_i^j from the control point $\mathbf{b}_i$ to $\mathbf{b}^0$ in direction u^j:

$$g_i^j \;=\; \frac{b_i - c_{j-1}(\frac{i}{l})}{\frac{i - i_{\min}}{l} \cdot u^j} \quad \text{for all } i, \text{ except where } \frac{i - i_{\min}}{l} \cdot u^j = 0. \tag{10.19}$$

Let i^j be a multi-index with smallest absolute value of associated slope g_i^j. Designate the control point $\mathbf{b}^j = \left(\frac{i^j}{l}, b_{i^j}\right)$ and the vector $w^j = \frac{i^j - i_{\min}}{l}$.

Define the lower bounding function

$$c_j(x) \;=\; c_{j-1}(x) + g_{i^j}^j u^j \cdot \left(x - \frac{i_{\min}}{l}\right). \tag{10.20}$$

Theorem 10.2. *Let $w^{j,k}$ be the vectors in $\mathbb{R}^j$ given by taking the first j components of w^k, defined as above, for $k = 1, \ldots, j$. Then the vectors $w^{j,1}, \ldots, w^{j,j}$ are linearly independent, for $j = 1, \ldots, n$.*

Proof: We proceed by induction. Define vectors $u^{j,k}$ analogously by taking the first j components of u^k. The result holds for $j = 1$. Assume that $w^{j-1,1}, \ldots, w^{j-1,j-1}$ are linearly independent. By adding an extra component, we have that $w^{j,1}, \ldots, w^{j,j-1}$ are linearly independent. By (10.17), noting that only the first j components of u^j are nonzero, we have that $u^{j,j} \cdot w^{j,k} = u^j \cdot w^k = 0$, for $k = 1, \ldots, j-1$, i.e. $u^{j,j}$ is in the orthogonal complement of $w^{j,1}, \ldots, w^{j,j-1}$. Similarly by (10.19), we have that $u^{j,j} \cdot w^{j,j} = u^j \cdot w^j \neq 0$, i.e. $w^{j,j}$ is not orthogonal to $u^{j,j}$. Therefore $w^{j,j}$ is not in the subspace spanned by $w^{j,1}, \ldots, w^{j,j-1}$. □

For the n iterations of the algorithm, the solution of such a sequence of systems of linear equations would normally require $\frac{1}{6}n^4 + O(n^3)$ arithmetic operations. However we can take advantage of the fact that, in the jth iteration, the vectors $w^1, \ldots, w^{j-1}$ are unchanged from the previous iteration. The solution of these systems can then be formulated as Gaussian elimination applied rowwise to a single $(n-1) \times (n-1)$ matrix. In addition, a sequence of back-substitution steps has to be performed. Then altogether only $n^3 + O(n^2)$ arithmetic operations are required.

Let

$$L \;=\; \sqrt[n]{\prod_{i=1}^{n} (l_i + 1)}.$$

There are then L^n Bernstein coefficients, so that the computation of the slopes g_i^j in all iterations requires at most $n^2L^n + L^nO(n)$ arithmetic operations.

Theorem 10.3. *In the context of the above algorithm, and setting $i^0 := i_{\min}$, it holds for all $j = 0, \ldots, n$ that*

$$c_j\left(\frac{i^k}{l}\right) = b^k, \quad \text{for } k = 0, \ldots, j.$$

Proof: We proceed by induction. We already have that $c_0\left(\frac{i^0}{l}\right) = b^0$. Assume that

$$c_{j-1}\left(\frac{i^k}{l}\right) = b^k, \quad \text{for } k = 0, \ldots, j-1.$$

Then we have for $k = 0$ by (10.20) and the induction hypothesis that

$$\begin{aligned} c_j\left(\frac{i^0}{l}\right) &= c_{j-1}\left(\frac{i^0}{l}\right) + \alpha_j u^j \cdot \left(\frac{i^0}{l} - \frac{i^0}{l}\right) \\ &= b^0. \end{aligned}$$

If $k = 1, \ldots, j-1$, we can conclude using (10.17)

$$\begin{aligned} c_j\left(\frac{i^k}{l}\right) &= c_{j-1}\left(\frac{i^k}{l}\right) + \alpha_j u^j \cdot \left(\frac{i^k}{l} - \frac{i^0}{l}\right) \\ &= b^k + \alpha_j u^j \cdot w^k \\ &= b^k. \end{aligned}$$

Finally, if $k = j$, we apply (10.19) to obtain

$$\begin{aligned} c_j\left(\frac{i^j}{l}\right) &= c_{j-1}\left(\frac{i^j}{l}\right) + \frac{b^j - c_{j-1}(\frac{i^j}{l})}{\frac{i^j - i^0}{l} \cdot u^j} u^j \cdot \left(\frac{i^j}{l} - \frac{i^0}{l}\right) \\ &= c_{j-1}\left(\frac{i^j}{l}\right) + b^j - c_{j-1}\left(\frac{i^j}{l}\right) \\ &= b^j. \quad \square \end{aligned}$$

In particular, we have that

$$c_n\left(\frac{i^k}{l}\right) = b^k, \ k = 0, \ldots, n, \tag{10.21}$$

which means that c_n passes through all $n+1$ control points $\mathbf{b}^0, \ldots, \mathbf{b}^n$. Since c_n is by construction a lower bound function, $\mathbf{b}^0, \ldots, \mathbf{b}^n$ must therefore span a lower facet of the convex hull of all the control points.

We can readily obtain a pointwise error bound for the underestimating function c_n.

Theorem 10.4. *The affine lower bounding function c_n satisfies*

$$0 \;\leq\; p(x) - c_n(x) \;\leq\; \max_{0 \leq i \leq l} \left\{ b_i - c_n\left(\frac{i}{l}\right) \right\},\; \forall x \in \mathbf{I},$$

which specifies in the univariate case to

$$0 \;\leq\; p(x) - c_1(x) \;\leq\; \max_{0 \leq i \leq l, i \neq i^0} \left\{ \left(\frac{b_i - b^0}{i - i^0} - \frac{b^1 - b^0}{i^1 - i^0} \right) (i - i^0) \right\}.$$

The proof is the same as for the proof of Theorem 10.5 in Subsection 10.2.7.

10.2.5 Method MinBC (Minimum Bernstein Coefficients)

Perhaps an obvious approach for the definition of an affine lower bounding function based upon the Bernstein control points is to start by designating the control points corresponding to the $n + 1$ smallest Bernstein coefficients. With a bit of luck, these points uniquely define an affine function, the hyperplane interpolating them, which can be calculated in a straighforward fashion as the solution of a system of linear equations. Two issues need to be addressed before a lower bounding function is obtained. Firstly, the choice of control points may need to modified to avoid the degenerate cases either where they do not define a unique interpolating hyperplane (i.e. a singular matrix arises in the system of equations) or where the hyperplane has infinite gradient. An example of potential degeneracy is given in Figure 10.4; here, the three minimum control points are collinear and so the choice of defining control points must be altered. A similar requirement as in (10.19) can be enforced, to avoid such cases. Secondly, the interpolating hyperplane is not guaranteed to be a valid lower bounding function. In this case it must be corrected by the computation of an error term followed by a downward shift.

Suppose that we have determined the minimum Bernstein coefficient $b_{i_{\min}}$ and n smallest others, $b^1, \ldots, b^n$, defining a non-degenerate lower bounding function c^* as their interpolant. We must then compute the maximum positive discrepancy between c^* and the control points:

$$\delta^+ \;=\; \max_{0 \leq i \leq l} \left\{ c^*\left(\frac{i}{l}\right) - b_i \right\}. \tag{10.22}$$

Assuming δ^+ is non-zero, a valid bounding function is obtained by performing a downward shift:

$$c(x) \;=\; c^*(x) - \delta^+. \tag{10.23}$$

An example of this method is given in Figure 10.4; we can see that this scheme does not always yield a tight bounding function.

10.2.6 Method MinS (Minimum Slopes)

A variant of the above scheme is to designate the control point corresponding to the minimum Bernstein coefficient and n others which connect to it with minimum absolute value

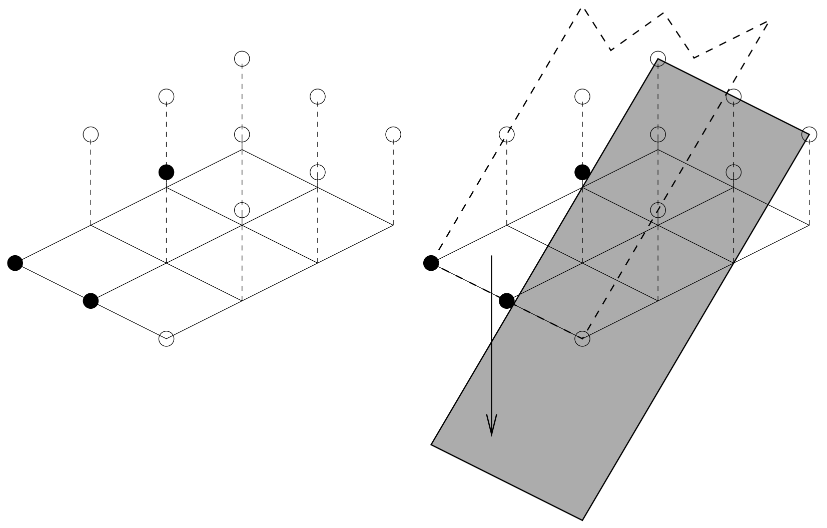


Figure 10.4: Method MinBC — an example of a poor lower bounding function; the designated control points are given in black and a downward shift is required.

of slope. As before a lower bounding function is obtained by interpolating the designated control points as the solution of a system of linear equations. Again, degenerate cases should be avoided and this bounding function may be invalid, in which case it is corrected by the computation of an error term followed by a downward shift, exactly as in (10.22) and (10.23).

