INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

DISSERTATION

CELL EXCLUSION ALGORITHMS

Submitted by

Melissa Erdmann

Department of Mathematics

In partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2001

COLORADO STATE UNIVERSITY

April 20, 2001

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY MELISSA ERDMANN ENTITLED "CELL EXCLUSION ALGORITHMS" BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT

CELL EXCLUSION ALGORITHMS

This dissertation will address two problems which frequently arise in applications: finding the real zeros of a nonlinear system of equations and finding the minimum real value of a function of several variables. Just to name a few, the fields of chemistry, biology, physics, robotics, and economics involve zero-finding problems. Optimization problems are also extremely prevalent. One important optimization problem is that of minimizing cost.

Cell exclusion algorithms apply an exclusion condition to some region, e.g., a cell, in which we expect all zeros to be found or on which we wish to determine the global minimum. Since an exclusion condition is a necessary but not sufficient condition for a cell to contain a zero or a point at which a function achieves its global minimum, a successful algorithm bounds the number of cells which remain at each iteration. Thus, we want to get as few false positive cells, i.e., cells which satisfy the condition but do not contain a zero or a point at which a function achieves its global minimum, as possible. We develop localized conditions which are more stringent than those which have been given in previous literature. More stringent exclusion conditions discard more cells and hence are more efficient.

In this dissertation we develop the theory behind zero-finding and optimization cell exclusion algorithms. We present both types of algorithms. Several different root conditions are introduced and their effectiveness upon implementation is analyzed. Indeed, we give multiple numerical examples.

Melissa Erdmann
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Summer 2001

ACKNOWLEDGEMENTS

Many thanks to Dr. Eugene L. Allgower and Dr. Kurt Georg for their ideas, guidance, and encouragement.

TABLE OF CONTENTS

Introduction

This dissertation will address two problems which frequently arise in applications: finding the real zeros of a nonlinear system of equations and finding the minimum real value of a function of several variables. In biology, for example, problems involving circulation within the heart and problems in neurophysiology may be translated into zero-finding problems. Chemistry equilibrium problems correspond to zero-finding problems. In addition, just to name a few, the fields of physics, robotics, and economics involve zero-finding problems. It is useful to be able to find all zeros of a system of equations and then choose the best one according to certain criteria. For example, in robotics a zero of a system may represent a position of a robot arm that must occur. One may then choose the zero that corresponds to the most mechanically efficient means of reaching this position. Optimization problems are also extremely prevalent. One important optimization problem is that of minimizing cost.

Very few algorithms effectively locate all of the real zeros of general functions beyond cell exclusion algorithms and the closely related interval methods. In the case of polynomial maps there are effective homotopy methods which locate all of the complex solutions. The literature concerning this approach is by now extensive. Recent surveys for this approach are given for example in [1, 14]. The drawbacks of these methods are that polynomial maps are not general and the number of complex solutions may be extremely large, whereas the number of real solutions may be very much smaller. An attempt to find multiple solutions in conjunction with

Newton's method by means of repeated deflation is given in [5]. It was discovered that not all solutions can be found by this approach and successive deflations do not necessarily yield nearby roots. In [2] deflation is related to homotopy methods.

Concerning methods for global optimization, a much richer literature exists. For example, see [8]. In fact, there is even a journal named for the topic. Since we have not yet implemented a cell exclusion algorithm for global optimization, we cannot discuss how cell exclusion algorithms compare with other methods.

Cell exclusion algorithms have been used for many years in the area of interval analysis, see e.g. [3, 10, 9, 12, 15]. One can easily describe the basic structure of such an algorithm. We begin with some region, e.g., a cell, in which we expect all zeros to be found or on which we wish to determine the global minimum. The algorithm is based on a given root condition or optimization condition which can be applied to each cell. If a cell fails the condition, we know it will not contain a zero or a point at which a function achieves its global minimum, and it can be discarded. If a cell satisfies the condition, the cell is subdivided and the condition is applied to the new, smaller cells. This leads to a recursive algorithm. The success and efficiency of these algorithms will strongly depend on the choice of the condition. Hence, the main purpose of this dissertation is to develop and analyze various conditions with the aim of deriving exclusion criteria which are numerically more efficient.

When finding the real, isolated zeros of a system of equations, cell exclusion algorithms may be used in conjunction with Newton's method. Given the same initial domain as a cell exclusion algorithm, Newton's method may not converge to all solutions of a system of equations, especially if two zeros are close to each other or if a zero has a small basin of attraction. Cell exclusion algorithms may be used to determine components corresponding to each of the isolated zeros. The

midpoint of each component may then be used as a starting point for Newton's method in order to determine more accurate approximations of the zeros.

Since an exclusion condition is a necessary but not sufficient condition for a cell to contain a zero or a point at which a function achieves its global minimum, a successful algorithm bounds the number of cells which remain at each iteration. Thus, we want to get as few false positive cells, i.e., cells which satisfy the condition but do not contain a zero or a point at which a function achieves its global minimum, as possible. We develop localized conditions which are more stringent than those which have been given in previous literature. More stringent exclusion conditions discard more cells and hence are more efficient.

In this dissertation we develop the theory behind zero-finding and optimization cell exclusion algorithms. We present both types of algorithms. Several different root conditions are introduced and their effectiveness upon implementation is analyzed. Indeed, we give multiple numerical examples.

List of Symbols, Abbreviations, and Terms

A set σ is a **cell** if it is an N-dimensional region	$\sigma = \prod_{i=1}^{N}[a_{i1}, a_{i2}]$ where $a_{ij} \in \mathbb{R}$, $a_{i1} < a_{i2}$
The **midpoint** of a cell σ	$m_\sigma = \begin{pmatrix} \frac{1}{2}(a_{11} + a_{12}) \\ \vdots \\ \frac{1}{2}(a_{N1} + a_{N2}) \end{pmatrix}$
The **mesh vector** of a cell σ	$d_\sigma = m_\sigma - \begin{pmatrix} a_{11} \\ \vdots \\ a_{N1} \end{pmatrix}$
The **lowermost corner** of a cell σ	$\underline{\sigma} = (a_{11}, a_{21}, \ldots, a_{N1})$
The **uppermost corner** of a cell σ	$\overline{\sigma} = (a_{12}, a_{22}, \ldots, a_{N2})$
Norm:	$\|x\|_\infty = \sup_{1 \le i \le N} \|x_i\|$
Mesh size of a cell σ with mesh vector d_σ	$\|d_\sigma\|$
Cell where we are searching for zeros or points where a function attains its global minimum	Λ
The set of zeros in the cell Λ of a system of equations $F(x) = 0$, where $F : \Lambda \subset \mathbb{R}^N \to \mathbb{R}^N$	$N_F(\Lambda)$
The set of points where a function $f : \Lambda \subset \mathbb{R}^N \to \mathbb{R}$ attains its global minimum in the cell Λ	$\mathcal{M}_f(\Lambda)$
Cell Exclusion Algorithm	CEA
A **root condition** is a computationally verifiable necessity test for the presence of a zero in a cell.	R.C.
A **minimization condition** is a computationally verifiable necessity test for the presence of a global minimum in a cell.	M.C.
A cell, $\sigma \subset \mathbb{R}^N$, has undergone one **level of partitioning** when it has been subdivided along each of its N dimensions.	

Set of cells which satisfy the root/minimization condition on the kth level of partitioning — Ω_k

Set of cells which result when Ω_{k-1} undergoes one level of partitioning — Γ_k

Sequential bisection is the partitioning which takes place when cells are bisected along one axis at a time.

Simultaneous bisection is the partitioning which takes place when cells are bisected along all axes at the same time.

The **length** of a multi-index $\alpha \in \mathbb{Z}_+^n$ — $|\alpha| = \sum_i \alpha_i$

The **factorial** of a multi-index α — $\alpha! = \prod_i \alpha_i!$

For a multi-index α and $x \in \mathbb{R}^N$ — $x^\alpha = \prod_i x_i^{\alpha_i}$

The **partial derivative** involving a multi-index α — $\partial^\alpha = (\alpha!)^{-1} \prod_i \partial_i^{\alpha_i}$

The sequence of minimum values obtained from the optimization algorithm on the kth level of partitioning — M_k

The Hessian of a function f — H_f

Chapter 1

CELL EXCLUSION ALGORITHMS AND ZERO FINDING

In this chapter we consider the problem of determining all real solutions of a nonlinear system of equations of the form:

$$F(x) = 0$$

where $F : \Lambda \subseteq \mathbb{R}^N \rightarrow \mathbb{R}^N$ is continuous and Λ is a compact region in $\mathbb{R}^N$.

1.1 Introductory Definitions

The following are some background definitions:

Definition 1.1.1 (Cell). *A set σ is said to be a cell if it is an N-dimensional region of the form*

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

where $a_{ij} \in \mathbb{R}$, $a_{i1} < a_{i2}$.

Definition 1.1.2 (Face). *We define a face ω of a cell σ to be a subset $\omega \subset \sigma$ such that*

$$\omega = \prod_{i=1}^{N} [a_{i1}, a_{i2}]$$

where $a_{i1} = a_{i2}$ for at least one i.

Definition 1.1.3 (Midpoint). *The midpoint of a cell σ is defined to be*

$$m_\sigma = \begin{pmatrix} \frac{1}{2}(a_{11} + a_{12}) \\ \vdots \\ \frac{1}{2}(a_{N1} + a_{N2}) \end{pmatrix}.$$

Often we will simply use m to denote the midpoint.

Definition 1.1.4 (Mesh Vector). *The mesh vector of a cell σ is defined to be*

$$d_\sigma = m_\sigma - \begin{pmatrix} a_{11} \\ \vdots \\ a_{N1} \end{pmatrix}.$$

Often we will use d to denote the mesh vector. The mesh vector represents the vector of distances from the midpoint of the cell to the faces of the cell.

Definition 1.1.5 (Mesh Size). *The mesh size of a cell σ with mesh vector d_σ is defined to be $\|d_\sigma\|$.*

Unless otherwise noted, the norm which we use is the infinity norm.

Definition 1.1.6 (Cellular Partition). *Let Γ be a finite set of cells and let Λ be a cell in $\mathbb{R}^N$. We say that Γ is a cellular partition of Λ if*

1. $\Lambda = \bigcup_{\sigma \in \Gamma} \sigma$.
2. *If $\sigma_1, \sigma_2 \in \Gamma$, then $\sigma_1 \cap \sigma_2$ is a common face of σ_1 and σ_2 or $\sigma_1 \cap \sigma_2 = \emptyset$.*
3. *The number of cells, $\sigma \in \Gamma$ such that $\sigma \cap \Lambda \neq \emptyset$ is finite.*

Definition 1.1.7 (Refinement). *Let Γ_1 and Γ_2 be any two cellular partitions of Λ. Γ_2 is said to be a refinement of Γ_1 if*

$$\forall_{\sigma_2 \in \Gamma_2} \exists_{\sigma_1 \in \Gamma_1} \quad \text{such that } \sigma_2 \subset \sigma_1$$

with strict inclusion holding in at least one case. We also say Γ_2 is finer than Γ_1.

A cell exclusion algorithm looks for zeros in some initial cell Λ. As the algorithm executes, Λ is partitioned into successively refined partitions.

1.2 A Cell Exclusion Algorithm

A cell exclusion algorithm (CEA) systematically discards cells as it progresses. In order to do this, the algorithm makes use of some test which we will refer to as a root condition. A root condition is a necessary, but not sufficient, condition which must be satisfied if a zero point is present in a cell. Thus, if a cell fails the root condition, we know it does not contain a zero and may be discarded immediately. This is one of the desirable feature of CEAs. One does not want to spend a great deal of time searching for zeros in a region where none can be found. In this dissertation we will discuss the application of various root conditions. Now we formally define a root condition:

Definition 1.2.1 (Root Condition). *A root condition is a computationally verifiable necessity test. $R(\sigma)$, for the presence of a zero in a cell, σ.*

As a result,

$$\exists_{x \in \sigma} F(x) = 0 \implies R(\sigma) = \text{yes}$$

$$R(\sigma) = \text{no} \implies \nexists_{x \in \sigma} F(x) = 0 .$$

Definition 1.2.2 ($N_F(\Lambda)$). *We define $N_F(\Lambda)$ to be the set of zeros in the cell Λ of a system of equations $F(x) = 0$, where $F : \Lambda \subset \mathbb{R}^N \to \mathbb{R}^N$.*

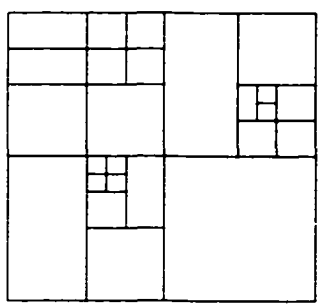
We will now give some introductory definitions and a more formal description of the cell exclusion algorithm which we sketched in the introduction. The algorithm is based on a given sequence of refining partitions Γ_k and uses at each stage a given root condition to exclude all cells which cannot contain a root of F. The remaining cells are then stored in Ω_k.

Definition 1.2.3 (Level of Partitioning). *A cell, $\sigma \subset \mathbb{R}^N$, has undergone one level of partitioning when it has been subdivided along each of its N dimensions.*

Definition 1.2.4 (Ω_k). *The set of cells which satisfy the root condition on the kth level of partitioning is called* Ω_k.

Definition 1.2.5 (Γ_k). *The set of cells which result when* Ω_{k-1} *undergoes one level of partitioning is called* Γ_k.