An example of this method is given in Figure 10.5; again, we see that a tight bounding function is not always obtained.

10.2.7 Method LLS (Linear Least Squares Approximation)

In the final method, we endeavour to find an affine function that closely approximates *all* of the Bernstein control points and to further reduce the computational complexity [GS07, GS08].

Let A be the matrix with $\prod_{i=1}^{n}(l_i+1)$ rows and $n+1$ columns where the i,jth element is defined as

$$a_{i,j} \;=\; i_j/l_j, \quad \text{for } 1 \leq j \leq n,\; a_{i,n+1} = 1.$$

Let b be a vector consisting of the corresponding $\prod_{i=1}^{n}(l_i+1)$ Bernstein coefficients. Then the coefficients of the linear least squares approximation of all the control points are given by the solution γ to

$$A^T A\gamma \;=\; A^T b, \tag{10.24}$$

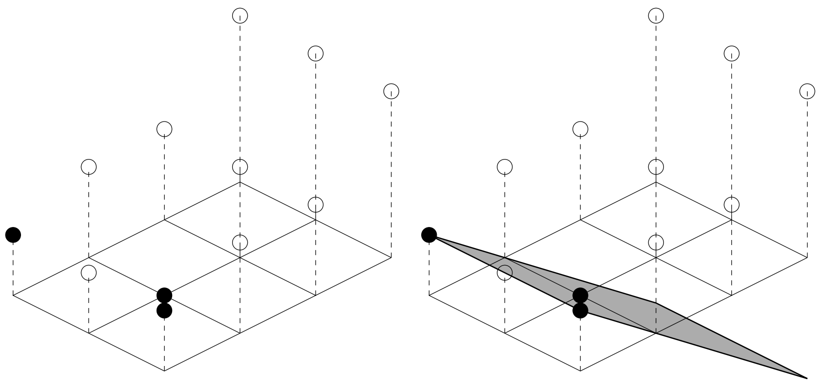


Figure 10.5: Method MinS — an example of a poor lower bounding function; the designated control points are given in black.

yielding the affine function

$$c^*(x) \;=\; \sum_{i=1}^{n} \gamma_i x_i + \gamma_{n+1}. \tag{10.25}$$

Numerically reliable approaches to solving the linear least squares problem are considered in [Bjö04]. As before (10.22) we compute the maximum positive discrepancy between c^* and the control points, δ^+, and perform a downward shift (10.23) to obtain a lower bounding function c. By construction, c is then a valid affine lower bounding function, and exhibits the same error bound as before:

Theorem 10.5. *The following error bound is valid:*

$$0 \;\le\; p(x) - c(x) \;\le\; \max_{0\le i\le l} \left\{ b_i - c_n\left(\frac{i}{l}\right)\right\},\ \forall x \in \mathbf{I}, \tag{10.26}$$

Proof: We already know (cf. Subsection 3.2.1) that the value of p at a vertex of the unit box I coincides with the respective Bernstein coefficient, i.e.

$$b_i \;=\; p\left(\frac{i}{l}\right),\ \text{ for all } \ i = 0,\ldots,l \ \text{ with } \ i_\mu \in \{0, l_\mu\},\ \mu = 1,\ldots,n.$$

We may consider dividing the surface of the convex hull of the control points into a lower and an upper part, in a natural way. The function describing the upper surface is piecewise affine over the uniform grid for the abscissae of the control points; let us denote it by u. The discrepancy $u - c$ is the difference of a piecewise affine and an affine function and must therefore assume its maximum at a point $\frac{i^*}{l}$ at which the associated control point is an exposed vertex of the convex hull, hence $u\left(\frac{i^*}{l}\right) = b_{i^*}$. So we may conclude that

$$\begin{array}{rcl} \max_{x\in\mathbf{I}}\left(p(x) - c(x)\right) & \le & \max_{x\in\mathbf{I}}\left(u(x) - c(x)\right) \\ & = & \max_{i=0,\ldots,l}\left(u\left(\frac{i}{l}\right) - c\left(\frac{i}{l}\right)\right) \\ & = & \max_{i=0,\ldots,l}\left(b_i - c\left(\frac{i}{l}\right)\right).\ \square \end{array}$$

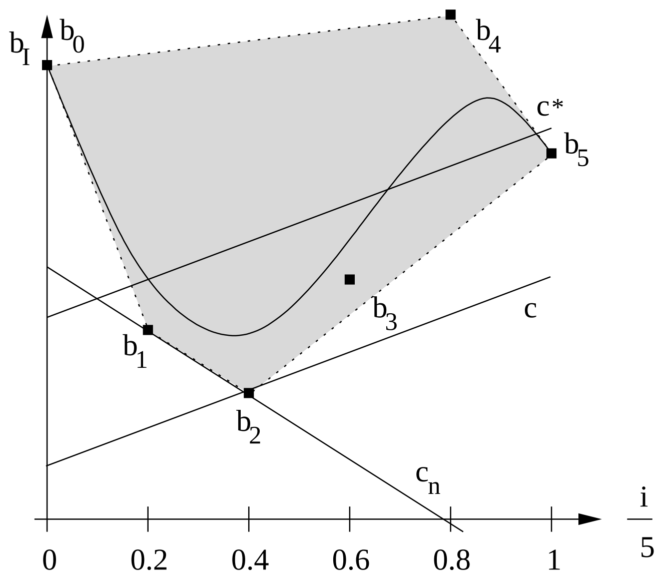


Figure 10.6: The curve of the polynomial from Figure 3.2, the convex hull (shaded) of its control points (marked by squares), the intermediate affine function c^* and the affine lower bounding function c from Method LLS versus the affine lower bounding function c_n from Method LE.

Again, the same affine transformation as before (3.23) can be applied in inverse, transforming c, a bounding function over $\mathbf{I}$, to an affine function which is then by construction a valid lower bounding function for p over $\mathbf{X}$.

Figure 10.6 illustrates the construction of an affine lower bounding function using this scheme for the same univariate polynomial of degree 5 as in Figure 3.2. It is compared to the bounding function obtained by Method LE.

10.2.8 Numerical Results

We run a series of tests to compare the performance of the described methods for affine lower bounding functions. We are interested in both the computational complexity (computation time) and the tightness of the functions (smallness of the error bound).

The methods were tested with polynomials in n variables with degree $l = (D, \ldots, D)$ and k non-zero terms over the unit box $\mathbf{I}$. The non-zero coefficients a_i were randomly generated with $a_i \in [-1, 1]$. In each case, the entire set of Bernstein coefficients was computed as a precursor to computing the bounding function.

Table 10.2 lists the results for different values of n, D, and k; $(D + 1)^n$ is the number of Bernstein coefficients. In each case, 100 random polynomials were generated and the mean computation time (including the time for the computation of the Bernstein coefficients) and discrepancy δ according to (10.26, right-hand side) are given. The results were produced with C++ on a single-core 2.4 GHz PC running Linux; the *BeBP* software package described in Section A.2, the interval library *filib++* [LTWvG01], and the *LP_SOLVE* [BD$^+$] software for solving linear programming problems were used.

Method				**Method C**		**Method LP**		**Method LE**	
n	D	k	$(D+1)^n$	time (s)	δ	time (s)	δ	time (s)	δ
2	2	5	9	0.000002	1.420	0.00020	0.976	0.000069	0.981
2	6	10	49	0.000011	2.002	0.0025	1.695	0.00031	1.677
2	10	20	121	0.000044	2.852	0.023	2.543	0.00074	2.511
4	2	20	81	0.000053	3.458	0.0082	2.847	0.0012	2.797
4	4	50	625	0.00055	5.682	2.82	5.056	0.0093	5.045
6	2	20	729	0.00056	4.075	4.48	3.403	0.016	3.353
8	2	50	6561	0.0090	6.941	>1 minute		0.24	6.291
10	2	50	59049	0.11	7.142	>1 minute		3.43	6.503
12	2	50	531441	1.32	7.377	>1 minute		>1 minute	
				Method MinBC		**Method MinS**		**Method LLS**	
2	2	5	9	0.000085	1.147	0.00011	0.961	0.000006	0.698
2	6	10	49	0.00031	4.914	0.00044	1.910	0.000024	1.496
2	10	20	121	0.00090	11.49	0.0012	3.014	0.000070	2.435
4	2	20	81	0.0012	4.797	0.0015	3.199	0.000090	2.468
4	4	50	625	0.0088	14.05	0.011	5.940	0.00079	4.870
6	2	20	729	0.015	5.921	0.017	3.687	0.0010	3.131
8	2	50	6561	0.21	14.33	0.24	7.360	0.018	6.300
10	2	50	59049	2.69	17.11	3.11	7.680	0.29	6.473
12	2	50	531441	>1 minute		>1 minute		3.81	6.712

Table 10.2: Numerical results (computation time and maximum discrepancies) for the six methods for affine lower bounding functions, applied to random polynomials over the unit box $\mathbf{I}$.

We can make the following observations:

- Being the simplest type of bounding function, Method C (a constant bound function) is of course the fastest. However constant bounding functions are crude and we would expect them to exhibit a mediocre error bound, which is what we see.
- The 'naive' choices of affine bounding function based on the smallest Bernstein coefficients (Method MinBC) or the smallest slopes (Method MinS) are unreliable, giving extremely poor bound functions in some cases. In several cases, the mean discrepancy is worse than a constant bound function. However, they are relatively fast.
- Method LP clearly demonstrates an improved error bound relative to constant bound functions, but is slow. The bulk of the computation time is occupied with the solution of the component LP problems, which seems unavoidable.

- Method LE delivers a slightly better average error bound than Method LP. Moreover, it is much faster — the solution of systems of linear equations is less computationally intensive than LP problems.
- Method LLS is faster than all except Method C, and it has the best average error bound.
- The best method for any given polynomial and box may vary, but is most often Method LLS.

10.2.9 An Equilibriation Transformation

Here we propose in outline a preprocessing step which may potentially improve the performance of some of the aforementioned methods for affine lower bounding functions. This is founded on the observation that methods based on minimum absolute values of slopes (e.g. Method LP, Method LE, Method MinS) may not perform well in situations where the polynomial exhibits a large gross change in value in one or more directions over the box. More specifically, this is the case where the range of values of one or more of its partial derivatives over the box does not contain zero and is far from zero, i.e. it is monotonically increasing in one or more directions with high Lipschitz constant. Such a polynomial in the univariate case is illustrated in Figure 10.7; the polynomial is increasing over the unit box. The obvious affine lower bounding function based on minimum slopes connects b_0 and b_1; however a bounding function connecting b_1 and b_6 more closely represents the gross shape of the polynomial and has a lower error bound.

We may thus apply an equilibriation affine transformation, in this case by adding the term $-\frac{b_6-b_0}{x}$ to the polynomial, so that the first and last Bernstein coefficients have the same value. Then, we compute an affine lower bounding function using minimum slopes, as before, before finally applying the inverse transformation to this bound function.

The procedure for computing such a transformation in the univariate case is obvious, but becomes non-trivial in the multivariate case. Where $n \geq 2$, it is not in general possible to equilibriate all 2^n vertex Bernstein coefficients to the same value with an affine transformation. Instead we might proceed, for example, by applying such a transformation in each direction in turn, so as to equilibriate the mean values of Bernstein coefficients on opposite pairs of faces of the box. Indeed such a transformation, if augmented by a suitable downward shift as in (10.23), might itself comprise a good choice of lower bounding function.

10.2.10 Verified Bounding Functions

It is often desirable to compute the affine lower bounding function in a rigorous fashion, i.e. in such a way that it can be *guaranteed* to stay below the given polynomial over the box, cf. [HK04, NS04, BVH05]. Otherwise, rounding errors may cause inaccuracies to be introduced into the calculation of the Bernstein coefficients and the corresponding bounding function. As a result, the computed affine function may not stay below the given polynomial everywhere over the box. We may also wish to treat problems with

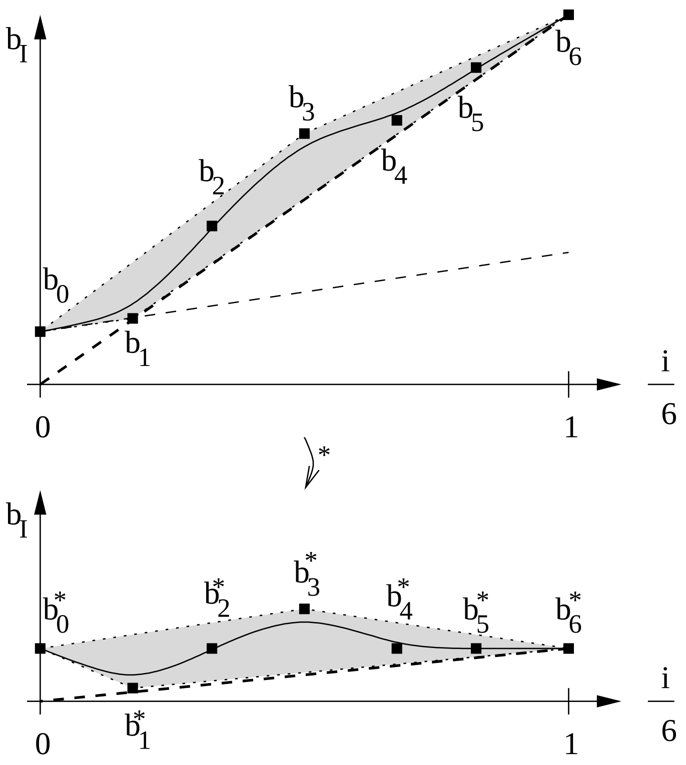


Figure 10.7: An equilibriation transformation applied to a degree 6 polynomial over the unit box and the corresponding change in choice of slope-based affine lower bounding function.

uncertain input data, such as a polynomial with interval coefficients, assuming that for each coefficient a lower and an upper bound can be determined. Such a polynomial might result either from real uncertainties in the problem, or otherwise very small intervals of machine precision width may be used to cater for rounding errors in the initial data input. The Bernstein coefficients b_i can be computed as before, using interval arithmetic. Each power-form coefficient a_i contributes only once to each Bernstein coefficient, so this can be done without any overestimation.

It is possible to interpolate interval control points in a safe fashion, e.g. [BVH05]. In the case of Method LLS (cf. Subsection 10.2.7), we may readily obtain a verified lower bounding function based upon interval Bernstein coefficients by proceeding as follows:

1. Given a polynomial with interval coefficients, compute its interval Bernstein coefficients as before, but with interval arithmetic.

2. Compute the linear least squares approximation of the control points as before, except using the *midpoints* of the interval Bernstein coefficients.

3. Compute the discrepancy δ^{+} and perform the downward shift as before, but according to the *lower bounds* of the control points (Bernstein coefficients).

Step 2 (the bulk of the computation) does not need to be performed rigorously, and is implemented with normal floating-point arithmetic. Since only the first and last steps need to be performed with interval arithmetic, the extra computational effort is relatively light.

11 Conclusions

We conclude this work with a review of the main results of the thesis and list a number of suggestions for future research.

11.1 Summary

We have considered methods applicable to systems of polynomial equations and problems involving polynomial inequalities, such as global optimisation problems. A branch-and-bound, or subdivision, framework is commonly employed. Here, at a particular iteration, we have a box or sub-box for which we wish to determine the existence or non-existence of solutions to a system of equations or an optimisation problem. As tests for systems of equations, we consider as a *boundary method* the topological degree, specifically the Brouwer degree, and as an *enclosure method* the Bernstein expansion and the related issue of bounding the ranges of component polynomials. For problems involving polynomial inequalities, we generate affine relaxations based upon the control points of the Bernstein expansion.

Comparison of Topological Degree and the Miranda Test

We may compare the use of the topological degree (cf. Chapter 7) against the Miranda test coupled with Bernstein expansion (cf. Chapter 8) as an existence test. Neither provide proofs of both existence and non-existence by themselves alone. The Miranda test is a sufficient but not a necessary condition for a solution to a system of equations; for a proof of non-existence, i.e. to exclude the possibility of solutions, it needs to be augmented with a proof of monotonicity (e.g. by computing Bernstein enclosures for the partial derivatives). The topological degree comprises an existence test only if the result is non-zero, due its root-counting property; in order to provide an existence or non-existence proof in the case of a zero outcome, suitable bounds on the Jacobian determinant are also needed. The Bernstein enclosure can also be used as a non-existence test by itself and provides enclosures which are generally tighter than those given by most interval enclosures.

In comparing the computational performance of the two alternatives, two observations become clear: Firstly, the computational cost of the Miranda test and the calculation of the Bernstein coefficients is generally predictable, whereas the computational cost of the recursive topological degree calculation is highly variable. Secondly, in non-trivial cases, the topological degree calculation is often more expensive. It itself is a type of branch-and-bound scheme; any overall solver where the component iterations themselves consist of branch-and-bound problems (i.e. doubly-nested subdivision overall) is likely to be prone to extreme slowness. Nevertheless, the topological degree calculation is *sometimes* very fast. It may be possible to additionally exploit the root-counting property of the topological degree (viz. it is equal to the sum of the signs of Jacobian determinants at roots), whereas the Miranda test only provides a binary result. Still, in the basic application, the root-counting property is superfluous, although it may be of greater interest where the topological degree is applied as a root designation or counting tool. In a branch-and-bound scheme where one makes a binary decision (e.g. to prune or to retain a sub-box, or to test candidate boxes for solutions), a binary result is precisely the level of information needed.

Applicability of the Topological Degree

The recursive algorithm given in outline by O. Aberth [Abe94] has been investigated in detail, particularly the central aspect of face subdivision. We have illustrated the impossibility of bounding the computational effort in the general case, but have performed an average-case analysis which strongly indicates (and proves, under a simplified model) that the face subdivision is characterised by an average branching factor which approaches $\frac{1}{2}$ from above as the box size tends to zero. We have explored the theory of subdivision here, introducing the concepts of crucial boxes, fatal boxes, and worthwhile subdivision; this theory may also be more generally applicable to other box subdivision schemes involving systems of equations. The merit of bisection is considered and a heuristic for choice of subdivision point is proposed. Some further specific conclusions for Chapter 7, concerning the details of the algorithm, are given in Section 7.6.

There may be applications for which it is desired to compute specifically the topological degree, rather than a solution to a system of equations, but the latter phenomenon seems far more common, so we shall focus on this context.