Figure 1.1 illustrates the number of cells in Γ_k for various levels of partitioning.



$\#(\Gamma_k)$	k
1	0
4	1
8	2
10	3
6	4

Figure 1.1: Illustration of Levels of Partitioning

Algorithm 1.2.1.

1. *Let* Γ_k *be a sequence of cellular partitions of* Λ *with* $\Gamma_0 = \{\Lambda\}$ *such that* Γ_{k+1} *is finer than* Γ_k , $k = 0, 1, \ldots$, *and such that the mesh sizes* $\lim_{k\to\infty} \|d_k\| = 0$.

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

3. *We assume that a given root condition can be implemented for each cell* $\sigma \subseteq \Lambda$, *and we assume that* Λ *satisfies this root condition (otherwise there would be no zeros in* Λ*).*

4. *Set* $\Omega_0 \leftarrow \{\Lambda\}$. *(Initialization)*

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

   (a) $\Omega_{k+1} \leftarrow \emptyset$.

   (b) For $\sigma \in \Omega_k$

For τ such that ($\tau \in \Gamma_{k+1}$ and $\tau \subset \sigma$)

If τ satisfies the root condition,

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

1.3 Optimality of Trisection

As a CEA executes, it partitions cells which satisfy the root condition by dividing successively along each axis. In this section we investigate the optimal way for this subdivision to take place, i.e., into how many cells a cell which satisfies the root condition should be partitioned as the algorithm progresses.

We will consider complexity to be measured by the number of times which we check the root condition.

Definition 1.3.1 (Asymptotically Optimal). *A sequence of cellular refinements $\{\Gamma_k\}$ of a cell Λ is said to be asymptotically optimal if there is a $k_0 > 0$ such that $\{\Gamma_k\}$ minimizes the complexity of the cell exclusion algorithm when neglecting the first k_0 steps of computation.*

Theorem 1.3.1 (Optimality of Trisection). *If $\{\Gamma_k\}$ is a sequence of cellular refinements that satisfy the condition:*

$$\#(\Omega_k) = C \qquad C \in \mathbb{N} \qquad k \geq k_0$$

then the sequence of refinements which minimizes the complexity of the cell exclusion algorithm is successively dividing in thirds along each axis.

Proof. 1. The ratio of the "volume" of the N-cube which would be formed by the longest side of cells in the k-th partition, $(2\|d_k\|)^N$, of cells in the k-th partition to the "volume" of cells in the k-th partition, $v(\Gamma_k)$, is bounded by some constant M. In other words, we will not have cells which are arbitrarily long in one or more directions. This is a reasonable assumption. We do not

want to have long, skinny cells. Thus, we can relate the computational complexity in the change in volume of cells to a change in the overall tolerance, i.e. mesh size, of a partition.

$$\frac{(2\|d_k\|)^N}{v(\Gamma_k)} \leq M \qquad\qquad M \in \mathbb{R}$$

2. We will subdivide each cell satisfying the root condition into the same number of pieces, p, along any axis.

Now let us consider the computational complexity of subdividing. We begin with a cell σ in Ω_k with volume a. After subdividing some number of times we assume we will have a cell in Ω_l with a smaller volume b. This change in volume is given by:

$$\frac{1}{p^{(l-k)}}a = b \Longrightarrow (l-k) = \frac{\ln(a/b)}{\ln p}.$$

Thus, we see the total amount of computational effort involved in subdividing Ω_k into Ω_l is proportional to:

$$(l-k)pC = \frac{\ln(a/b)}{\ln p}pC.$$

Since we fix our beginning and ending volumes, a and b, the only variable in this equation is p. Thus, our problem becomes one of minimizing $f(p) = \frac{p}{\ln(p)}$ for $p \in \mathbb{N}$. This function achieves its minimum at $p = e$. Since p should, however, be an integer, the optimum value is $p = 3$. We see $f(3) = 2.7307$,
$f(2) = f(4) = 2.8854$. □

We did some numerical experiments using trisection. However, since the costs of bisection and trisection are very close, unless otherwise noted we bisect in the numerical algorithms for the sake of simplicity.

1.4 Consequences of the Cell Exclusion Algorithm

The following theorem summarizes straightforward consequences of the cell exclusion algorithm 1.2.1.

Theorem 1.4.1. *Let $N_F(\Lambda)$ be the solution set of the system $F(x) = 0$ with $x \in \Lambda$ and $\{\Omega_k\}$ be the cell sequence created by the cell exclusion algorithm 1.2.1, then the following hold:*

1. *We have monotonic enclosure of the solutions. That is, for all k if $\tilde{x} \in N_F(\Lambda)$, then $\tilde{x} \in \sigma$ for some $\sigma \in \Omega_k$, and all cells which are elements of Ω_{k+1} are elements of partitions of cells in Ω_k.*

2. *We may determine zeros to within a given tolerance. If $\tilde{x} \in N_F(\Lambda)$ and $\tilde{x} \in \sigma$, then $\|m_\sigma - \tilde{x}\| \leq \|d_\sigma\|$.*

3. *For every zero there is a sequence of cells which converges to this zero. If $N_F(\Lambda) \neq \emptyset$, then for any $\tilde{x} \in N_F(\Lambda)$, there is a sequence of cells, $\sigma_k \in \Omega_k$, such that*

$$\|m_{\sigma_k} - \tilde{x}\| \to 0 \qquad \text{as} \qquad \|d_{\sigma_k}\| \to 0\,.$$

4. *If the algorithm terminates in a finite number of steps, then no solution to the system exists.*

Chapter 2

A LIPSCHITZ ROOT CONDITION

In this chapter we will examine the first root condition, the Lipschitz root condition which is introduced in [20]. We may use the fact that F satisfies the Lipschitz condition to arrive at a root condition.

Suppose F satisfies a Lipschitz condition:

$$\|F(x) - F(y)\| \leq K\|x - y\|$$

where K is a Lipschitz constant.

Assume $\tilde{x} \in \sigma$ and $F(\tilde{x}) = 0$. If there exists a solution $\tilde{x} \in \sigma$, then

$$\|F(m_\sigma) - F(\tilde{x})\| = \|F(m_\sigma)\| \leq K\|m_\sigma - \tilde{x}\| \leq K\|d_\sigma\|\,.$$

This implies that

$$\|F(m_\sigma)\| \leq K\|d_\sigma\| \tag{2.1}$$

is a root condition. That is, if $\|F(m_\sigma)\| > K\|d_\sigma\|$, then σ contains no roots and may be excluded.

We note that K could be global or could depend on the cell σ. Also, K is oftentimes a vector or a matrix of Lipschitz constants.

Given a function, $f(x)$, in one variable, the maximum value of $|f'(x)|$ can be used as a Lipschitz constant. Let us examine a polynomial in one variable, $p(x)$.

All such polynomials are Lipschitz on closed intervals. We note we can determine a better Lipschitz constant for a specific cell σ by doing a Taylor expansion about the midpoint of a cell.

Let us consider the behavior of $p(x)$ on $\sigma = [e, f]$, and let us assume without loss of generality that $|f| > |e|$. If we calculate K from the given polynomial, we get

$$p(x) = \Sigma_{i=0}^{N} a_i x^i$$

$$p'(x) = \Sigma_{i=1}^{N} i * a_i x^{i-1}$$

$$|p'(x)| \leq \Sigma_{i=1}^{N} i * |a_i (f)^{i-1}| = K .$$

On the other hand we may expand $p(x)$ about the midpoint of σ, m_σ, to obtain K,

$$p(x) = \Sigma_{i=0}^{N} b_i (x - m_\sigma)^i$$

$$p'(x) = \Sigma_{i=1}^{N} i * b_i (x - m_\sigma)^{i-1}$$

$$|p'(x)| \leq \Sigma_{i=1}^{N} i * |b_i * (f - e)/2|^{i-1} = K .$$

We notice that if e and f are close together, as they will be as we do successive refinements, the second estimate for K is much better.

Theorem 2.0.2 (Convergence of Cells to Roots). *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the Lipschitz root condition* (2.1)*, and let $N_F(\Lambda)$ be the set of solutions to the system. Then $N_F(\Lambda) \neq \emptyset$ if and only if the CEA does not terminate in a finite number of steps. In this case, $N_F(\Lambda) = \tilde{\Omega}$ where $\tilde{\Omega} = \lim_{n \to \infty} \Omega_n$.*

Proof. It suffices to show that whenever the algorithm does not terminate in a finite number of steps, then $N_F(\Lambda) \neq \emptyset$ and $N_F(\Lambda) = \tilde{\Omega}$.

Because we have a valid root condition, $N_F(\Lambda) \subseteq \tilde{\Omega}$ for any $n \geq 0$.

Now we must show that every element of $\tilde{\Omega}$ is also a zero, i.e., $\tilde{\Omega} \subseteq N_F(\Lambda)$. Since the algorithm does not terminate in a finite number of steps, there are cells in Ω_n for each n. Thus, $\tilde{\Omega} \neq \emptyset$.

For any fixed $\tilde{x} \in \tilde{\Omega}$, by Theorem 1.4.1, there is a sequence of cells $\{\sigma_n\}$, $\sigma_n \in \Omega_n$, such that $\tilde{x} \in \sigma_n$ for any $n \geq 0$. Since each cell σ_n must satisfy the Lipschitz root condition (2.1), we have $\|F(m_{\sigma_n})\| \leq K\|d_n\|$.

This yields the following result:

$$\|F(\tilde{x})\| \leq \|F(m_{\sigma_n})\| + \|F(\tilde{x}) - F(m_{\sigma_n})\| \leq K\|d_n\| + K\|d_n\| \to 0$$

$$\text{as} \quad n \to \infty .$$

Thus, $\tilde{x} \in \tilde{\Omega} \implies \tilde{x} \in N_F(\Lambda)$, so $\tilde{\Omega} \subseteq N_F(\Lambda)$. Therefore, $\tilde{\Omega} = N_F(\Lambda)$, which proves the claim. □

Corollary 2.0.1. *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the Lipschitz root condition* (2.1)*, and let $N_F(\Lambda)$ be the set of solutions to the system. If F is continuous on Λ, F is $\mathcal{C}^1$ on a neighborhood of $N_F(\Lambda)$, and $N_F(\Lambda)$ consists of a finite number of regular solutions of the system (namely, $F'(\tilde{x})$ is invertible for any $\tilde{x} \in N_F(\Lambda)$), then there is a constant N_0 such that $\#(\Omega_n) \leq N_0$ for any $n \geq 0$.*

Proof. First, for any $\tilde{x} \in N_F(\Lambda)$, we may expand F in a first order Taylor expansion about $\tilde{x}$,

$$F(x) = F'(\tilde{x})(x - \tilde{x}) + o(\|x - \tilde{x}\|) .$$

Taking norms of both sides and using the fact that

$$\|F'(\tilde{x})(x - \tilde{x})\| \geq \|F'(\tilde{x})^{-1}\|^{-1}\|x - \tilde{x}\|,$$

we obtain that

$$\|F(x)\| \geq \|F'(\tilde{x})^{-1}\|^{-1}\|x - \tilde{x}\| + o(\|x - \tilde{x}\|) .$$

Due to the regularity at $\tilde{x}$ there exists a $K > 0$ such that $\|F'(\tilde{x})^{-1}\|^{-1} = K$ and hence

$$\|F(x)\| \geq K\|x - \tilde{x}\| + o(\|x - \tilde{x}\|)\,.$$

We may absorb the asymptotic term into a constant. Hence, for $0 < C < \|F'(\tilde{x})\|$, there is an $\eta > 0$ such that

$$\|F(x)\| \geq C\|x - \tilde{x}\| \qquad \text{for} \qquad \|x - \tilde{x}\| \leq \eta\,.$$

Thus, if we look at $x = m_\sigma$ for some $\sigma \in \Omega_n$, we have

$$\|F(m_\sigma)\| \geq C\|m_\sigma - \tilde{x}\| \qquad \text{for} \qquad \|m_\sigma - \tilde{x}\| \leq \eta\,. \tag{2.2}$$

Similarly, for each $\tilde{x} \in N_F(\Lambda)$, we may find such a C and η. Since $N_F(\Lambda)$ contains a finite number of zeros, we may simply choose the minimum of such η and C so that the above inequality holds for all cells σ such that $\|m_\sigma - \tilde{x}\| \leq \eta$ for some $\tilde{x} \in N_F(\Lambda)$.

Also, by the Lipschitz root condition for $\sigma \in \Omega_n$, there holds

$$\|F(m_\sigma)\| \leq K\|d_n\|\,. \tag{2.3}$$

Combining the two inequalities (2.2) and (2.3), we get

$$\|m_\sigma - \tilde{x}\| \leq K\|d_n\|/C \qquad \text{for} \qquad \|m_\sigma - \tilde{x}\| \leq \eta\,.$$

Now we will consider the case where $\|m_\sigma - \tilde{x}\| > \eta$ for all $\tilde{x} \in N_F(\Lambda)$ and prove that such a cell will eventually fail the root condition and be discarded.

Let

$$S = \Lambda \setminus \cup B(N_F(\Lambda), \eta)\,.$$

Since F is continuous on the compact set S, $\|F(x)\|$ attains a minimum, $\kappa > 0$, on S.

Thus, $\|F(x)\| \geq \kappa$ for all $x \in S$.

Now consider $m_\sigma \in S$. In order to satisfy the root condition (2.1), we must have

$$\|F(m_\sigma)\| \leq K\|d_n\|$$

for all subdivisions (Ω_n) with mesh size $\|d_n\|$. If we take $\|d_n\|$ sufficiently small, we will have $K\|d_n\| < \kappa \leq \|F(m_\sigma)\|$. Thus, for sufficiently large n, any cell with midpoint in S will be discarded.

Hence, for sufficiently large n, the volume of a ball around a root $\tilde{x}$ which may contain a cell in $\mathbb{R}^N$ which satisfies the root condition is at most $((K_\sigma/C + 1)2\|d_n\|)^N$. The volume of a given cell within this ball is approximately $(2\|d_n\|)^N$. Thus, the total possible cells which could fit inside this ball is approximately

$$((K_\sigma/C + 1)2\|d_n\|)^N \div (2\|d_n\|)^N = (K_\sigma/C + 1)^N .$$

The largest possible number of cells which could fit inside the η-ball around a root would be $((K_\sigma/C+1)2\|d_n\|)^N \div v(\Gamma_n)$, where $v(\Gamma_n)$ is the N-dimensional "volume" of cells in Γ_n. In applications, we will be able to relate $v(\Gamma_n)$ and $(2\|d_n\|)^N$ by some constant ρ. As was mentioned in Theorem 1.3.1, this will be possible since we will not be dealing with long, skinny cells.

Let

$$\rho = \max_n \frac{(2\|d_n\|)^N}{v(\Gamma_n)} .$$

Since we are dealing with a finite number of roots J, the total number of cells will be bounded by $\rho * (K_\sigma/C + 1)^N * J = N_0$. □

2.1 Numerical Results

In this section examples are presented for which the Lipschitz root condition (2.1) was used.

Here are two definitions which are needed for examining the numerical results in this dissertation:

Definition 2.1.1 (Sequential Bisection). *Sequential bisection is the partitioning which takes place when cells are bisected along one axis at a time.*

Definition 2.1.2 (Simultaneous Bisection). *Simultaneous bisection is the partitioning which takes place when cells are bisected along all axes at the same time.*

For all of the root conditions we consider, it generally occurs that cell exclusion algorithms yield significantly more cells than there are roots. This is not surprising, since in general, a cell exclusion algorithm will not only produce the cell containing a root, but neighboring cells as well. The number of neighboring cells increases exponentially with the dimension of the problem. However, we found linking cells which are close together into connected components to be an effective means of isolating zeros. (See the Java code in the appendix.) After sufficient partitioning, each component corresponds to an isolated zero. Then if the mesh size of a component is small enough, we may stop. Otherwise, we can use the midpoint of the component as a starting point for Newton's method.

1. This problem comes from the 1996 paper by Xu [20]. We used the Lipschitz root condition for this problem with a vector of Lipschitz constants.

   The domain of this problem is taken as $\Lambda = [-1, 2] \times [-20, 5]$, where $F(x)$ is defined by the following:

$$f_1(x_1, x_2) = 1/2 \sin(x_1 x_2) - x_2/4\pi - x_1/2$$
$$f_2(x_1, x_2) = (1 - 1/4\pi)(e^{2x_1} - e) + e x_2/\pi - 2e x_1$$

   We used the same Lipschitz constants, (11.6, 13), for this problem which are used in [20]. This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the root condition and $2\|d\| <$ tolerance, then σ is in Ω.

   We see that for both types of bisection the number of cells in Ω is bounded by $N_0 \approx 900$.

Tolerance	Cells in Ω: Sequential Bisection	Cells in Ω: Simultaneous Bisection	R.C. Checks: Sequential Bisection	R.C. Checks: Simultaneous Bisection
.1	742	779	6,653	6,136
.01	843	898	20,975	20,668
.001	847	896	31,239	31,404
.0001	839	897	41,517	42,200
.00001	844	898	55,243	56,584

Solutions
(-0.26059929, 0.6225309)
(0.29944869, 2.8369278)
(0.5, π)
(1.66342198, -16.2827906)
(1.65458272, -15.8191882)
(1.60457055, -13.3629017)
(1.57822540, -12.1766898)
(1.53050532, -10.2022479)
(1.48131957, -8.3836127)
(1.43394933, -6.8207653)
(1.33742561, -4.1404386)
(1.29436046, -3.1372198)

Note that the number of cells for simultaneous and sequential bisection are different. This is due to the fact that this problem does not have a "square" domain.

It is also interesting to observe that the number of root condition checks for sequential bisection and simultaneous bisection are nearly identical for this problem. This can be explained by the fact that as the algorithm progresses more than half of the cells will be discarded.

Let us consider one cell of which at least half will be discarded. When using simultaneous bisection, this cell will be partitioned immediately into four cells and four root condition checks will be performed. When using sequential bisection, this cell will be bisected along the first axis into two cells, two root condition checks will be performed, and typically one cell will be discarded. The remaining cell will be bisected along the second axis and two more root condition checks will be performed. This makes a total of four root condition checks, which is the same number that takes place when using simultaneous bisection.

2. This problem comes from Xu's 1997 paper [19]. The domain of this problem is taken as $\Lambda = [-4, 4] \times [-4, 4]$, where $F(x)$ is defined by the following:

$$f_1(x_1, x_2) = x_1 - 4x_2^2$$

$$f_2(x_1, x_2) = \cos(x_1) - x_2$$

For this problem we used a matrix of Lipschitz constants instead of a vector of Lipschitz constants. We observe that this leads to a tighter root condition, so we obtain fewer cells than Xu using a global root condition. We used the Lipschitz matrix:

$$\begin{bmatrix} 1 & 32 \\ 1 & 1 \end{bmatrix}$$

This problem was numerically solved using a C^{++} implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 60$.

Levels	**Cells in Ω Global**	**Xu 1997 Global**	**Xu 1997 Local**
5	47	~ 135	~ 15
10	55	~ 110	~ 15
15	56	~ 110	~ 15
20	55	~ 110	~ 15
25	55	~ 110	~ 15
30	56	~ 110	~ 15

Solutions
(2.476468, -.786840)
(3.502147, -.935701)
(1.036674, .509086)

Chapter 3

A MONOTONICITY ROOT CONDITION

In this chapter we present the monotonicity root condition which is introduced in [20]. First, we must define what it means for a mapping F to be monotonically decomposable. Assume that we have two vectors

$$x = (x_1, x_2, x_3, \ldots, x_n) \quad \text{and} \quad y = (y_1, y_2, y_3, \ldots, y_n) \quad \text{in } \mathbb{R}^N .$$

We define $x \leq y$ iff $x_i \leq y_i$ for all $i \in \{1, 2, \ldots, N\}$.

Definition 3.0.3 (Isotone). *A mapping $G : \mathbb{R}^N \to \mathbb{R}^N$ is said to be isotone if $G(x) \leq G(y)$ whenever $x \leq y$.*

Definition 3.0.4 (Monotonically Decomposable). *The map F is said to be monotonically decomposable if there are two isotone mappings G and H such that $F = G - H$.*

Given a cell

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

where $a_{ij} \in \mathbb{R}$, $a_{i1} < a_{i2}$, we may define the "lowermost" and "uppermost" corners, $\underline{\sigma}$ and $\overline{\sigma}$ respectively, of the cell as follows:

$$\text{"lowermost corner"} = \underline{\sigma} = (a_{11}, a_{21}, \ldots, a_{N1})$$

$$\text{"uppermost corner"} = \overline{\sigma} = (a_{12}, a_{22}, \ldots, a_{N2}) .$$

If F is a monotonically decomposable mapping, and σ contains a solution, say $\tilde{x}$, to the system $F(x) = 0$, then we can arrive at a root condition as follows:

We know $G(\tilde{x}) - H(\tilde{x}) = F(\tilde{x}) = 0$. By the isotone property of G and H,

$$G(\underline{\sigma}) - H(\overline{\sigma}) \leq 0 = F(\tilde{x}) \leq G(\overline{\sigma}) - H(\underline{\sigma}) .$$

From these equations, we obtain the root condition:

$$G(\underline{\sigma}) \leq H(\overline{\sigma}) \text{ and } H(\underline{\sigma}) \leq G(\overline{\sigma}) . \tag{3.1}$$

Notice that all polynomial systems are monotonically decomposable. If one is considering a positive domain Λ, we may decompose the polynomials according to terms having positive and negative coefficients (see Example 3.0.1). If one is considering a negative domain, one can expand the polynomials about the lowermost corner of the cell under consideration. Then one can decompose the polynomials according to the positive and negative coefficients of this expansion (see Example 3.0.2).

Example 3.0.1. *Consider the following polynomial with* $\Lambda = [0, 3]$*:*

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

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

Example 3.0.2. *Consider the following system of polynomials,* $F(x)$*, with* $\Lambda = [-3, 3] \times [-3, 3]$*:*

$$f_1(x_1, x_2) = (x_1 + 1)x_2$$

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

Expanding $F(x)$ about (-3,-3),

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

Thus, we may take $G(x)$ to be $g_1(x_1, x_2) = (x_1 + 3)(x_2 + 3) + 6$, $g_2(x_1, x_2) = (x_2+3)^2+8$ and $H(x)$ to be $h_1(x_1, x_2) = 3(x_1+3)+2(x_2+3)$, $h_2(x_1, x_2) = 6(x_2+3)$.

We see $F(x)$ is monotonically decomposable on Λ with $F(x) = G(x) - H(x)$.

Theorem 3.0.1 (Convergence of Cells to Roots). *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the monotonicity root condition* (3.1)*, and let $N_F(\Lambda)$ be the set of solutions to the system. If $F = G - H$ is monotonically decomposable with G and H continuous, then $N_F(\Lambda) \neq \emptyset$ if and only if the CEA does not terminate in a finite number of steps. In the latter case, $N_F(\Lambda) = \tilde{\Omega}$ where $\tilde{\Omega} = \lim_{n\to\infty} \Omega_n$.*

Proof. It suffices to show that whenever the algorithm does not terminate in a finite number of steps, then $N_F(\Lambda) \neq \emptyset$ and $N_F(\Lambda) = \tilde{\Omega}$.

Because we have a valid root condition, $N_F(\Lambda) \subseteq \tilde{\Omega}$ for any $n \geq 0$.

Now we must show that every element of $\tilde{\Omega}$ is also a zero, i.e., $\tilde{\Omega} \subseteq N_F(\Lambda)$. Since the algorithm does not terminate in a finite number of steps, there are cells in Ω_n for each n. Thus, $\tilde{\Omega} \neq \emptyset$.

For any fixed $\tilde{x} \in \tilde{\Omega}$, by Theorem 1.4.1, there is a sequence of cells $\{\sigma_n\}$, $\sigma_n \in \Omega_n$, such that $\tilde{x} \in \sigma_n$ for any $n \geq 0$.

Since each cell σ_n must satisfy the monotonicity root condition (3.1), we have

$$\|(G + H)(\overline{\sigma_n}) - (G + H)(\underline{\sigma_n})\|$$
$$= \|G(\overline{\sigma_n}) - H(\underline{\sigma_n})\| + \|H(\overline{\sigma_n}) - G(\underline{\sigma_n})\| .$$

The mapping $G + H$ is uniformly continuous on σ_n. This implies that

$$\|(G + H)(\overline{\sigma_n}) - (G + H)(\underline{\sigma_n})\| \to 0 \text{ as } \|d_n\| \to 0 .$$

Thus, from the above equation it follows that

$$\|G(\overline{\sigma_n}) - H(\underline{\sigma_n})\| \to 0 \text{ and } \|H(\overline{\sigma_n}) - G(\underline{\sigma_n})\| \to 0 \text{ as } n \to \infty . \tag{3.2}$$

Now we can show $\tilde{x} \in N_F(\Lambda)$.

If $F(\tilde{x}) = G(\tilde{x}) - H(\tilde{x}) \geq 0$, then

$$0 \leq G(\tilde{x}) - H(\tilde{x}) \leq G(\overline{\sigma_n}) - H(\underline{\sigma_n}) .$$

Thus, in this case it follows from (3.2) that,

$$0 \leq \|G(\tilde{x}) - H(\tilde{x})\| \leq \|G(\overline{\sigma_n}) - H(\underline{\sigma_n})\| \to 0 \text{ as } n \to \infty .$$

If $F(\tilde{x}) = G(\tilde{x}) - H(\tilde{x}) \leq 0$, then

$$0 \leq H(\tilde{x}) - G(\tilde{x}) \leq H(\overline{\sigma_n}) - G(\underline{\sigma_n}) .$$

Thus, in this case it follows from (3.2) that,

$$0 \leq \|G(\tilde{x}) - H(\tilde{x})\| \leq \|H(\overline{\sigma_n}) - G(\underline{\sigma_n})\| \to 0 \text{ as } n \to \infty .$$

Thus, $\tilde{x} \in \tilde{\Omega} \implies \tilde{x} \in N_F(\Lambda)$, so $\tilde{\Omega} \subseteq N_F(\Lambda)$. Therefore, $\tilde{\Omega} = N_F(\Lambda)$, which proves the claim. □

Corollary 3.0.1. *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the monotonicity root condition* (3.1)*, and let $N_F(\Lambda)$ be the set of solutions to the system. If G and H are C^2 on Λ and $N_F(\Lambda)$ consists of a finite number of regular solutions of the system (namely, $\|F'(\tilde{x})\|$ is invertible for any $\tilde{x} \in N_F(\Omega)$), then there is a constant N_0 such that $\#(\Omega_n) \leq N_0$ for any $n \geq 0$.*

Proof. Since G, H are C^2, $G + H$ is also Lipschitz. That is, there is a constant K such that

$$\|(G + H)(x) - (G + H)(y)\| \leq K\|x - y\| \qquad \forall\ x, y \in \Lambda\,.$$

Now consider any cell $\sigma \in \Omega_n$ with $\underline{\sigma} \leq m_\sigma \leq \overline{\sigma}$. Two cases are possible: either $F(m_\sigma) = G(m_\sigma) - H(m_\sigma) \geq 0$ or $F(m_\sigma) = G(m_\sigma) - H(m_\sigma) \leq 0$.