The question of how this algorithm, or any scheme for topological degree computation, may be integrated into the existing milieu of solvers for nonlinear systems of equations remains open. As yet, there is no scheme for the computation of the Brouwer degree over an arbitrary box for which the computational effort is bounded in the worst case. It thus seems hard to avoid the conclusion that the topological degree is best applied sparingly or judiciously, perhaps as a solution-counting pre-processing tool or a verification routine for very small boxes which complements rather comprises the 'main engine' of a branch-and-bound solver, which may be accelerated with other box contraction techniques.

Given the unpredictable computational effort, it may make sense to *start* a topological degree computation, perhaps even multiple instances in parallel with different choices of ray direction (cf. Subsection 4.4.3), in order to check whether one of them terminates quickly. If all instances are non-trivial, it may be more efficient to terminate them and perform a subdivision in the branch-and-bound algorithm, rather than in the topological degree algorithm.

Applicability of the Bernstein Expansion

The main computational advantages and disadvantages of the Bernstein expansion (cf. Chapter 3) are already known. Aside from the major application to Bézier curves, from the point of view of polynomial approximation, two beneficial attributes are the tightness of the Bernstein enclosure — the coefficients of the Bernstein expansion of a given polynomial over a specified box tightly bound the range of the polynomial over the box — and the relative numerical stability of computation with the Bernstein form compared to the power form. The principal disadvantage is that the traditional computation of the entire set of Bernstein coefficients (previously assumed to be necessary to compute the Bernstein enclosure) exhibits exponential complexity with respect to dimension (number of variables), rendering the approach infeasible for problems with many variables.

With the introduction of the implicit Bernstein form (cf. Chapter 9), this disadvantage has been largely eliminated for sparse polynomials and single-orthant boxes, which appear in many types of problem. For such problems, the Bernstein enclosure can typically be obtained without recourse to the computation of the entire set of Bernstein coefficients; the new technique represents the coefficients implicitly and uses lazy evaluation. Instead, the complexity becomes approximately linear with respect to the number of terms; this constitutes a dramatic speedup for many kinds of polynomials that typically appear in problems. Bernstein expansion as a range-enclosing method thus becomes feasible for many categories of polynomials with more variables.

The Bernstein expansion has also been applied to the solution of systems of polynomial equations (cf. Chapter 8). It can be seen that the tight range-enclosing property of the Bernstein expansion can be used to effectively prune boxes in a branch-and-bound scheme. At the end, a set of very small candidate boxes for solutions are obtained. The Miranda test is used to prove the existence of solutions; a computational speedup and a preconditioning scheme are proposed.

A complete solver for systems of polynomial equations is challenging, and would ideally incorporate multiple approaches. As well as the aforementioned schemes, narrowing operators, e.g. [VHMK97, Gra00], or domain reduction, e.g. [SP93, MP09], could be included. The implicit Bernstein form would doubtless improve the performance for many categories of polynomial.

The Bernstein expansion has also recently been employed to obtain tight enclosures for the ranges of rational functions [GSS12]. On the occasion of the 100th anniversary of its introduction by S. N. Bernstein [Ber12], there appears an extensive survey paper [Far12] as well as a forthcoming special issue of the journal *Reliable Computing* dedicated to Bernstein expansion [Ge12].

Affine Bounding Functions

In Chapter 10 we have presented a series of methods for the computation of guaranteed affine lower bounding functions for polynomials, based upon the Bernstein expansion and the use of interval arithmetic. Such bounding functions can be applied (as a black-box component) to the solution of constrained global optimisation problems within a branch-and-bound framework; if they are used to construct relaxations for a global optimisation problem, then sub-problems over boxes can be reduced to linear programming problems, which are easier to solve.

Due to the tightness of the Bernstein enclosure and the convex hull property (cf. Subsection 3.2.6), we can obtain bounding functions which are both tight and broadly shape-preserving. We have seen that the question of how to compute such functions in the multivariate case is highly non-trivial, and naive approaches often yield poor-quality functions. The investigated methods are based upon the computation of slopes and the solution of linear programming problems, systems of linear equations, or linear least squares approximations. Bounding functions for derivatives can also be computed; first- and second-order information is easily obtainable from the Bernstein coefficients.

The series of methods investigated currently rely upon the availability of the entire array of Bernstein coefficients. As noted above, the exponential complexity limits these approaches to polynomials in relatively few variables. Nevertheless, the approach is suited to many polynomials which appear in typical global optimisation problems; even if the number of variables in the problem is large, the number which appear in any given constraint function is typically much fewer. A revised or new method based on the more efficient implicit Bernstein form would similarly extend the applicability of the Bernstein expansion to a larger class of polynomials. It should ultimately be possible to exploit the implicit Bernstein form to provide a good-quality guaranteed affine bounding function for a high degree sparse polynomial in many variables over a box, in low or moderate time.

11.2 Future Work

Topological Degree

There are still several potential improvements to the recursive algorithm for the computation of Brouwer degree, described in Section 4.4 and investigated in Chapter 7.

- Further work may be possible with the complexity study of the face subdivision process undertaken in Section 7.3. Conjectures 7.1 and 7.2 require proof; an extension to arbitrary nonlinear functions, however, would seem very difficult. However such detailed study of the minutiae of the algorithm may be of limited practical interest.
- There is the potential to improve the subdivision procedure, i.e. to reduce the number of faces in total to be processed, with new heuristic(s) for the selection of the subdivision point as an alternative to bisection; a proposal is made in Subsection 7.5.8. It remains to implement and test such heuristic rules.
- Conceptually, the algorithm relies upon a ray projected from the origin intersecting the image of a box under the mapping in question (cf. Subsection 4.4.3). For a given example, it is often possible to reduce the number of face images intersected by altering the direction of this ray, which may be chosen freely. The key question, which remains to be investigated, is the way in which an advantageous ray direction might be determined.
- It ought to be possible to develop an experimental topological-degree based branch-and-bound global solver for systems of nonlinear equations. The topological degree

computation needs to be augmented either with some kind of monotonicity test or, more likely, bounds for the ranges of the Jacobian determinant, in order to provide complete non-existence and existence tests. It is suspected that the computational cost of the topological degree calculations is too high for *repetetive* application in a branch-and-bound environment, but this remains open. Such a solver may also provide a suitable environment for testing the topological degree against other root verification methods.

Solution of Systems of Polynomial Equations

- Up to now, the implicit Bernstein form has not been employed in the branch-and-bound algorithm for the solution of systems of polynomial equations, described in Chapter 8. Its use ought to offer a considerable speedup for many problems involving sparse polynomials. However, the preconditioning step, if retained, may have the property of generating preconditioned polynomials that are no longer sparse (i.e. increasing the number of non-zero terms in their power-form expression).

Improved Bernstein Expansion

- The current algorithm (cf. Subsection 9.3.2) for the computation of the Bernstein enclosure based upon the implicit Bernstein form requires that the box is restricted to a single orthant of $\mathbb{R}^n$; if this requirement is not satisfied, the box must firstly be subdivided. However an improved algorithm designed for boxes spanning multiple orthants of $\mathbb{R}^n$ may be possible.

Affine Bounding Functions

The affine bounding functions for polynomials based upon the Bernstein expansion, introduced in Section 10.2, may potentially be developed further.

- These affine lower bounding functions have already been tested in a simulated branch-and-bound environment. It would further be of interest to integrate these functions into a complete branch-and-bound solver for global optimisation problems and test them therein; for example the *COCONUT* software environment [Sch03], a general-purpose package for the solution of global optimisation and continuous constraint satisfaction problems, presents a potential modular black-box environment for such tests.

- A question of significant interest is whether a good quality affine bounding function utilising the implicit Bernstein form is possible. Of the described methods, it is only obvious that the constant bounding function can make use of the implicit Bernstein form. The generally best method, Method LLS (cf. Subsection 10.2.7) would seem to be impossible to adapt, since the linear least squares approximation requires all control points (Bernstein coefficients) explicitly. Methods based upon minimum slopes may be adaptable to the implicit Bernstein form, provided that the slopes are computed in coordinate directions only. Were the partial derivatives to be computed symbolically, polynomial sparseness would be preserved and the implicit Bernstein form could readily yield the minimum and maximum slope in each direction. A suitable modification of Method LP or LE may thus be possible.

- It is conjectured that the equilibriation transformation proposed in Subsection 10.2.9 may improve the performance of the affine bounding functions based upon minimum slopes. It remains to construct a precise formulation in the multivariate case — there are several possibilities — and test this.
- In principle, the Bernstein approach may be extended to the construction of tight bounding functions for arbitrary sufficiently differentiable functions, by using the Taylor expansion. A high-degree Taylor polynomial can be calculated, for which the Bernstein form and the resulting bounds or bounding functions can be computed, as before. The remainder of the Taylor expansion can be enclosed in an interval, by using established methods from interval analysis, e.g. the software package *ACETAF* [EN03]. Subtracting this interval from the lower bound of the Taylor polynomial, or from a lower bounding function with a downshift, provides a lower bound or lower bounding function for the given function. However, such Taylor polynomials are in general dense, for which the computational advantage of the implicit Berstein form is negated.

Bibliography

[Abe94] O Aberth. Computation of topological degree using interval arithmetic, and applications. *Mathematics of Computation*, 62(205):1–10, January 1994.

[AF96] C S Adjiman and C A Floudas. Rigorous convex underestimators for general twice-differentiable problems. *J. Global Optimization*, 9(1):23–40, 1996.

[AG90] E Allgower and K Georg. *Numerical Continuation Methods*, volume 13 of *Springer Series in Computational Mathematics.* Springer, Berlin, Heidelberg, New York, 1990.

[AG97] E Allgower and K Georg. Numerical path following. In P G Ciarlet and J L Lions, editors, *Handbook of Numerical Analysis, Vol. V, Techniques of Scientific Computing (Part 2)*, pages 3–207. Elsevier Sci. Publ., Amsterdam, Lausanne, New York, 1997.

[AH35] P Alexandroff and H Hopf. *Topologie.* Springer, Berlin, 1935.

[AH83] G Alefeld and J Herzberger. *Introduction to Interval Computation.* Academic Press, New York, 1983. Translated from 1974 publication, Wissenschaftsverlag, Mannheim.

[AMF95] I P Androulakis, C D Maranas, and C A Floudas. aBB: A global optimization method for general constrained nonconvex problems. *J. Global Optimization*, 7(4):337–363, 1995.

[AS92] O Aberth and M J Schaefer. Precise computation using range arithmetic, via C++. *ACM Trans. on Mathematical Software*, 18(4):481–491, 1992.

[BCR08] F Boudaoud, F Caruso, and M-F Roy. Certificates of positivity in the Bernstein basis. *Discrete and Computational Geometry*, 39(4):639–655, 2008.

[BD$^+$] M Berkelaar, J Dirks, et al. LP_SOLVE: Linear programming code. `http://lpsolve.sourceforge.net/`.

[Ber12] S N Bernstein. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Comm. Soc. Math. Kharkow*, 13:1–2, 1912.

[Ber00] J Berchtold. *The Bernstein Basis in Set-Theoretic Geometric Modelling.* PhD thesis, University of Bath, UK, 2000.

[BFLW09] T Beelitz, A Frommer, B Lang, and P Willems. A framework for existence tests based on the topological degree and homotopy. *Numerische Mathematik*, 111(4):493–507, 2009.

[Bjö04] A Björck. The calculation of linear least squares problems. In A Iserles et al., editors, *Acta Numerica 2004*, pages 1–53. Cambridge University Press, 2004.

[BMP03] H Brönnimann, G Melquiond, and S Pion. The Boost interval arithmetic library. In *Real Numbers and Computers*, pages 65–80, Lyon, 2003. `http://www.boost.org/`.

[Bow99] A Bowyer. *SvLis — Introduction and User Manual.* Information Geometers Ltd. and the University of Bath, 1999. `http://people.bath.ac.uk/ensab/G_mod/Svlis/book/node1.html`.

[Bro12] L E J Brouwer. Über Abbildung von Mannigfaltigkeiten. *Mathematische Annalen*, 71:97–115, 1912.

[BS95] G V Balaji and J D Seader. Application of interval Newton methods to chemical engineering problems. *Reliable Computing*, 1(3):215–223, 1995.

[BSV$^+$01] C Bliek, P Spellucci, L N Vicente, et al. Algorithms for solving nonlinear constrained and optimization problems: The state of the art, 2001. A progress report of the COCONUT project, available under `http://www.mat.univie.ac.at/~neum/glopt/coconut/StArt.html`.

[Buc70] B Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequationes Mathematicæ*, 4:374–383, 1970.

[BVH05] G Borradaile and P Van Hentenryck. Safe and tight linear estimators for global optimization. *Mathematical Programming*, 102(3):495–518, 2005.

[CD$^+$10] C Chen, J H Davenport, et al. Triangular decomposition of semi-algebraic systems. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation (ISSAC 2010)*, pages 187–194. ACM Press, New York, 2010.

[CS66] G T Cargo and O Shisha. The Bernstein form of a polynomial. *J. Res. Nat. Bur. Standards*, 70B:79–81, 1966.

[Dow90] M L Dowling. A fast parallel Horner algorithm. *SIAM Journal on Computing*, 19:133–142, 1990.

[Dre05] A Dreyer. *Interval Analysis of Analog Circuits with Component Tolerances.* Shaker-Verlag, Aachen, Germany, 2005.

[DS09] T Dang and D Salinas. Image computation for polynomial dynamical systems using the Bernstein expansion. In A Bouajjani and O Maler, editors, *Computer Aided Verification CAV'09*, volume 5643 of *Lecture Notes in Computer Science*, pages 277–287. Springer, 2009.

[DST93] J H Davenport, Y Siret, and E Tournier. *Computer Algebra.* Academic Press, London, second edition, 1993.

[EN03] I Eble and M Neher. ACETAF: A software package for computing validated bounds for Taylor coefficients of analytic functions. *ACM Trans. on Math. Software*, 29:263–286, 2003.

[ESS84] A Eiger, K Sikorski, and F Stenger. A bisection method for systems of nonlinear equations. *ACM Trans. on Mathematical Software*, 10(4):367–377, 1984.

[Far02] G Farin. *Curves and Surfaces in CAGD.* Morgan Kaufmann, San Francisco, fifth edition, 2002.

[Far12] R T Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29:379–419, 2012.

[Fau02] J-C Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation (ISSAC 2002)*, pages 75–83. ACM Press, New York, 2002.

[Fis90] H C Fischer. Range computations and applications. In C Ullrich, editor, *Contributions to computer arithmetic and self-validating numerical methods*, pages 197–211. J. C. Baltzer, Amsterdam, 1990.

[FL00] D Fausten and W Luther. Verifizierte Lösungen von nichtlinearen polynomialen Gleichungssystemen. Technical report, Gerhard Mercator University Duisburg, 2000. Schriftenreihe des Fachbereichs Mathematik no. SM-DU-477.

[Flo00] C A Floudas. *Deterministic Global Optimization: Theory, Methods, and Applications*, volume 37 of *Nonconvex Optimization and its Applications.* Kluwer Acad. Publ., Dordrecht, Boston, London, 2000.

[FR88] R T Farouki and V T Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1–26, 1988.

[FRI99] FRISCO — Framework for Integrated Symbolic-numeric Computation project, 1996–1999. `http://www.nag.co.uk/projects/FRISCO.html`.

[GAO] Gaol (not Just Another Interval Library). `http://gaol.sourceforge.net/`.

[Gar86] J Garloff. Convergent bounds for the range of multivariate polynomials. In K Nickel, editor, *Interval Mathematics 1985*, volume 212 of *Lecture Notes in Computer Science*, pages 37–56. Springer, Berlin, Heidelberg, New York, 1986.

[Gar93] J Garloff. The Bernstein algorithm. *Interval Computations*, 2:154–168, 1993.

[Gar00] J Garloff. Application of Bernstein expansion to the solution of control problems. *Reliable Computing*, 6:303–320, 2000.

[Gar09] J Garloff. Interval Gaussian elimination with pivot tightening. *SIAM J. Matrix Anal. Appl.*, 30(4):1761–1772, 2009.

[Gar10] J Garloff. Karl L. E. Nickel (1924–2009). *Reliable Computing*, 14:61–65, 2010.

[Ge12] J Garloff and A P Smith (editors). Special issue of the journal *Reliable Computing* on the Use of Bernstein Polynomials in Reliable Computing, to appear, 2012.

[GG99a] J Garloff and B Graf. Robust Schur stability of polynomials with polynomial parameter dependency. *Multidimensional Systems and Signal Processing*, 11:189–199, 1999.

[GG99b] J Garloff and B Graf. Solving strict polynomial inequalities by Bernstein expansion. In N Munro, editor, *The Use of Symbolic Methods in Control System Analysis and Design*, pages 339–352. The Institution of Electrical Engineers (IEE), London, 1999.

[GJS03a] J Garloff, C Jansson, and A P Smith. Inclusion isotonicity of convex-concave extensions for polynomials based on Bernstein expansion. *Computing*, 70:111–119, 2003.

[GJS03b] J Garloff, C Jansson, and A P Smith. Lower bound functions for polynomials. *J. Computational and Applied Mathematics*, 157:207–225, 2003.

[GLO] GLOBAL library. `http://www.gamsworld.org/global/globallib.htm`.

[GPS12] J Garloff, E D Popova, and A P Smith. Solving linear systems with polynomial parameter dependency with application to the verified solution of problems in structural mechanics. In A Chinchuluun, P M Pardalos, R Enkhbat, and E N Pistikopoulos, editors, *Proceedings of the International Conference on Optimization, Simulation and Control, Ulaanbaatar, Mongolia (2010)*, Springer Optimization and Its Applications. Springer-Verlag, to appear, 2012.

[Gra00] L Granvilliers. Towards cooperative interval narrowing. In *Proceedings 3rd International Workshop on Frontiers of Combining Systems (FroCoS'2000, Nancy, France)*, volume 1794 of *Lecture notes in artificial intelligence*. Springer, Berlin, Heidelberg, New York, 2000.

[Gra07] C Grandón. *Solution of Systems of Distance Equations with Uncertainty.* PhD thesis, Université de Nice-Sophia Antipolis, 2007.

[GS01a] J Garloff and A P Smith. Investigation of a subdivision based algorithm for solving systems of polynomial equations. *J. of Nonlinear Analysis: Series A Theory and Methods*, 47(1):167–178, 2001.

[GS01b] J Garloff and A P Smith. Solution of systems of polynomial equations by using Bernstein expansion. In G. Alefeld, S. Rump, J. Rohn, and T. Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 87–97. Springer, 2001.