In the first case by monotonicity, $0 \leq G(m_\sigma) - H(m_\sigma) \leq G(\overline{\sigma}) - H(\underline{\sigma})$.

Since by the monotonicity root condition (3.1), $H(\overline{\sigma}) - G(\underline{\sigma}) \geq 0$, we have

$$\begin{aligned} 0 \leq |F(m_\sigma)| = |G(m_\sigma) - H(m_\sigma)| &\leq |G(\overline{\sigma}) - H(\underline{\sigma}) + H(\overline{\sigma}) - G(\underline{\sigma})| \\ &= |(G + H)(\overline{\sigma}) - (G + H)(\underline{\sigma})|\,. \end{aligned} \tag{3.3}$$

In the second case $F(m_\sigma) = G(m_\sigma) - H(m_\sigma) \leq 0$ which implies by monotonicity, $0 \leq H(m_\sigma) - G(m_\sigma) \leq H(\overline{\sigma}) - G(\underline{\sigma})$.

Since by the monotonicity root condition (3.1), $G(\overline{\sigma}) - H(\underline{\sigma}) \geq 0$, we have

$$\begin{aligned} 0 \leq |F(m_\sigma)| = |H(m_\sigma) - G(m_\sigma)| &\leq |H(\overline{\sigma}) - G(\underline{\sigma}) + G(\overline{\sigma}) - H(\underline{\sigma})| \\ &= |(G + H)(\overline{\sigma}) - (G + H)(\underline{\sigma})|\,. \end{aligned} \tag{3.4}$$

Thus, in both cases (3.3) and (3.4) we conclude

$$\begin{aligned} \|F(m_\sigma)\| &\leq \|(G + H)(\overline{\sigma}) - (G + H)(\underline{\sigma})\| \\ &\leq K\|\overline{\sigma} - \underline{\sigma}\| \leq 2K\|d_n\|\,. \end{aligned}$$

This proof may now be finished in the same manner as that of Corollary 2.0.1. □

3.1 Numerical Results

In this section examples are presented for which the monotonicity root condition (3.1) was used.

1. This is an equation in one variable. The domain of this problem is taken as $\Lambda = [0, 5]$:

$$f(x) = (x - 2)(x - 1)$$

In one dimension the sequential bisection and simultaneous bisection algorithms should perform identically. We notice that there is one more root condition check for the sequential bisection than the simultaneous bisection. The reason for this is that the sequential bisection algorithm checks the initial cell and the simultaneous bisection algorithm does not.

This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the root condition and $2\|d\| <$ tolerance, then σ is in Ω.

We see that the number of cells in Ω is bounded by $N_0 \approx 15$.

Tolerance	**Cells in Ω**	**R.C. Checks: Sequential Bisection**	**R.C.Checks: Simultaneous Bisection**
.1	15	57	56
.01	12	137	136
.001	12	233	232
.0001	12	305	304
.00001	12	377	376

2. This problem comes from the 1996 paper by Xu [20].

The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{4} [0, 10]$$

where $F(x)$ is defined by the following:

$$f_1(x) = x_1^3 + x_2^3 - x_3^2 - x_4^2 + x_1 x_2 - 6$$

$$f_2(x) = x_1 x_2 (1 + x_1 + x_2) - x_3 x_4 - 6$$

$$f_3(x) = -x_3^3 - x_4^3 + x_3 x_4 + x_1 + x_2 + 4$$

$$f_4(x) = x_1^3 + x_2^3 + x_3 x_4 - x_3 + x_4 - 8$$

This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the root condition and $2\|d\| <$ tolerance, then σ is in Ω.

We see that for bisection the number of cells in Ω is bounded by $N_0 \approx 62$ and for trisection the number of cells in Ω is bounded by $N_0 \approx 56$.

Tolerance	Cells in Ω: Bisection	Cells in Ω: Trisection	R.C. Checks: Sequential Bisection	R.C. Checks: Simultaneous Bisection
.1	56	52	1,863	3,456
.01	58	54	3,023	6,208
.001	62	54	4,901	10,272
.0001	62	56	6,153	13,088
.00001	50	48	7,287	15,680

Midpoints of selected cells were taken as the starting points for Newton's method.

Solutions Obtained After Application of Newton's Method
(.921697994035985, 1.96746594981475, 1.98252447605658, .530962690049853)
(1.96746594981475, .921697994035985, 1.98252447605658, .530962690049853)

3. The following problem is taken from the 1996 paper by Xu [20]. The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{4} [0, 10]$$

where $F(x)$ is defined by the following:

$$f_1(x) = x_1^2 + x_2^2 - x_3^2 - x_4^2 + 5$$

$$f_2(x) = x_1 x_2 - x_3 x_4 + 1$$

$$f_3(x) = -x_3^3 - x_4^3 + x_1 + x_2 + 25$$

$$f_4(x) = x_1^3 + x_2^3 - x_3 - x_4 - 5$$

This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the root condition and $2\|d\| <$ tolerance, then σ is in Ω.

We see that for bisection the number of cells in Ω is bounded by $N_0 \approx 1,712$ and for trisection the number of cells in Ω is bounded by $N_0 \approx 1,500$.

Tolerance	**Cells in Ω: Bisection**	**Cells in Ω: Trisection**	**R.C. Checks: Sequential Bisection**	**R.C. Checks: Simultaneous Bisection**
.1	348	702	4,275	5,552
.01	1,712	1,170	31,945	44,880
.001	1,328	1,368	88,329	139,024
.0001	1,384	1,484	127,773	204,560
.00001	1,460	1,433	169,417	272,912

Midpoints of selected cells were taken as the starting points for Newton's method.

Solutions Obtained After Application of Newton's Method
(.00315181855733726, 2.02755457627325, .335492718717397, 2.99973867685411)
(2.02755457627325, .00315181855733726, .335492718717397, 2.99973867685411)
(.00315181855733726, 2.02755457627325, 2.99973867685411, .335492718717397)
(2.02755457627325, .00315181855733726, 2.99973867685411, .335492718717397)
(1,2,1,3)
(2,1,1,3)
(1,2,3,1)
(2,1,3,1)

4. The domain of the following problem is taken as

$$\Lambda = \prod_{i=1}^{5}[0,1]$$

where $F(x)$ is defined by the following:

$$f_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 1$$

$$f_2(x) = -x_1 + x_2^2 + x_3^2 + x_4^2 + x_5^2$$

$$f_3(x) = x_1^2 - x_2 + x_3^2 + x_4^2 + x_5^2$$

$$f_4(x) = x_3^4 - x_5^2$$

$$f_5(x) = x_4^2 - x_5^2$$

This problem was numerically solved using a Matlab implementation. The measure of tolerance is given by the cell size. Consider a cell σ with mesh vector d. If σ satisfies the root condition and $2\|d\| <$ tolerance, then σ is in Ω.

We see that the number of cells in Ω is bounded by $N_0 \approx 48$.

Tolerance	**Cells in Ω**	**R.C. Checks: Sequential Bisection**	**R.C. Checks: Simultaneous Bisection**
.1	45	1,495	3,584
.01	48	3,153	8,128
.001	48	4,883	12,672
.0001	48	7,161	18,944
.00001	45	8,869	23,648

The following table lists the 8 solutions to this system. The first one lies in the initial domain Λ and the others may be found by symmetry.

Solutions
(.618034, .618034, .418202, .174893, .174893)
(.618034, .618034, -.418202, .174893, .174893)
(.618034, .618034, .418202, -.174893, .174893)
(.618034, .618034, .418202, .174893, -.174893)
(.618034, .618034, -.418202, -.174893, .174893)
(.618034, .618034, -.418202, .174893, -.174893)
(.618034, .618034, .418202, -.174893, -.174893)
(.618034, .618034, -.418202, -.174893, -.174893)

5. Consider the equation in one variable:

$$f(x) = (x-3)(x-2)(x-1)x$$

We use 6 as a Lipschitz constant for $\Lambda = [0, 3]$ and 50 as a Lipschitz constant for $\Lambda = [0, 4]$. The numbers of cells satisfying the root conditions are given. In particular, we note that the zero lying on the boundary of Λ is found. We see the number of cells stabilizes for both root conditions and both domains. This problem was numerically solved using a C^{++} implementation.

Levels	**Lipschitz** $\Lambda = [0, 3]$	**Lipschitz** $\Lambda = [0, 4]$	**Monotonicity** $\Lambda = [0, 3]$	**Monotonicity** $\Lambda = [0, 4]$
5	8	26	28	30
10	8	62	137	164
15	8	62	137	160
20	8	62	132	160
25	8	62	132	160
30	8	62	132	160

6. Consider the equation in one variable with a root of multiplicity 2:

$$f(x) = (x-3)(x-2)(x-1)x^2$$

We use 18 as a Lipschitz constant for $\Lambda = [0,3]$ and 224 as a Lipschitz constant for $\Lambda = [0,4]$. The numbers of cells satisfying the root conditions are given.

This problem was numerically solved using a C^{++} implementation.

Levels	**Lipschitz** $\Lambda = [0,3]$	**Lipschitz** $\Lambda = [0,4]$	**Monotonicity** $\Lambda = [0,3]$	**Monotonicity** $\Lambda = [0,4]$
5	23	27	32	32
10	39	315	195	236
15	144	590	184	222
20	740	2,409	184	222
25	4,112	12.711	184	222
30	23,187	70,984	184	222

We notice that when using the Lipschitz root condition (2.1) for this equation with a multiple root, the number of cells explodes. Hence, we see the importance of regularity in controlling the number of cells.

Chapter 4

TAYLOR SERIES ROOT CONDITIONS

In this chapter we investigate root conditions which make use of Taylor series. One advantage of these root conditions is that they are localized. Unlike the Lipschitz root condition (2.1) which involves the same dominating value, $K\|d_n\|$, for all cells in Ω_n, these root conditions involve dominating functions which depend upon the cell, σ, which is under consideration.

When working with Taylor series in multi-dimensions, we will be using multi-indices. We introduce some standard notation for dealing with multi-indices. Let $\alpha \in \mathbb{Z}_+^n$ be a multi-index:

1. The length of α is defined by $|\alpha| := \sum_i \alpha_i$
2. The factorial of α is defined by $\alpha! := \prod_i \alpha_i!$
3. If $x \in \mathbb{R}^N$, then we define $x^\alpha := \prod_i x_i^{\alpha_i}$
4. We define the partial derivatives $\partial^\alpha := (\alpha!)^{-1} \prod_i \partial_i^{\alpha_i}$

4.1 A Taylor Series Root Condition

In this section we present a Taylor Series root condition which was developed in [19]. Let $f : \mathbb{R}^N \to \mathbb{R}$ be a function which can be expressed as a formal series

$$f(x) = \sum_{|\alpha|=0}^{\infty} c_\alpha x^\alpha \qquad \text{where } c_\alpha \in \mathbb{R}$$

Definition 4.1.1 (Absolute Value). *We define the absolute value of f as:*

$$A(f)(x) = \sum_{|\alpha|=0}^{\infty} |c_\alpha| x^\alpha.$$

Definition 4.1.2 ($\prec\prec$). *We say that $f \prec\prec g$ if and only if*

$$f(x) = \sum_{|\alpha|=0}^{\infty} c_\alpha x^\alpha \qquad g(x) = \sum_{|\alpha|=0}^{\infty} b_\alpha x^\alpha$$

and $b_\alpha \geq |c_\alpha|$ for all α.

The following theorem is proved in [19]. A much simpler proof than that in [19] will be given in Theorem 4.2.1 .

Theorem 4.1.1. *Let σ be a cell with midpoint m and mesh vector d. If $F = (f_1, f_2, \ldots, f_N) : \mathbb{R}^N \to \mathbb{R}^N$ and $G = (g_1, g_2, \ldots, g_N) : \mathbb{R}^N \to \mathbb{R}^N$ are such that $g_i \succ\succ f_i$ for $i = 1, 2, \ldots, N$, then*

$$|f_i(m)| \leq g_i(|m| + d) - g_i(|m|) \tag{4.1}$$

for $i = 1, 2, \ldots, N$ is a root condition for σ.

Let us note that the exclusion test may be stopped as soon as the root condition (4.1) fails to hold for any i.

The following propositions will be used help construct the system G of the previous theorem. They are both found in [19]. The main tool used in proving the first proposition is the triangle inequality.

Proposition 4.1.1. *If f_j is a formal series for $j = 1, 2, \ldots, k$, $F = (f_1, f_2, \ldots, f_k)$, and $A(F) = (A(f_1), A(f_2), \ldots, A(f_k))$, then*

1. $\sum_{j=1}^{k} |c_j| A(f_j) \succ\succ A(\sum_{j=1}^{k} c_j f_j)$.
2. $(A(F))^\alpha \succ\succ A(F^\alpha)$, *where* $F^\alpha = (f_1)^{\alpha_1}(f_2)^{\alpha_2} \ldots (f_k)^{\alpha_k}$.