[GS04] J Garloff and A P Smith. An improved method for the computation of affine lower bound functions for polynomials. In C A Floudas and P M Pardalos, editors, *Frontiers in Global Optimization*, volume 74 of *Nonconvex Optimization with its Applications*, pages 135–144. Kluwer Acad. Publ., Dordrecht, Boston, London, 2004.

[GS05] J Garloff and A P Smith. A comparison of methods for the computation of affine lower bound functions for polynomials. In C Jermann, A Neumaier, and D Sam, editors, *Global Optimization and Constraint Satisfaction*, volume 3478 of *Lecture Notes in Computer Science*, pages 71–85. Springer-Verlag, Berlin, Heidelberg, 2005.

[GS07] J Garloff and A P Smith. Guaranteed affine lower bound functions for multivariate polynomials. *Proc. Appl. Math. Mech.*, 7:1022905–1022906, 2007.

[GS08] J Garloff and A P Smith. Rigorous affine lower bound functions for multivariate polynomials and their use in global optimisation. *Lecture Notes in Management Science*, 1:199–211, 2008. Proceedings of the 1st International Conference on Applied Operational Research, Tadbir Institute for Operational Research, Systems Design and Financial Services.

[GSS12] J Garloff, A Schabert, and A P Smith. Bounds on the range of multivariate rational functions. *Proc. Appl. Math. Mech.*, submitted, 2012.

[Hal05] T C Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162:1065–1185, 2005.

[Han80] E R Hansen. Global optimization using interval analysis — the multidimensional case. *Numerische Mathematik*, 34:247–240, 1980.

[Han03] E R Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, Inc., New York, second edition, 2003.

[HK04] S Hongthong and R B Kearfott. Rigorous linear overestimators and underestimators. Technical report, University of Louisiana, 2004.

[HMSP96] C-Y Hu, T Maekawa, E C Sherbrooke, and N M Patrikalakis. Robust interval algorithm for curve intersections. *Computer-Aided Design*, 28:495–506, 1996.

[HS95a] H Hong and V Stahl. Bernstein form is inclusion monotone. *Computing*, 55(1):43–53, 1995.

[HS95b] H Hong and V Stahl. Safe starting regions by fixed points and tightening. *Computing*, 53(3–4):322–335, 1995.

[IEE] IEEE Interval Standard Working Group — P1788. `http://grouper.ieee.org/groups/1788/`.

[Int] Interval computations homepage. `http://www.cs.utep.edu/interval-comp/`.

[Jan00] C Jansson. Convex-concave extensions. *BIT*, 40:291–313, 2000.

[JR95] C Jäger and D Ratz. A combined method for enclosing all solutions of nonlinear systems of polynomial equations. *Reliable Computing*, 1:41–64, 1995.

[Jud85] G Judge. Developing an interest in polynomials — an example from the economics of investment decision making. *Teaching Mathematics and its Applications*, 4(3):127–129, 1985.

[Kan96] J S Kang. Wu stratification and retract decomposition. In *Effective Methods in Algebraic Geometry (MEGA)*, 1996.

[Kea79a] R B Kearfott. An efficient degree-computation method for a generalized method of bisection. *Numerische Mathematik*, 32:109–127, 1979.

[Kea79b] R B Kearfott. A summary of recent experiments to compute the topological degree. In *Proceedings of a Conference on Applied Nonlinear Analysis*, pages 627–633. Academic Press, New York, 1979.

[Kea96a] R B Kearfott. Interval computations: Introduction, uses and resources. *Euromath Bulletin*, 2(1):95–112, 1996.

[Kea96b] R B Kearfott. *Rigorous Global Search: Continuous Problems*, volume 13 of *Nonconvex Optimization and its Applications.* Kluwer Acad. Publ., Dordrecht, Boston, London, 1996.

[Kea08] R B Kearfott. Mainstream contributions of interval computations in engineering and scientific computing, 2008. `http://www.cs.utep.edu/interval-comp/kearfottPopular.pdf`.

[Kio78] J B Kioustelidis. Algorithmic error estimation for approximate solutions of nonlinear systems of equations. *Computing*, 19:313–320, 1978.

[KK$^+$92] R Klatte, U W Kulisch, et al. *PASCAL-XSC — Language Reference with Examples.* Springer-Verlag, New York, 1992.

[Knu86] D E Knuth. *MetaFont: the Program.* Addison-Wesley, 1986.

[Knu94] O Knuppel. PROFIL/BIAS — a fast interval library. *Computing*, 53(3–4):277–287, 1994.

[Kro69] L Kronecker. Über Systeme von Funktionen mehrerer Variabeln. *Monatsber. Berlin Akad.*, pages 159–193; 688–698, 1869.

[Kul08] U W Kulisch. *Computer Arithmetic and Validity: Theory, Implementation, and Applications.* De Gruyter, Berlin, New York, 2008.

[Laz09] D Lazard. Thirty years of polynomial system solving, and now? *J. of Symbolic Computation*, 44:222–231, 2009.

[Llo78] N G Lloyd. *Degree Theory.* Cambridge University Press, 1978.

[LLT08] T L Lee, T Y Li, and C H Tsai. HOM4PS-2.0: A software package for solving polynomial systems by the polyhedral homotopy continuation method. *Computing*, 83(2–3):109–133, 2008.

[Lor53] G G Lorentz. *Bernstein Polynomials.* Univ. Toronto Press, Toronto, 1953.

[LR81] J M Lane and R F Riesenfeld. Bounds on a polynomial. *BIT*, 21:112–117, 1981.

[LTWvG01] M Lerch, G Tischler, and J Wolff von Gudenberg. `filib++` — Interval library specification and reference manual. Technical Report 279, University of Würzburg, 2001.

[MG$^+$08] A Milani, G F Gronchi, et al. Topocentric orbit determination: Algorithms for the next generation surveys. *Icarus*, 195(1):474–492, May 2008.

[Mir41] C Miranda. Un' osservazione su un teorema di Brouwer. *Boll. Un. Mat. Ital. Ser. 2*, 3:5–7, 1941.

[MK80] R E Moore and J B Kioustelidis. A simple test for accuracy of approximate solutions to nonlinear (or linear) systems. *SIAM J. Numer. Anal.*, 17:521–529, 1980.

[MKC09] R E Moore, R B Kearfott, and M J Cloud. *Introduction to Interval Analysis.* SIAM, Philadelphia, 2009.

[MM90] K Meintjes and A P Morgan. Chemical equilibrium systems as numerical test problems. *ACM Trans. on Mathematical Software*, 16(2):143–151, 1990.

[Moo66] R E Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.

[Moo76] R E Moore. On computing the range of values of a rational function of n variables over a bounded region. *Computing*, 16:1–15, 1976.

[Moo79] R E Moore. *Methods and Applications of Interval Analysis*, volume 2 of *SIAM Studies in Applied Mathematics.* SIAM, Philadelphia, 1979.

[Mor87] A P Morgan. *Solving polynomial systems using continuation for engineering and scientific problems.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[MP09] B Mourrain and J-P Pavone. Subdivision methods for solving polynomial equations. *J. of Symbolic Computation*, 44:292–306, 2009.

[MS87] A P Morgan and V Shapiro. Box-bisection for solving second-degree systems and the problem of clustering. *ACM Trans. on Mathematical Software*, 13(2):152–167, 1987.

[MSW89] A P Morgan, A J Sommese, and L T Watson. Finding all isolated solutions to polynomial systems using HOMPACK. *ACM Trans. on Mathematical Software*, 15(2):93–122, 1989.

[NA07] P S V Nataraj and M Arounassalame. A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *Int. J. of Automation and Computing*, 4(4):342–352, 2007.

[NAG] Numerical Algorithms Group (NAG) Library. `http://www.nag.co.uk/numeric/numerical_libraries.asp`.

[Neu90] A Neumaier. *Interval Methods for Systems of Equations.* Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[Noo89] V W Noonburg. A neural network modelled by an adaptive Lotka-Volterra system. *SIAM Journal of Applied Mathematics*, 49(6):1779–1792, 1989.

[NR08] J W Nilsson and S A Riedel. *Electric Circuits.* Prentice-Hall, eighth edition, 2008.

[NS04] A Neumaier and O Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A*, 99:283–296, 2004.

[OR09] E Outerelo and J M Ruiz. *Mapping Degree Theory*, volume 108 of *Graduate Studies in Mathematics.* American Mathematical Society, 2009.

[OT75] T O'Neal and J Thomas. The calculation of topological degree by quadrature. *SIAM Journal of Numerical Analysis*, 12:673–680, 1975.

[PBP02] H Prautzsch, W Boehm, and M Paluszny. *Bézier and B-Spline Techniques.* Springer, Berlin, Heidelberg, 2002.

[POS95] PoSSo — Polynomial System Solving project, 1993–1995. `http://posso.dm.unipi.it/`.

[PRO] PROFIL/BIAS (Programmer's Runtime Optimised Fast Interval Library, Basic Interval Arithmetic Subroutines). `http://www.ti3.tu-harburg.de/keil/profil/index_e.html`.

[PS00] J M Peña and T Sauer. On the multivariate Horner scheme. *SIAM J. Numer. Anal.*, 37:1186–1197, 2000.

[Ray07] S Ray. *A New Approach to Range Computation of Polynomials using the Bernstein Form.* PhD thesis, Indian Institute of Technology, Dept. of Systems and Control Engineering, Bombay, India, 2007.

[RC95] D Ratz and T Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *J. Global Optimization*, 7:183–207, 1995.

[Ric96] D S Richardson. Solution of elementary systems of equations in a box in $\mathbb{R}^n$. In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation (ISSAC 1996)*, pages 120–126. ACM Press, New York, 1996.

[Ric99] D S Richardson. Weak Wu stratification in $\mathbb{R}^n$. *J. of Symbolic Computation*, 28(1–2):213–223, July 1999.

[Riv70] T J Rivlin. Bounds on a polynomial. *J. Res. Nat. Bur. Standards*, 74B:47–54, 1970.

[RN09] S Ray and P S V Nataraj. An efficient algorithm for range computation of polynomials using the Bernstein form. *J. Global Optimization*, 45:403–426, 2009.

[Rok77] J Rokne. Bounds for an interval polynomial. *Computing*, 18:225–240, 1977.

[Rok81] J Rokne. The centred form for interval polynomials. *Computing*, 27:339–348, 1981.

[Rok82] J Rokne. Optimal computation of the Bernstein algorithm for the bound of an interval polynomial. *Computing*, 28:239–246, 1982.

[RR88] H Ratschek and J Rokne. *New Computer Methods for Global Optimization.* Ellis Horwood Ltd., Chichester, 1988.

[Rum88] S M Rump. Algorithms for verified inclusions — theory and practice. *Reliability in Computing*, 19:109–126, 1988.

[Rum99] S M Rump. INTLAB — INTerval LABoratory. In T Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Acad. Publ., Dordrecht, 1999. `http://www.ti3.tu-harburg.de/rump/`.

[Rum05] S M Rump. Computer-assisted proofs and self-validating methods. In B Einarsson, editor, *Handbook on Accuracy and Reliability in Scientific Computation*, pages 195–240. SIAM, 2005.

[Sch03] H Schichl. Mathematical modeling and global optimization, 2003. Habilitationsschrift, University of Vienna, Cambridge University Press, to appear.

[Sik82] K Sikorski. Bisection is optimal. *Numerische Mathematik*, 40:111–117, 1982.

[Sik97] K Sikorski. *Optimal Solution of Nonlinear Equations.* Oxford Press, 1997.

[SJ05] W Stein and D Joyner. SAGE: System for Algebra and Geometry Experimentation. *ACM SIGSAM Bulletin*, 39(2):61–64, 2005. `http://www.sagemath.org/index.html`.

[Ske92] R Skeel. Roundoff error cripples Patriot missile. *SIAM News*, 25(4):11, 1992.

[Smi09] A P Smith. Fast construction of constant bound functions for sparse polynomials. *J. Global Optimization*, 43(2–3):445–458, 2009.

[SP93] E C Sherbrooke and N M Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10:379–405, 1993.

[Sta95] V Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations.* PhD thesis, Johannes Kepler Universität, Linz, September 1995.

[Sun58] T Sunaga. Theory of interval algebra and its applications to numerical analysis. *RAAG Memoirs*, 2:29–46, 1958.

[TF01] Y Tsai and R T Farouki. BPOLY: An object-oriented library of algorithms for polynomials in Bernstein form. *ACM Trans. on Mathematical Software*, 27(1):267–296, 2001.

[TP96] W Trump and H Prautzsch. Arbitrarily high degree elevation of Bezier representations. *Computer Aided Geometric Design*, 13(5):387–398, 1996.

[TS02] M Tawarmalani and N V Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and its Applications.* Kluwer Acad. Publ., Dordrecht, Boston, London, 2002.

[Ver99] J Verschelde. PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Trans. on Mathematical Software*, 25(2):251–276, 1999.

[VHMK97] P Van Hentenryck, D McAllester, and D Kapur. Solving polynomial systems using a branch and prune approach. *SIAM J. Numer. Anal.*, 34:797–827, 1997.

[Vra89] M N Vrahatis. A short proof and a generalization of Miranda's existence theorem. *Proc. Amer. Math. Soc.*, 107:701–703, 1989.

[Vra95] M N Vrahatis. An efficient method for locating and computing periodic orbits of nonlinear mappings. *Journal of Computational Physics*, 119:105–119, 1995.

[War56] M Warmus. Calculus of approximations. *Bull. Acad. Polon. Sci., Cl. III*, 4(5):253–259, 1956.

[Win96] F Winkler. *Polynomial Algorithms in Computer Algebra.* Springer, Vienna, New York, 1996.

[Wu84] W-T Wu. Basic principles of mechanical theorem proving in elementary geometries. *J. Sys. Sci. and Math. Sci.*, 1(3):207–235, 1984.

[WW92] D Weber-Wulff. Rounding error changes Parliament makeup. *The Risks Digest*, 13(37), 1992.

[XSCa] History of XSC-Languages and Credits. `http://www2.math.uni-wuppertal.de/wrswt/xsc/history.html`.

[XSCb] XSC Languages (C-XSC, PASCAL-XSC). `http://www.xsc.de/`.

[You31] R C Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104:260–290, 1931.

[ZG98] M Zettler and J Garloff. Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Trans. Automat. Contr.*, 43:425–431, 1998.

[Zha05] H Zhang. *Nondeterministic Linear Static Finite Element Analysis: An Interval Approach.* PhD thesis, School of Civil and Environmental Engineering, Georgia Institute of Technology, 2005.

[ZN88] S Zuhe and A Neumaier. A note on Moore's interval test for zeros of nonlinear systems. *Computing*, 40:85–90, 1988.

[Zum08] R Zumkeller. *Global Optimization in Type Theory.* PhD thesis, École Polytechnique, Paris, 2008.

A Software

Here we give an overview of two of the main software packages that were developed during the course of this research.

A.1 Topological Degree Software

This section summarises the software for Brouwer (topological) degree computations that was created to undertake the experiments described in Chapter 7. The software package was implemented in C++ and designed in a modular fashion to use an interval arithmetic implementation of choice; in the first instance a custom interval arithmetic package was used; subsequently the *filib++* library [LTWvG01] was employed. An alternative software package (e.g. utilising advanced interval algorithms to enhance the performance) can be substituted without affecting the system for degree computation. The interval arithmetic implementation is however required to have the property that degenerate intervals (i.e. point values) and non-degenerate intervals can be used interchangably, since faces are gradually reduced in dimension (becoming child faces) by successively restricting interval fields to point values. We wish to evaluate (interval extensions for) functions over a face by applying interval arithmetic to such an aggregate of non-degenerate and degenerate intervals.

We can identify a three-tiered hierarchy to the processing of data in this algorithm: there are low-level operations performed on intervals, mid-level operations performed on faces (which are composed of intervals), and high-level operations performed on arrays or sets of faces (in which intervals are effectively doubly-nested). This hierarchy naturally lends itself to an object-oriented approach where the the algorithm can essentially be expressed as a face generation and processing scheme.

Below we outline the main data structures used by the software, the main degree computation routine, and the different output modes.

Data Structure: Function

The software permits three different modes for function input and evaluation with interval arguments:

1. The component functions are coded inline using native floating-point arithmetic. This allows an expert user to optimise the implementation in order to minimise the effect of the dependency problem, but requires non-trivial code rewriting and compilation for each new problem instance.

2. The component functions are coded inline using an interval arithmetic library. The ease of this depends on the interval arithmetic package used; for a library such as

filib++, for example, an interval-valued function can be written in exactly the same way as one returning a floating-point value. The code still requires rewriting and compilation each time, although in an easy fashion which can be done by a non-expert.

3. The functions are entered symbolically during run-time (e.g. by a non-expert user) and stored as their own data type. (For brevity, we omit the technical details of the input format and parser.) This has the advantage of not requiring any recompilation and is thus the mode that was used for the results presented herein.

In the latter case, a recursive data structure is used whereby each function is represented as a binary tree. The nodes consist of operators and the leaves are literal values and variable instances. An operator node has one or two branches (sub-trees), depending on the number of arguments required. The parenthetic level is related to depth in the tree, in that operators of lower precedence occur higher up.
Function:

- n: number of variables (integer)
- *node*: (one of a number of fixed values for simple unary or binary operators)
- c: constant (floating-point)
- i: variable index or exponent (natural number)
- f_l: first sub-tree (pointer to function)
- f_r: first sub-tree (pointer to function)

Data Structure: Box

Given a dimension n, a box (cf. Definition 2.2) in $\mathbb{R}^n$ has a straightforward representation as an array of n intervals.
Box:

- n: number of variables (integer)
- x: box width (array of n intervals)

Data Structure: Face

A face in $\mathbb{R}^n$ is similar to a box, but slightly more complicated, in that an orientation field is required and also the dimension of the face (defined as the number of non-degenerate interval fields) is less than the number of variables. A child face therefore has a lower dimension than its parent, but they share the same n.
Face:

- n: number of variables (integer)

- d: dimension (integer, where $d < n$)
- x: face width (array of n intervals, $n - d$ of which are degenerate)
- σ: orientation (integer; either $+1$ or -1)

Routine: Evaluate

Where the above datatype for functions is used, such a function can be evaluated over a (sub-)face in a recursive fashion. The relevant interval operation is called for each operator node, with arguments given by the result of the recursive evaluation over the sub-trees. Variables are substituted by the interval or point values for the corresponding fields of the face.
INPUT:

- f_i: (single-valued) function (function)
- s: (face)

OUTPUT:

- $\mathbf{y}$: an interval enclosure for $f(s)$ (interval)

Routine: TDegree

Conceptually, the degree finder is expressed as a single function, taking an array of symbolic functions and a box as input, and returning a single integer. Main subroutines generate the initial face array L, subdivide and check faces, and search and resolve overlaps, utilising a hierarchy of array, face, and interval processes.
INPUT:

- n: number of variables (integer)
- $\mathcal{F}$: (multi-valued) function (array of n functions)
- $\mathbf{X}$: (box)

OUTPUT:

- $deg(\mathcal{F}, \mathbf{X}, 0)$: Brouwer degree of $\mathcal{F}$ at 0 relative to $\mathbf{X}$ (integer)

Output

The result of a topological degree computation is a single integer; nevertheless four different options for program output are implemented:

- **Terse Output**: If no diagnostic information is required, the program merely returns the (integer) result.