Here are two examples that illustrate this proposition:

Example 4.1.1. *Let* $f_1 = x^2 - x - 3$, $f_2 = x^2 - 4x - 5$, $c_1 = -1$, *and* $c_2 = 2$.

$$\sum_{j=1}^{2} |c_j| A(f_j) = 3x^2 + 9x + 13$$

$$A(\sum_{j=1}^{2} c_j f_j) = x^2 + 7x + 7$$

We see

$$\sum_{j=1}^{2} |c_j| A(f_j) \succ\succ A(\sum_{j=1}^{2} c_j f_j).$$

Example 4.1.2. *Let* $f_1 = x - 1$, $f_2 = x + 2$, *and* $\alpha = 2 + 1$.

$$\begin{aligned} (A(F))^{\alpha} &= (x+1)^2(x+2) \\ &= x^3 + 4x^2 + 5x + 2 \\ A(F^{\alpha}) &= A((x-1)^2(x+2)) \\ &= x^3 + 3x + 2 \end{aligned}$$

Here we see

$$(A(F))^{\alpha} \succ\succ A(F^{\alpha}).$$

Proposition 4.1.2 (Composition Rule). *Let* $f = h(Q)$ *where* $h : \mathbb{R}^M \to \mathbb{R}$ *and* $Q = (q_1, \ldots, q_M) : \mathbb{R}^N \to \mathbb{R}^M$ *with each* q_i *being a series. If there exists a series* $j : \mathbb{R}^M \to \mathbb{R}$ *and* $W = (w_1, \ldots, w_M) : \mathbb{R}^N \to \mathbb{R}^M$ *with each* w_i *being a series such that*

$$j \succ\succ A(h) \qquad \text{and} \qquad w_i \succ\succ A(q_i)$$

for $i = 1, 2, \ldots M$, *then*

$$g = j(W) \succ\succ A(f).$$

This idea of building functions which dominate other functions in the $\prec\prec$ sense to arrive at a root condition is very useful. In the next section this idea will be further refined.

4.2 An Improved Taylor Series Root Condition

The method proposed by Xu in [19] is often far less than optimal. In this section we develop an improved Taylor series root condition which generally discards more cells.

We begin with some introductory definitions:

Definition 4.2.1 (A_k). *We define A_k to be the space of continuous functions $f : \mathbb{R}^N \to \mathbb{R}$ such that $\partial^\alpha f$ is absolutely continuous for $|\alpha| < k$.*

Recalling the multi-index definition of the partial derivative, we note that for $f \in A_k$ the following Taylor formula with integral remainder holds:

$$f(m+h) = f(m) + \sum_{0<|\alpha|<k} \partial^\alpha f(m) h^\alpha + \int_0^1 \sum_{|\beta|=k} \partial^\beta f(m+th) w_k(dt) h^\beta$$

where $w_k(dt) = k(1-t)^{k-1}(dt)$ is a probability measure on the interval $[0,1]$.

Definition 4.2.2 (Cone). *In A_k we introduce the cone K_k of functions such that $g \in K_k$ if and only if $\partial^\alpha g(x) \leq \partial^\alpha g(y)$ for $0 \leq x \leq y$ and $|\alpha| \leq k$.*

Notice that any polynomial with positive coefficients will lie in such a cone. The following definition will be useful in the development of the root condition.

Definition 4.2.3 (Order of Domination). *We write $f \prec_k g$ and say that g dominates f with order k if and only if $f \in A_k$, $g \in K_k$, and*

$$|\partial^\alpha f(x)| \leq (\partial^\alpha g)(|x|)$$

for all $x \in \mathbb{R}^N$ and $|\alpha| \leq k$. If $f \prec_k g$ for all $k \geq 0$, we write $f \prec_\infty g$.

Note that by definition, $f \prec_k g \Longrightarrow f \prec_l g$ for all $0 \leq l \leq k$.

Theorem 4.2.1. *Given two power series* $f(x) = \sum_{|\alpha|=0}^{\infty} c_\alpha x^\alpha$ *and* $g(x) = \sum_{|\alpha|=0}^{\infty} d_\alpha x^\alpha$, *then*

$$f \prec_\infty g \Longleftrightarrow f \prec\prec g.$$

Proof. If $f \prec_\infty g$, then $|c_\alpha| = |\partial^\alpha f(0)| \leq \partial^\alpha g(0) = d_\alpha$ for all α. Hence, $f \prec\prec g$.

On the other hand, assume $f \prec\prec g$. For technical reasons we introduce the monomial $\xi^\alpha : x \longrightarrow x^\alpha$. Then for any fixed β, we estimate termwise

$$|\partial^\beta f(x)| \leq \sum_{|\alpha|=0}^{\infty} |c_\alpha||\partial^\beta \xi^\alpha(x)| \leq \sum_{|\alpha|=0}^{\infty} d_\alpha(\partial^\beta \xi^\alpha)(|x|) = \partial^\beta g(|x|).$$

Thus, $f \prec_\infty g$.

□

The following examples illustrate the difference between various orders of domination:

1. $\sin x \prec\prec \sinh x$, but $\sin x \prec_3 x + 1/6x^3$

2. $\cos x \prec\prec \cosh x$, but $\cos x \prec_1 1 + x$, $\cos x \prec_2 1 + 1/2x^2$,
   $\cos x \prec_4 1 + 1/2x^2 + 1/24x^4$

3. $\log(1 + x) \prec\prec -\log(1 - x)$ for $|x| < 1$, but
   $\log(1 + x) \prec_3 x + x^2/2 + x^3/3$ for $|x| < 1$

4. $e^x \prec_\infty e^x$
   Notice that we cannot bound e^x using any order less than infinity.

Several useful theorems follow which enable us to find a function g which suitably dominates another function f.

Theorem 4.2.2. *If* $f \prec_k g$, *then* $\lambda f \prec_k |\lambda| g$.

Theorem 4.2.3. *If* $f_i \prec_k g_i$, *then* $\sum_i f_i \prec_k \sum_i g_i$.

Theorem 4.2.4. *If* $f_i \prec_k g_i$, *then* $\prod_i f_i \prec_k \prod_i g_i$.

The preceding theorem may be proved using the product rule which leads to termwise domination.

Theorem 4.2.5. *Let* $f \prec_k g$ *and* $f_i \prec_k g_i$. *Now if we define the function compositions* $F = f(f_1, \ldots, f_n)$ *and* $G = g(g_1, \ldots, g_n)$, *then* $F \prec_k G$.

The preceding theorem may be proved using the chain rule to obtain termwise domination.

The following examples illustrate how the preceding theorems may be applied.

Example 4.2.1.

$$t^2 - 3 \prec_3 t^2 + 3$$

and

$$\cos(3t) \prec_3 1 + \frac{9t^2}{2} + \frac{27t^3}{6}$$

implies

$$t^2 - 3 + \cos(3t) \prec_3 t^2 + 3 + 1 + \frac{9t^2}{2} + \frac{27t^3}{6}$$

Example 4.2.2.

$$\cos y \prec_2 1 + \frac{y^2}{2}$$

and

$$y - \pi \prec_2 y + \pi$$

implies

$$(y - \pi)\cos y \prec_2 \left(1 + \frac{y^2}{2}\right)(y + \pi)$$

Example 4.2.3.

$$\frac{1}{1+t} \prec\prec \frac{1}{1-t} \qquad \textit{for } |t| < 1$$

and

$$\sin x \prec_3 x + \frac{1}{6}x^3$$

implies

$$\frac{1}{1+\frac{1}{2}\sin x} \prec_3 \frac{1}{1-\frac{1}{2}\left(x+\frac{1}{6}x^3\right)} \qquad \textit{for } |x+\tfrac{1}{6}x^3| < 2$$

Example 4.2.4.

$$\sin s \prec_3 s + \frac{1}{6}s^3$$

and

$$\cos t \prec_3 1 + \frac{1}{2}t^2 + \frac{1}{6}t^3$$

and

$$1 - xy \prec\prec 1 + xy$$

implies

$$\sin(\cos(1-xy)) \prec_3 \left(1+\frac{1}{2}(1+xy)^2+\frac{1}{6}(1+xy)^3\right)+\frac{1}{6}\left(1+\frac{1}{2}(1+xy)^2+\frac{1}{6}(1+xy)^3\right)^3$$

Example 4.2.5.

$$\log(1+x) \prec_3 x + \frac{x^2}{2} + \frac{x^3}{3} \qquad \textit{for } |x| < 1$$

and

$$\sin(2t) \prec_3 2t + \frac{(2t)^3}{6}$$

and

$$\frac{y^3}{6} - \frac{y^2}{2} + y - 1 \prec_3 \frac{y^3}{6} + \frac{y^2}{2} + y + 1$$

implies

$$\begin{aligned}\log(1+\sin(2(y^3/6-y^2/2+y-1))) \prec_3\ & 2(y^3/6+y^2/2+y+1)+\frac{4}{3}(y^3/6+y^2/2+y+1)^3\\ &+\frac{1}{2}(2(y^3/6+y^2/2+y+1)+\frac{4}{3}(y^3/6+y^2/2+y+1)^3)^2\\ &+\frac{1}{3}(2(y^3/6+y^2/2+y+1)+\frac{4}{3}(y^3/6+y^2/2+y+1)^3)^3\end{aligned}$$

Example 4.2.6.

$$x^3 + x^2 - 4x - 2 \prec_4 x^3 + x^2 + 4x + 2$$

and

$$\sin y \prec_4 y + \frac{y^3}{6} + \frac{y^4}{24}$$

implies

$$(\sin y)^3+(\sin y)^2-4(\sin y)-2 \prec_4 \left(y+\frac{y^3}{6}+\frac{y^4}{24}\right)^3+\left(y+\frac{y^3}{6}+\frac{y^4}{24}\right)^2+4\left(y+\frac{y^3}{6}+\frac{y^4}{24}\right)+2$$

Now we introduce the improved Taylor series root condition which uses the idea of orders of domination.

Theorem 4.2.6. *Let σ be a cell with midpoint m and mesh vector d. Let $f(x) \prec_k g(x)$ for $x \in \sigma$, and let $q \leq k$. We then obtain the root condition:*

$$|f(m)| \leq g(|m| + d) - g(|m|) - \sum_{0<|\alpha|<q} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|)d^\alpha. \tag{4.2}$$

We notice that the quantity following the summation is nonnegative since g dominates f with order k. Thus, whenever this quantity is positive we are subtracting off a positive number and obtain a tighter root condition.

Proof. Let us assume we have a cell $\sigma = [m + d, m - d]$ with a zero at $m + h$ where $|h| \leq d$ and $f \prec_k g$.

Now we may use Taylor's formula to obtain

$$f(m + h) - f(m) = -f(m) = \sum_{0<|\alpha|<k} \partial^\alpha f(m)h^\alpha + \int_0^1 \sum_{|\beta|=k} \partial^\beta f(m + th)w_k(dt)h^\beta$$

which implies

$$|f(m)| \leq \sum_{0<|\alpha|<k} |\partial^\alpha f(m)||h^\alpha| + \int_0^1 \sum_{|\beta|=k} \partial^\beta g(|m| + |th|)w_k(dt)|h^\beta|$$

$$\leq \sum_{0<|\alpha|<k} |\partial^\alpha f(m)| d^\alpha + \int_0^1 \sum_{|\beta|=k} \partial^\beta g(|m| + td) w_k(dt) d^\beta$$

$$= \sum_{0<|\alpha|<k} |\partial^\alpha f(m)| d^\alpha + g(|m| + d) - g(|m|) - \sum_{0<|\alpha|<k} \partial^\alpha g(|m|) d^\alpha$$

$$= g(|m| + d) - g(|m|) - \sum_{0<|\alpha|<k} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|) d^\alpha.$$

□

Corollary 4.2.1. *Let σ be a cell with midpoint m and mesh vector d. Let $f(x) \prec_1 g(x)$ for $x \in \sigma$. We then obtain the root condition:*

$$|f(m)| \leq g(|m| + d) - g(|m|)$$

This result follows immediately from the preceding theorem.

Theorem 4.2.7 (Convergence of Cells to Roots). *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the improved Taylor series root condition* (4.2)*, and let $N_F(\Lambda)$ be the set of solutions to the system. Then $N_F(\Lambda) \neq \emptyset$ if and only if the CEA does not terminate in a finite number of steps. In the latter case, $N_F(\Lambda) = \tilde{\Omega}$ where $\tilde{\Omega} = \lim_{n\to\infty} \Omega_n$.*

Proof. It suffices to show that whenever the algorithm does not terminate in a finite number of steps, then $N_F(\Lambda) \neq \emptyset$ and $N_F(\Lambda) = \tilde{\Omega}$.

Because we have a valid root condition, $N_F(\Lambda) \subseteq \tilde{\Omega}$ for any $n \geq 0$.

Thus, we must show that every element of $\tilde{\Omega}$ is also a zero, i.e., $\tilde{\Omega} \subseteq N_F(\Lambda)$. Since the algorithm does not terminate in a finite number of steps, there are cells in $\tilde{\Omega}$ for each n. Thus, $\tilde{\Omega} \neq \emptyset$.