- **Tabular Output**: For an analysis of the algorithm's characteristics, as undertaken in Chapter 7, a concise chart of diagnostic information for each major iteration is displayed, just as in Tables 7.1 to 7.4.
- **Graphical Output**: For a more complete illustration of the face subdivision process, this option produces a pseudo-graphical illustration of the tree of sub-faces. It is thus possible to observe the structure of this tree (e.g. problem branches, node clustering) in a way that is not possible with the tabular output. An example is given in Figure A.1; `*` indicates a subdivision (i.e. a non-terminal face), `0` a terminal face that is discarded, and `+->` a terminal face that is retained.

```
*---0                  *---*---0              0     +->       0      0
    0                      |   *---0
    *---0                  |   |   0
    |   *---0              |   |   0
    |   |   0              |   |   0
    |   |   0              |   0
    |   |   0              |   *---0
    |   0                  |       *---0
    |   0                  |       |   0
    *---*---0              |       |   0
        |   0              |       |   0
        |   *---0          |       0
        |   |   0          |       +->
        |   |   0          *---0
        |   |   0          |   0
        |   0              |   *---0
        0                  |   |   0
        *---0              |   |   +->
        |   0              |   |   0
        |   0              |   0
        |   0              +->
        0                  +->
```

Figure A.1: Example subdivision trees for the six faces (two non-terminal and four terminal) of a box in $\mathbb{R}^3$.

- **Verbose Output**: This option is useful only for debugging and very detailed low-level analysis. Every single face that is generated during the subdivision process is displayed, and it is stated whether it is discarded, retained, or subdivided. For large examples, this option can generate several megabytes of text.

A.2 Bernstein Expansion Software

The algorithms detailed in Section 3.3 and in Subsection 9.3.2 are implemented in *BeBP*, a C++ software package for the computation of Bernstein-based affine bound functions for polynomials. Interval arithmetic is used extensively throughout, for which the *filib++* library [LTWvG01] is employed. As noted in Section 3.3, there is very little previously existing software in the public domain that deals with the Bernstein expansion either rigorously or in the multivariate case.

We give here a brief overview of this software package. The following subsections introduce the data structures used by the program, and some of the most important routines available. By default, the program operates in a rigorous mode, whereby the coefficients of polynomials and Bernstein coefficients are stored and manipulated as intervals with double-precision floating point endpoints. This delivers bounds which are guaranteed to be valid.

There are two versions of the main data structure and routines (with the exception of the bounding functions), one for the standard Bernstein form, and one for the implicit Bernstein form.

Data Structure: Polynomial

Polynomials are passed to the program in a sparse representation, consisting of a one-dimensional array of non-zero terms. The ordering of the terms in the polynomial is unimportant. Each term thus consists of a coefficient with an array of associated variable exponents. Coefficients are stored as intervals; they may be entered as point values, in which case they are converted to intervals of machine-precision width. Implicit in this data structure are n, the number of variables, and l, the degrees in each variable. These values are instead stored alongside the polynomial in the BCPB data structure, below.
Polynomial:

- k: number of terms (integer)
- t: terms (array of k terms); each term consists of
  - a: coefficient (interval)
  - i: multi-index exponent (array of n integers)

Data Structure: BCPB

The principal data construct created by the program, designed as a 'workspace' for applications, is an aggregate structure called a BCPB (Bernstein coefficients, polynomial, box). By storing the Bernstein coefficients in such a workspace alongside the corresponding polynomial and box, they do not need to be recomputed from scratch in subsequent application iterations; the more efficient subdivision-based scheme can be used.
BCPB:

- n: number of variables (integer)

- l: degrees in each variable $l_1, \ldots, l_n$ (array of n integers)
- p: polynomial (polynomial)
- $\mathbf{X}$: box (array of n intervals)
- b: Bernstein coefficients ($(l_1+1) \times \ldots \times (l_n+1)$ array of intervals)

Data Structure: IBCPB

An alternative to the BCPB data structure, above, is an aggregrate which makes use of the implicit Bernstein form (cf. Section 9.3). This workspace has the same data fields as the BCPB, but the Bernstein coefficients of the component univariate monomials are stored, instead of the whole set of the Bernstein coefficients.
IBCPB:

- n: number of variables (integer)
- l: degrees in each variable $l_1, \ldots, l_n$ (array of n integers)
- p: polynomial (polynomial)
- $\mathbf{X}$: box (array of n intervals)
- b: Bernstein coefficients ($k \sum_{i=1}^{n} l_i + 1$ array of intervals)

Routine: Initialise

Given a polynomial and box as input, this procedure initialises a workspace and calculates and stores the corresponding Bernstein coefficients. In the case of the usual Bernstein form, the entire set of coefficients are computed, according to the algorithm in Subsection 3.3.1; in the case of the implicit Bernstein form, instead those of the component univariate monomials are computed.
INPUT:

- n: number of variables (integer)
- l: degrees in each variable (array of n integers)
- p: polynomial (polynomial)
- $\mathbf{X}$: box (array of n intervals)

OUTPUT:

- w: work structure (pointer to BCPB or IBCPB)

Routine: Subdivide

Given a workspace as input (containing a box and Bernstein coefficients), this procedure performs a subdivision into two sub-boxes, according to the algorithm in Subsection 3.3.2. The old workspace is destroyed and two new workspaces are created. The boxes and Bernstein coefficients are updated for each new workspace.
INPUT:

- w: work structure (pointer to BCPB or IBCPB)
- d: subdivision direction (integer)
- x_λ: split point (floating point)

OUTPUT:

- w_1: work structure (pointer to BCPB or IBCPB)
- w_2: work structure (pointer to BCPB or IBCPB)

Routine: Range

Given a workspace as input (containing a polynomial and a box), this procedure returns a tight outer estimation for the range of the polynomial over the box. This range is equal to the Bernstein enclosure, i.e. the range spanned by the minimum and maximum Bernstein coefficients. In many cases (often for small boxes and/or where the polynomial is monotonic over the box), the range is provided without overestimation (except for the outward rounding which is inherent in the interval arithmetic). Due to the use of interval arithmetic, this range is a guaranteed outer estimation. In the case of the implicit Bernstein form, the algorithm in Subsection 9.3.2 is used.
INPUT:

- w: work structure (pointer to BCPB or IBCPB)

OUTPUT:

- r: range (interval)

Routine: Derivative

Given a workspace as input (containing a polynomial and Bernstein coefficients), together with a choice of one of the variables, this procedure calculates the Bernstein coefficients of the partial derivative of the polynomial with respect to the chosen variable, over the same box, according to Subsection 3.3.4. These coefficients are placed into a new workspace which is created, along with the derivative of the polynomial and the same box.
INPUT:

- w: work structure (pointer to BCPB or IBCPB)

- d: index of differentiation variable (integer)

OUTPUT:

- w_1: work structure (pointer to BCPB or IBCPB)

Routine: LowerBF (6 Versions)

Given a workspace as input (containing a polynomial and a box), this procedure returns a tight affine lower bounding function for the polynomial over the box. The workspace must be of type BCPB, i.e. the entire set of Bernstein coefficients is explicity required. The coefficients $a_0, \ldots, a_n$ of the bound function are returned in an array. There are six versions of this routine, according to the variant methods described in Subsections 10.2.2 to 10.2.7.

For the variant which requires it (Method LP), the *LP_SOLVE* [BD$^+$] package for solving linear programming problems is used. Other methods may require the solution of a system of linear equations; here a Householder transformation is employed to yield a QR decomposition. In the case of the method using the linear least squares approximation (Method LLS), it is worth noting that the bulk of the computation does not need to be performed using interval arithmetic; the midpoints of the Bernstein coefficient intervals are used to construct the linear least squares problem. Only the final stage of the computation, the downward shift, needs to be performed with interval arithmetic. This suffices for the lower bounding function to be guaranteed to be valid.
INPUT:

- w: work structure (pointer to BCPB)

OUTPUT:

- c: affine function (array of $n+1$ coefficients(floating point))

Routine: UpperBF (6 Versions)

This procedure is entirely analogous to LowerBF (in all variants), except that an affine upper bounding function is computed.
INPUT:

- w: work structure (pointer to BCPB)

OUTPUT:

- c: affine function (array of $n+1$ coefficients(floating point))

Routine: AffineAF

This procedure takes the same input as LowerBF (and UpperBF), but instead yields an affine function which closely approximates the polynomial over the box. As in Subsection 10.2.7, a linear least squares approximation is employed in the same fashion, except that the final stage, the downward (or upward) shift, is omitted. This then yields an affine function which closely approximates the polynomial over the box. It may be of use for applications in which an affine approximator, instead of an affine relaxation, is required. As before, the coefficients $a_0, \ldots, a_n$ of the approximating function are returned in an array.
INPUT:

- w: work structure (pointer to BCPB)

OUTPUT:

- c: affine function (array of $n + 1$ coefficients(floating point))