For any fixed $\tilde{x} \in \tilde{\Omega}$, by Theorem 1.4.1, there is a sequence of cells $\{\sigma_n\}$, $\sigma_n \in \Omega_n$, such that $\tilde{x} \in \sigma_n$ for any $n \geq 0$. Since each cell σ_n must satisfy the improved Taylor series root condition (4.2) and g and f are continuous,

$$|f(\tilde{x})| \leq |f(\tilde{x}) - f(m_n)| + |f(m_n)|$$

$$\leq |f(\tilde{x}) - f(m_n)| + g(|m_n| + d_n) - g(|m_n|) - \sum_{0<|\alpha|<k} (\partial^\alpha g(|m_n|) - |\partial^\alpha f(m_n)|) d_n^\alpha \to 0$$
$$\text{as } n \to \infty.$$

Thus, $\tilde{x} \in \tilde{\Omega} \implies \tilde{x} \in N_F(\Lambda)$, so $\tilde{\Omega} \subseteq N_F(\Lambda)$. Therefore, $\tilde{\Omega} = N_F(\Lambda)$, which proves the claim. □

The following definition will be used to show that the number of cells in Ω_n obtained from the CEA 1.2.1 using this improved root condition is bounded.

Definition 4.2.4 (Order of a Zero). *We say that a zero of f is of order p if*

1. $\partial^\alpha f(\tilde{x}) = 0$ *for* $|\alpha| < p$.

2. *There exists an* $\epsilon > 0$ *such that* $\epsilon \|m - \tilde{x}\|^p \leq f(m)$ *for* $\|m - \tilde{x}\| \leq \epsilon$.

Notice that if $f(\tilde{x}) = 0$ and $f'(\tilde{x}) \neq 0$, then $\tilde{x}$ is a zero of order 1.

Corollary 4.2.2. *Let Ω_n be the sets of cells generated by the CEA 1.2.1 using the improved Taylor series root condition* (4.2), *and let $N_F(\Lambda)$ be the set of solutions to the system. If F is continuous on Λ and $N_F(\Lambda)$ consists of a finite number of solutions, where each zero has at most order k, then there is a constant N_0 such that $\#(\Omega_n) \leq N_0$ for any $n \geq 0$.*

Proof. First, we establish the following assertion:

There exist constants $C, \delta > 0$ such that if $\sigma \in \Omega_n$, with midpoint m and mesh vector d such that $\|d\| \leq \delta$, then there exists a zero $\tilde{x}$ of F such that $\|m - \tilde{x}\| \leq C\|d\|$. In other words, all cells which satisfy the improved Taylor series root condition (4.2) must lie within some neighborhood of a zero.

Then we will use the fact that F has a finite number of solutions to finish the proof.

Assume the assertion is false. Then there exists a sequence $\{\sigma_n\}$, $\sigma_i \subset \Omega_i$ with midpoints m_i and mesh vectors d_i such that $\|d_i\| \leq 1/i$ and $\|m_i - \tilde{x}\| > i\|d_i\|$ for all zeros $\tilde{x} \in N_F(\Lambda)$.

Since Λ is compact, the m_i are bounded, and we can find a convergent subsequence of the m_i, $\{m_{i_j}\}_{j=1}^{\infty}$, such that

$$\lim_{j \to \infty} m_{i_j} = \tilde{x}.$$

By Theorem 4.2.7, it follows that $\tilde{x}$ is a zero. Hence, $\tilde{x}$ has some order $p \leq k$. Thus, by definition, there exists an $\epsilon > 0$ such that

$$\epsilon \|m_{i_j} - \tilde{x}\|^p \leq |f(m_{i_j})| \tag{4.3}$$

for $\|m_{i_j} - \tilde{x}\| < \epsilon$.

The improved Taylor series root condition (4.2) gives

$$\begin{aligned} |f(m_{i_j})| &\leq g(|m_{i_j}| + d_{i_j}) - g(|m_{i_j}|) - \sum_{0<|\alpha|<k} (\partial^\alpha g(|m_{i_j}|) - |\partial f(m_{i_j})|) d_{i_j}^\alpha \\ &\leq g(|m_{i_j}| + d_{i_j}) - g(|m_{i_j}|) - \sum_{0<|\alpha|<p} (\partial^\alpha g(|m_{i_j}|) - |\partial f(m_{i_j})|) d_{i_j}^\alpha \\ &= \sum_{0<|\alpha|<p} |\partial f(m_{i_j})| d_{i_j}^\alpha + \int_0^1 \sum_{|\beta|=p} g(|m_{i_j}| + t d_{i_j}) w_k(dt) d_{i_j}^\beta. \end{aligned} \tag{4.4}$$

Expanding $\partial^\alpha f(m_{i_j})$ about $\tilde{x}$ and using the fact that all derivatives of order lower than p vanish we obtain

$$\partial^\alpha f(m_{i_j}) = \int_0^1 \sum_{\gamma:|\gamma|+|\alpha|=p} \partial^\gamma \partial^\alpha f(\tilde{x} + t(m_{i_j} - \tilde{x})) \omega_{p-|\alpha|}(dt) (m_{i_j} - \tilde{x})^{p-|\alpha|}.$$

Hence,

$$|\partial^\alpha f(m_{i_j})| = \boldsymbol{O}(\|(m_{i_j} - \tilde{x})\|^{p-|\alpha|}).$$

Since $m_{i_j} \to \tilde{x}$, as $j \to \infty$, $\|m_{i_j} - \tilde{x}\|^{p-|\alpha|}$ will be largest for $|\alpha| = 1$. Also, since $\|d_{i_j}\| \leq 1/i_j$, $d_{i_j}^\alpha \leq \|d_{i_j}\|$ for $|\alpha| > 0$. Using these facts and the fact that g is bounded on σ, it follows from (4.4) that

$$|f(m_{i_j})| \leq C \|d_{i_j}\| \|m_{i_j} - \tilde{x}\|^{p-1}. \tag{4.5}$$

Using (4.3) and (4.5) yields

$$\epsilon\|m_{i_j} - \tilde{x}\| \leq C\|d_{i_j}\| \implies \|m_{i_j} - \tilde{x}\| \leq \tilde{C}\|d_{i_j}\| .$$

Now letting $j \to \infty$, this contradicts $\|m_{i_j} - \tilde{x}\| > i_j\|d_{i_j}\|$ for all i_j. Thus, we have established the assertion.

As a result, the N-dimensional "volume" of a region about a zero which can contain cells which satisfy the root condition is

$$(2C\|d\| + 2\|d\|)^N = (2\|d\|)^N(C+1)^N .$$

The largest possible number of cells which could fit inside this region around a root would be $(2\|d\|)^N(C+1)^N \div v(\Gamma_n)$, where $v(\Gamma_n)$ is the N-dimensional "volume" of cells in Γ_n. In applications we will be able to relate $v(\Gamma_n)$ and $(2\|d_n\|)^N$ by some constant ρ. As was mentioned in Theorem 1.3.1, this will be possible since we will not be dealing with long, skinny cells.

Let

$$\rho = \max_n \frac{(2\|d_n\|)^N}{v(\Gamma_n)} .$$

Since we are dealing with a finite number of roots J, the total number of cells will be bounded by $\rho * (C+1)^N * J = N_0$.

□

Theorem 4.2.8. *Let σ be a cell with center m and mesh vector d. Let $f(m+x) \prec_k g(x)$ for $x \in \sigma$, and let $q \leq k$. We then obtain the root condition:*

$$|f(m)| \leq g(d) - g(0) - \sum_{0<|\alpha|<q} (\partial^\alpha g(0) - |\partial^\alpha f(m)|)d^\alpha. \tag{4.6}$$

The proof of this theorem is analogous to the proof of Theorem 4.2.6.

Corollary 4.2.3. *Let σ be a cell with midpoint m and mesh vector d. Let $f(m+x) \prec_1 g(x)$ for $x \in \sigma$. We then obtain the root condition:*

$$|f(m)| \leq g(d) - g(0)$$

This result follows immediately from the preceding theorem.

Oftentimes it is advantageous to have this "shifted" domination of the form $f(m+x) \prec_k g(x)$. For example, consider the functions $f(x) = e^x$ and $g(x) = e^x$. Notice $f(x) \prec_\infty g(x)$, which implies $f(x) \prec_1 g(x)$. Using the improved Taylor series root condition (4.2), we arrive at the inequality

$$|e^m| \leq e^{|m|+d} - e^{|m|} = e^{|m|}(e^d - 1)\,.$$

Let $e^{m+x} = e^m e^x = \tilde{g}(x)$, so $f(m+x) \prec_1 \tilde{g}(x)$. Now using the shifted improved Taylor series root condition (4.6), we arrive at the inequality

$$|e^m| \leq e^m e^d - e^m = e^m(e^d - 1).$$

If the components of m are negative, the root condition (4.6) provides a significantly tighter test than that of the root condition (4.2).

4.3 Numerical Results

In this section examples are presented for which the improved Taylor series root condition (4.2) was used.

1. This is an equation in one variable. The dominating function which is used is the corresponding polynomial with positive coefficients. We list the number of cells which satisfy the improved Taylor series root condition (4.2) for varying orders of domination q. We see the number of cells satisfying the root condition decreases dramatically as q increases.

   The domain of this problem is taken as $\Lambda = [-10, 10]$:

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

   This problem was numerically solved using a Matlab implementation.

Level	$q=1$	$q=2$	$q=3$	$q=4$
1	2	2	2	2
2	4	4	4	4
3	8	7	7	7
4	12	10	8	7
5	21	10	7	7
6	32	14	8	6
7	48	18	9	6
8	76	25	10	6
9	122	34	12	6
10	199	47	13	6

2. This four-dimensional fixed point problem $x = G(x)$ is taken from [21]. We recast this fixed point problem as the zero point problem $G(x) - x = 0$.

The domain of this problem is taken as

$$\Lambda = [-\pi, \pi]^2 \times [-1.5, 1.5]^2$$

where $F(x) = G(x) - x$ is given by

$$\begin{aligned} f_1(x) &= x_1 + C_1(x_3 - \alpha \sin(x_1)\cos(x_2)) - x_1 \\ f_2(x) &= x_2 + C_2(x_4 - \alpha \cos(x_1)\sin(x_2)) - x_2 \\ f_3(x) &= D_1(x_3 - \alpha \sin(x_1)\cos(x_2)) - x_3 \\ f_4(x) &= D_2(x_4 - \alpha \cos(x_1)\sin(x_2)) - x_4 \end{aligned}$$

where $\mu_1 = .1\pi, \mu_2 = .2\pi, C_1 = (1-\exp(-2\mu_1))/(2\mu_1), C_2 = (1-\exp(-2\mu_2))/(2\mu_2)$, $D_1 = \exp(-2\mu_1), D_2 = \exp(-2\mu_2), \alpha = 5$.

We obtain a dominating function of order 3 by replacing all minus signs in F with plus signs, $\sin(x_i)$ with $x_i + x_i^3/6$, and $\cos(x_i)$ with $1 + x_i^2/2 + x_i^3/6$. Before trying order 3 domination, we experimented with order 1 domination and were unsuccessful. Using order 3 domination, all 13 solutions are isolated. This isolation occurs at level 7.

This problem was numerically solved using a Matlab implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 2,688$.

Level	Cells in Ω	Level	Cells in Ω
1	16	6	160
2	256	7	96
3	2,688	8	192
4	1,180	9	220
5	328	10	228

Solutions
$(-\pi, -\pi, 0, 0)$
$(-\pi, 0, 0, 0)$
$(0, -\pi, 0, 0)$
$(-\pi/2, -\pi/2, 0, 0)$
(0,0,0,0)
$(-\pi, \pi, 0, 0)$
$(-\pi/2, \pi/2, 0, 0)$
$(0, \pi, 0, 0)$
$(\pi/2, -\pi/2, 0, 0)$
$(\pi, -\pi, 0, 0)$
$(\pi, 0, 0, 0)$
$(\pi/2, \pi/2, 0, 0)$
$(\pi, \pi, 0, 0)$

3. The following example is used in [20, 21, 22].

The domain of this problem is taken as

$$\Lambda = [-1, 2] \times [-20, 5]$$

where $F(x)$ is given by

$$f_1(x_1, x_2) = \frac{1}{2}\sin(x_1 x_2) - \frac{x_2}{4\pi} - \frac{x_1}{2}$$
$$f_2(x_1, x_2) = \left(1 - \frac{1}{4\pi}\right)(e^{2x_1} - e) + \frac{e x_2}{\pi} - 2e x_1$$

We experiment with 2 dominating functions. The first function, $G(x)$, dominates $F(x)$ with order 1. The second function, $H(x)$, dominates $F(x)$ with order 5. We replace $\sin(x_1 x_2)$ with the necessary terms of the Taylor series where minus signs are changed to plus signs. For fifth order domination, we replace e^{2x_1} with terms of the Taylor series expanded about 0 given by

$$e^{2x_1} = 1 + 2x_1 + \frac{(2x_1)^2}{2} + \frac{(2x_1)^3}{6} + \frac{(2x_1)^4}{24} + \frac{(2x_1)^5}{120}e^{\xi}$$

where ξ lies in the cell σ with midpoint m and mesh vector d under consideration. We take the maximum value of this remainder term on σ, $\frac{(2x_1)^5}{120}e^{m+d}$, to get the dominating function $H(x)$. We use fewer terms in the Taylor series expansion to calculate $G(x)$ in the same manner. We list the number of cells which satisfy the improved Taylor series root condition (4.2) for orders of domination $q = 1$ and $q = 5$. As in the first example of this section, we see the number of cells in Ω decreases significantly as q increases. This problem was also solved using the Lipschitz root condition (2.1) in Chapter 2. Notice that the number of cells satisfying the improved Taylor series root condition (4.2) is much smaller.

$G(x)$ is given by

$$g_1(x_1, x_2) = \frac{1}{2}(x_1 x_2) + \frac{x_2}{2\pi} + \frac{x_1}{2}$$
$$g_2(x_1, x_2) = \left(1 - \frac{1}{4\pi}\right)((1 + e^{2(m+d)}(2x_1)) + e) + \frac{ex_2}{\pi} + 2ex_1$$

$H(x)$ is given by

$$h_1(x_1, x_2) = \frac{1}{2}\left((x_1 x_2) + \frac{(x_1 x_2)^3}{6} + \frac{(x_1 x_2)^5}{120}\right) + \frac{x_2}{2\pi} + \frac{x_1}{2}$$
$$h_2(x_1, x_2) = \left(1 - \frac{1}{4\pi}\right)\left(\left(1 + (2x_1) + \frac{(2x_1)^2}{2} + \frac{(2x_1)^3}{6} + \frac{(2x_1)^4}{24} + \frac{(2x_1)^5}{120}e^{m+d}\right) + e\right) + \frac{ex_2}{\pi} + 2ex_1$$

This problem was numerically solved using a Matlab implementation.

Level	$q = 1$	$q = 5$	**Level**	$q = 1$	$q = 5$
1	4	3	6	78	30
2	11	9	7	76	26
3	28	20	8	84	26
4	38	26	9	78	25
5	62	34	10	80	23

Solutions
(-0.26059929, 0.6225309)
(0.29944869, 2.8369278)
(0.5, π)
(1.66342198, -16.2827906)
(1.65458272, -15.8191882)
(1.60457055, -13.3629017)
(1.57822540, -12.1766898)
(1.53050532, -10.2022479)
(1.48131957, -8.3836127)
(1.43394933, -6.8207653)
(1.33742561, -4.1404386)
(1.29436046, -3.1372198)

4. This four-bar engineering problem, see [13], is taken from Verschelde's web page [18]. It is a very long and involved polynomial system, so we will not list it here. As dominating functions, we take the corresponding polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish. Of this problem's 36 complex solutions, 3 are real. One is $(0, 0, 0, 0)$ and the other two lie close together. These zeros are isolated by the algorithm.

   The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{4} [0, 2]$$

   This problem was numerically solved using a Matlab implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 2,348$.

Level	Cells in Ω	Level	Cells in Ω
1	16	6	1,423
2	235	7	546
3	994	8	390
4	2,091	9	343
5	2,348	10	308

Solutions
(0,0,0,0)
(0.4959655, 0.1585929, 1.3982598, 0.3518096)
(0.5067453, 0.1625774, 1.4519916, 0.3683850)

5. The following system is a general economic equilibrium model which appears in [16]. The functions are taken from Verschelde's web page [18].

   The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{3} [-2, 2]$$

   where $F(x)$ is defined by

$$f_1(x) = x_2^4 - 20/7x_1^2$$

$$f_2(x) = x_1^2 x_3^4 + 7/10 x_1 x_3^4 + 7/48 x_3^4 - 50/27 x_1^2 - 35/27 x_1 - 49/216$$

$$\begin{aligned} f_3(x) = {} & 3/5 x_1^6 x_2^2 x_3 + x_1^5 x_2^3 + 3/7 x_1^5 x_2^2 x_3 + 7/5 x_1^4 x_2^3 - 7/20 x_1^4 x_2 x_3^2 \\ & - 3/20 x_1^4 x_3^3 + 609/1000 x_1^3 x_2^3 + 63/200 x_1^3 x_2^2 x_3 - 77/125 x_1^3 x_2 x_3^2 \\ & - 21/50 x_1^3 x_3^3 + 49/1250 x_1^2 x_2^3 + 147/2000 x_1^2 x_2^2 x_3 \\ & - 23863/60000 x_1^2 x_2 x_3^2 - 91/400 x_1^2 x_3^3 - 27391/800000 x_1 x_2^3 \\ & + 4137/800000 x_1 x_2^2 x_3 - 1078/9375 x_1 x_2 x_3^2 - 5887/200000 x_1 x_3^3 \\ & - 1029/160000 x_2^3 - 24353/1920000 x_2 x_3^2 - 343/128000 x_3^3 \end{aligned}$$

   As dominating functions, we take the corresponding polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish. Of this problem's 136 complex solutions, 14 are real. It should be noted that 3 of these real solutions are singular, so other root conditions will not work on this problem.

   This problem was numerically solved using a Matlab implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 490$.

Level	Cells in Ω	Level	Cells in Ω
1	8	6	126
2	48	7	94
3	240	8	76
4	490	9	72
5	238	10	60

Solutions
(-0.35000000, -0.76916057, 0)
(-0.89225985, -1.22808567, 1.14448660)
(-0.11864417, -0.4478228, 1.06565913)
(-0.03009699, -0.22555087, 1.10815809)
(-0.03009699, 0.22555087, -1.10815809)
(-0.11864417, 0.4478228, -1.06565913)
(-0.35000000, 0.76916057, 0)
(-0.89225985, 1.22808567, -1.14448660)
(0.70934009, -1.09499022, -1.16055893)
(0.61616821, -1.02054611, 1.15936721)
(0.03966868, -0.25894459, 1.12559270)
(0.03966868, 0.25894459, -1.12559270)
(0.61616821, 1.02054611, -1.15936721)
(0.70934009, 1.09499022, 1.16055893)

6. Here we have the Wright problem taken from Verschelde's web page [18].

The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{5} [-6, 6]$$

where $F(x)$ is defined by

$$f_1(x) = x_1^2 - x_1 + x_2 + x_3 + x_4 + x_5 - 10$$

$$f_2(x) = x_2^2 - x_2 + x_1 + x_3 + x_4 + x_5 - 10$$

$$f_3(x) = x_3^2 - x_3 + x_2 + x_1 + x_4 + x_5 - 10$$

$$f_4(x) = x_4^2 - x_4 + x_2 + x_3 + x_1 + x_5 - 10$$

$$f_5(x) = x_5^2 - x_5 + x_2 + x_3 + x_4 + x_1 - 10$$

As dominating functions, we take the corresponding polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish. This problem has 32 solutions, all of which are real, and are isolated by the algorithm after 7 levels.

This problem was numerically solved using a Java implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 1,298$.

Level	Cells in Ω	Level	Cells in Ω
1	32	11	1,048
2	443	12	1,298
3	863	13	1,148
4	1,013	14	1,033
5	1,258	15	1,088
6	1,128	16	1,068
7	1,148	17	1,248
8	1,128	18	1,103
9	1,068	19	1,143
10	1,143	20	1,068

Solutions
(- 5, - 5, - 5, - 5, - 5)
(- 3.3723, - 3.3723, - 3.3723, - 3.3723, 5.3723)
(- 3.3723, - 3.3723, - 3.3723, 5.3723, - 3.3723)
(- 2, - 2, - 2, 4, 4)
(- 3.3723, - 3.3723, 5.3723, - 3.3723, - 3.3723)
(- 2, - 2, 4, - 2, 4)
(- 2, - 2, 4, 4, - 2)
(- 1, - 1, 3, 3, 3)
(- 3.3723, 5.3723, - 3.3723, - 3.3723, - 3.3723)
(- 2, 4, - 2, - 2, 4)
(- 2, 4, - 2, 4, - 2)
(- 1, 3, - 1, 3, 3)
(- 2, 4, 4, - 2, - 2)
(- 1, 3, 3, - 1, 3)
(- 1, 3, 3, 3, - 1)
(- .3723, 2.3723, 2.3723, 2.3723, 2.3723)
(5.3723, - 3.3723, - 3.3723, - 3.3723, - 3.3723)
(4, - 2, - 2, - 2, 4)
(4, - 2, - 2, 4, -2)
(3, - 1, - 1, 3, 3)
(4, - 2, 4, - 2, - 2)
(3, - 1, 3, - 1, 3)
(3, - 1, 3, 3, - 1)
(2.3723, - .3723, 2.3723, 2.3723, 2.3723)
(4, 4, - 2, - 2, - 2)
(3, 3, - 1, - 1, 3)
(3, 3, - 1, 3, - 1)
(2.3723, 2.3723, - .3723, 2.3723, 2.3723)
(3, 3, 3, - 1, - 1)
(2.3723, 2.3723, 2.3723, - .3723, 2.3723)
(2.3723, 2.3723, 2.3723, 2.3723, - .3723)
(2, 2, 2, 2, 2)

7. This is the Boon problem taken from Verschelde's web page [18]. This problem comes from the field of neurophysiology.

   The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{5}[-2, 2]$$

   where $F(x)$ is defined by

$$f_1(x) = x_5^2 + x_3^2 - 1$$

$$f_2(x) = x_6^2 + x_4^2 - 1$$

$$f_3(x) = x_1 x_3^3 + x_2 x_4^3 - 1.2$$

$$f_4(x) = x_1 x_5^3 + x_2 x_6^3 - 1.2$$

$$f_5(x) = x_1 x_3^2 x_5 + x_2 x_4^2 x_6 - .7$$

$$f_6(x) = x_1 x_3 x_5^2 + x_2 x_4 x_6^2 - .7$$

   As dominating functions, we take the corresponding polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish. This problem has 8 solutions, all of which are real, and are isolated by the algorithm.

   This problem was numerically solved using a Java implementation. We see that the number of cells in Ω is bounded by $N_0 \approx 17,568$.

Level	Cells in Ω	Level	Cells in Ω
1	64	6	13,416
2	4,096	7	15,672
3	10,564	8	14,064
4	6,132	9	13,808
5	17,568	10	13,896

Solutions
(- .9154, - .4025, - .4025, - .9154, - 1.4417, - 1.4417)
(- .4025, - .9154, - .9154, - .4025, - 1.4417, - 1.4417)
(- .9154, .4025, - .4025, .9154, - 1.4417, 1.4417)
(- .4025, .9154, - .9154, .4025, - 1.4417, 1.4417)
(.4025, - .9154, .9154, - .4025, 1.4417, - 1.4417)
(.9154, - .4025, .4025, - .9154, 1.4417, - 1.4417)
(.4025, .9154, .9154, .4025, 1.4417, 1.4417)
(.9154, .4025, .4025, .9154, 1.4417, 1.4417)

8. The heart-dipole problem, taken from Verschelde's web page [18], is the highest dimensional problem which we solved. This problem comes from the field of medicine.

   The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{8} [-2, 2]$$

   where $F(x)$ is defined by

$$f_1(x) = x_1 + x_2 - .63254$$

$$f_2(x) = x_3 + x_4 + 1.34534$$

$$f_3(x) = x_5x_1 + x_6x_2 - x_7x_3 - x_8x_4 + .8365348$$

$$f_4(x) = x_7x_1 + x_8x_2 + x_5x_3 + x_6x_4 - 1.7345334$$

$$f_5(x) = x_1x_5^2 - x_1x_7^2 - 2x_3x_5x_7 + x_2x_6^2 - x_2x_8^2 - 2x_4x_6x_8 - 1.352352$$

$$f_6(x) = x_3x_5^2 - x_3x_7^2 + 2x_1x_5x_7 + x_4x_6^2 - x_4x_8^2 + 2x_2x_6x_8 + .843453$$

$$f_7(x) = x_1x_5^3 - 3x_1x_5x_7^2 + x_3x_7^3 - 3x_3x_7x_5^2 + x_2x_6^3 - 3x_2x_6x_8^2 + x_4x_8^3 - 3x_4x_8x_6^2 + .9563453$$

$$f_8(x) = x_3x_5^3 - 3x_3x_5x_7^2 - x_1x_7^3 + 3x_1x_7x_5^2 + x_4x_6^3 - 3x_4x_6x_8^2 - x_2x_8^3 + 3x_2x_8x_6^2 - 1.2342523$$

   As dominating functions, we take the corresponding polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish. This problem has 2 real solutions. After 7 levels of partitioning the 2 solutions are isolated, but there are also 2 false positive components. After 8 levels of partitioning the 2 false positive components have

been excluded, and we are left with the 2 components corresponding to the 2 solutions.

This problem was numerically solved using a Java implementation. When solving higher dimensional problems, we observe that the bound on the number of cells can be quite large. In this case, we see $N_0 \approx 1,178,268$.

It is interesting to note that the Bezout number for this system is 576. This means that for general coefficients such a system could have 576 complex solutions with accounting for multiple solutions.

Level	Cells in Ω	Level	Cells in Ω
1	144	5	1,178,268
2	7,942	6	672,596
3	134,222	7	36,042
4	534,655	8	31,536

Solutions
(- .0105, .6431, .2675, -1.6128, 1.1652, -.9551, -.3529, -.1888)
(.6431, -.0105, -1.6128, .2675, -.9551, 1.1652, -.1888, -.3529)

9. The following example comes from the field of physics [17]. It is a fixed point problem $K = H(K)$. We recast this fixed point problem as the zero point problem $H(K) - K = 0$.

The domain of this problem is taken as

$$\Lambda_1 = \prod_{i=1}^{3}[-.1, 1.1]$$

where $G(K) = H(K) - K$ is given by

$$g_1(K) = \frac{1}{2}\ln\left(\frac{A\cosh(2K_1)}{B\cosh(2K_2)\cosh(2K_3)}\right) - K_1$$

$$g_2(K) = \frac{1}{2}\ln\left(\frac{A\cosh(2K_2)}{B\cosh(2K_1)\cosh(2K_3)}\right) - K_2$$

$$g_3(K) = \frac{1}{2}\ln\left(\frac{A\cosh(2K_3)}{B\cosh(2K_1)\cosh(2K_2)}\right) - K_3$$

with

$$A = \exp(K_1 + K_2 + K_3)\cosh(2(K_1 + K_2 + K_3))$$

$$+ \exp(K_1 - K_2 - K_3)\cosh(2(K_1 - K_2 - K_3))$$

$$+ \exp(-K_1 - K_2 + K_3)\cosh(2(K_1 + K_2 - K_3))$$

$$+ \exp(-K_1 + K_2 - K_3)\cosh(2(K_1 - K_2 + K_3))$$

$$B = \exp(K_1 + K_2 + K_3) + \exp(K_1 - K_2 - K_3)$$

$$+ \exp(-K_1 - K_2 + K_3) + \exp(-K_1 + K_2 - K_3)$$

We use the substitution $x_i = \exp^{-K_i}$ to transform the system of equations into a polynomial system where the Java software (see appendix) can be readily implemented. As dominating functions, we take the corresponding

polynomial system with positive coefficients. We assume the highest order of domination. In other words, we calculate the derivatives in the Taylor series root condition (4.2) until they vanish.

This problem was numerically solved using a Java implementation.

Levels	Cells in Ω	Levels	Cells in Ω
1	8	6	2,053
2	64	7	3,916
3	262	8	7,690
4	586	9	15,091
5	1,048	10	28,747

Here we use the same system of equations, but the domain is taken as

$$\Lambda_2 = \prod_{i=1}^{3} [.25, 1.25]$$

We use the substitution $x_i = \exp^{-K_i}$.

This problem was numerically solved using a Java implementation.

Levels	Cells in Ω	Levels	Cells in Ω
1	8	11	885
2	64	12	882
3	315	13	879
4	233	14	861
5	136	15	855
6	249	16	846
7	444	17	879
8	741	18	885
9	1,164	19	912
10	921	20	876

We notice that the number of cells satisfying the Taylor series root condition stabilizes for the second domain, Λ_2, but explodes using the first domain, Λ_1.

This is due to the fact that $x_1 = 0, x_2 = 0$, and $x_3 = 0$ are values which are included in Λ_1. We see in Figure 4.1 that lines of solutions where two of x_1, x_2, and x_3 are zero are included in Λ_1. The number of cells along these lines explodes. These lines clearly do not represent isolated solutions, and they also do not represent physically realistic solutions. When we exclude zero in Λ_2, the number of cells stabilizes. In this case we obtain 8 components which correspond to 8 isolated solutions. See Figure 4.2. We notice the propeller-like region in Figure 4.1 corresponds to 4 isolated solutions in Figure 4.2.

After 20 levels of partitioning the 876 cells in Ω are linked into 8 components corresponding to the 8 isolated zeros. The midpoints and mesh vectors of these components, which are produced by the algorithm, are given.

	Midpoint	**Mesh Vector**
1	(.74611807, .74611807, .74611807)	(.00001001, .00001001, .00001001)
2	(.71841860, .71841860, .80202198)	(.00000620, .00000620, .00000954)
3	(.71841860, .80202198, .71841860)	(.00000620, .00000954, .00000620)
4	(.54368925, 1.00000000, 1.00000000)	(.00000048, .00000095, .00000095)
5	(.80202198, .71841860, .71841860)	(.00000954, .00000620, .00000620)
6	(1.00000000, .54368925, 1.00000000)	(.00000095, .00000048, .00000095)
7	(1.00000000, 1.00000000, .54368925)	(.00000095, .00000095, .00000048)
8	(1.00000000, 1.00000000, 1.00000000)	(.00000095, .00000095, .00000095)

Solutions
(.7461, .7461, .7461)
(.71842, .71842, .80202)
(.71842, .80202, .71842)
(.543689, 1, 1)
(.80202, .71842, .71842)
(1, .543689, 1)
(1, 1, .543689)
(1,1,1)

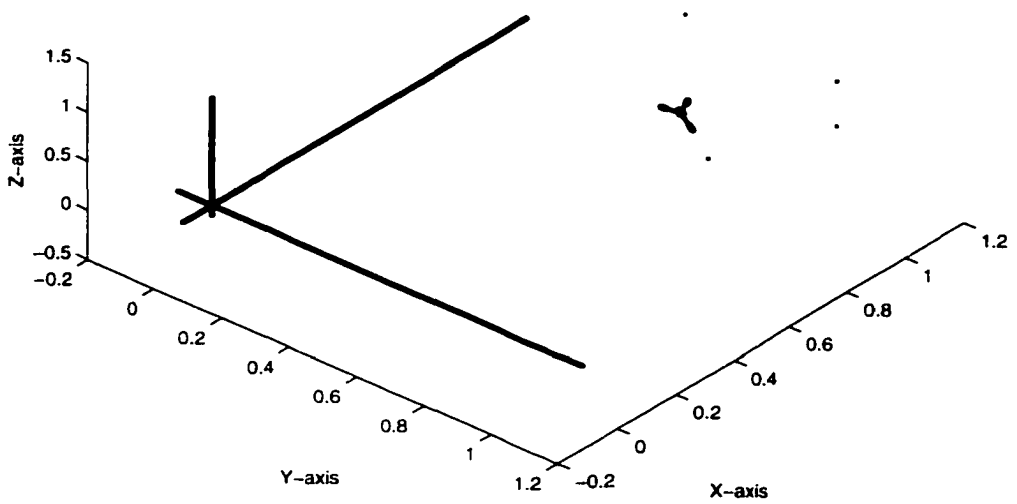


Figure 4.1: Physics Example with Λ_1

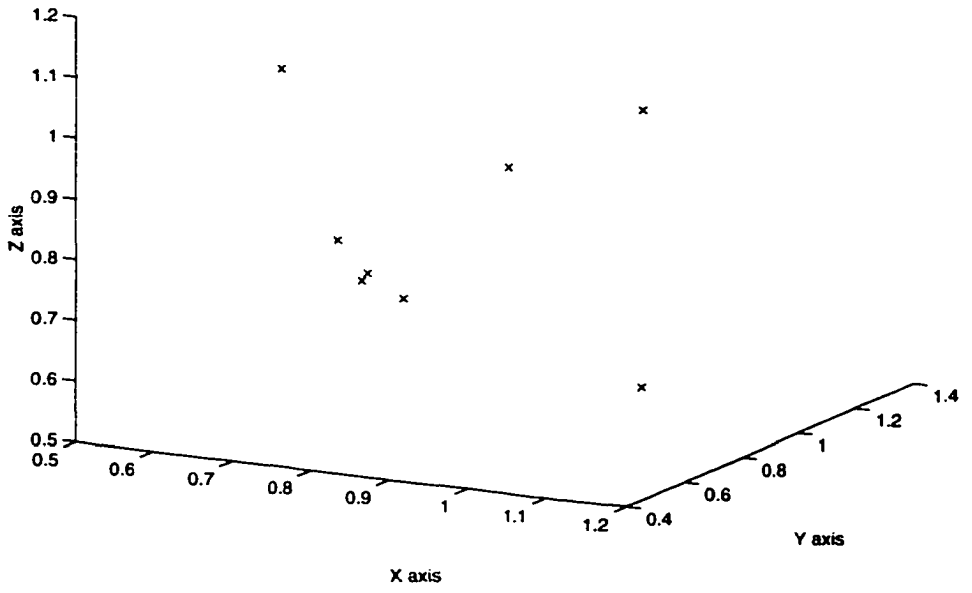


Figure 4.2: Physics Example with Λ_2

Chapter 5

CELL EXCLUSION ALGORITHMS AND OPTIMIZATION

In this chapter we will investigate how cell exclusion algorithms may be used to find the global minimum of a continuous function from $\mathbb{R}^N$ to $\mathbb{R}$ of relatively small dimension N on a domain Λ.

We begin with some introductory definitions:

Definition 5.0.1 (Minimization Condition). *A minimization condition is a computationally verifiable necessity test for the presence of a global minimum in a cell.*

Definition 5.0.2 ($\mathcal{M}_f(\Lambda)$). *We define $\mathcal{M}_f(\Lambda)$ to be the set of points where a function $f : \Lambda \subset \mathbb{R}^N \to \mathbb{R}$ attains its global minimum in the cell Λ.*

Definition 5.0.3 (Ω_k). *The set of cells which satisfy the minimization condition on the kth level of partitioning is called Ω_k.*

Definition 5.0.4 (Γ_k). *The set of cells which result when Ω_{k-1} undergoes one level of partitioning is called Γ_k.*

5.1 A General Optimization Algorithm

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

Algorithm 5.1.1. *1. Let Γ_k be the sequence of cellular partitions of Λ defined above with $\Gamma_0 = \{\Lambda\}$ such that Γ_{k+1} is finer than $\Gamma_k, k = 0, 1, \ldots$ and the mesh sizes $\lim_{k\to\infty} \|d_k\| = 0$. Let M_k be the current approximate minimum value generated by the algorithm at level k.*

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

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

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

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

If $\|d_k\| < tol$,

then print M_k.

Else

i. $\Omega_{k+1} \leftarrow \emptyset$.

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

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

If σ satisfies the minimization condition,

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

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

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

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

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

5.2 Consequences of the Optimization Algorithm

The sequence we obtain using the minimization algorithm 5.1.1 does indeed converge to the global minimum.

Theorem 5.2.1. *(Obtaining the Global Minimum) Let $f : \Lambda \subset \mathbb{R}^N \to \mathbb{R}$ be a continuous function on some cell Λ, and let $\tilde{x} \in \mathcal{M}_f(\Lambda)$. The decreasing sequence of values $\{M_n\}$ generated by the minimization algorithm 5.1.1 converges to the global minimum $f(\tilde{x})$.*

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

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

Now for $\tilde{x} \in \mathcal{M}_f(\Lambda)$, it follows that $\tilde{x} \in \sigma_n \subset \Lambda$ for a sequence of cells $\{\sigma_n\}_{n=1}^{\infty}$ with mesh vectors d_n and midpoints m_n such that $\|d_n\| \to 0$ as $n \to \infty$.

For sufficiently large n,

$$\|m_n - \tilde{x}\| < \delta \text{ and hence } f(m_n) - f(\tilde{x}) < \epsilon$$

$$\text{and as a result } f(\tilde{x}) \leq M_n \leq f(m_n) < f(\tilde{x}) + \epsilon$$

Thus, since ϵ is arbitrary, M_n must approach $f(\tilde{x})$ as $n \to \infty$. □

Lemma 5.2.1. *Let f be C^2 on Λ and let $f : \Lambda \subset \mathbb{R}^N \to \mathbb{R}$ attain its global minimum at a regular zero point, $\tilde{x}$, of the gradient on the interior of the cell Λ, then*

$$M_n \leq f(\tilde{x}) + C\|d_n\|^2$$

where M_n is the sequence defined in the minimization algorithm 5.1.1, d_n is the sequence of mesh vectors in Ω_n, and C is some positive constant independent of n.

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

$$M_n \leq \min_{\sigma \in \Omega_n} f(m_\sigma)$$

We know $\tilde{x} \in \sigma$ for a $\sigma \in \Omega_n$ with midpoint m_σ.

Now, expanding f about $\tilde{x}$ we have

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

since the gradient of f vanishes at $\tilde{x}$. We denote the Hessian of f by H_f.

Now since $\tilde{x}$ is a regular point of the gradient, there exists a constant λ^* such that

$$0 < \lambda^* = \|H_f(\tilde{x})\|_2 .$$

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

$$f(m_\sigma) - f(\tilde{x}) \leq C\|m_\sigma - \tilde{x}\|^2 \implies M_n \leq f(m_\sigma) \leq f(\tilde{x}) + C\|d_n\|^2 .$$

We may increase C to make the inequality hold for all preceding levels and all global minimum points.

□

5.3 A Monotonicity Minimization Condition

Theorem 5.3.1. *Let σ be a cell, let f be a monotonically decomposable function with $f = g - h$, and let $M \in \mathbb{R}$. If f satisfies the following condition:*

$$g(\underline{\sigma}) - h(\overline{\sigma}) > M \tag{5.1}$$

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

Proof. By monotonicity, $g(\underline{\sigma}) - h(\overline{\sigma}) \leq g(x) - h(x) = f(x) \quad \forall x \in \sigma$.

Thus, $M < g(\underline{\sigma}) - h(\overline{\sigma}) \implies M < f(x) \quad \forall x \in \sigma$.

□

5.4 A Taylor Series Minimization Condition

Theorem 5.4.1. *Let σ be a cell with midpoint m and mesh vector d and let $M \in \mathbb{R}$. If $f \prec_k g$ and f satisfies the following condition:*

$$f(m) - g(|m| + d) + g(|m|) + \sum_{0<|\alpha|<k} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|)d^\alpha > M \qquad (5.2)$$

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

Proof. Let $m + h \in \sigma$ with $|h| \leq d$.

Now we may use Taylor's formula to obtain

$$f(m+h) = f(m) + \sum_{0<|\alpha|<k} \partial^\alpha f(m)h^\alpha + \int_0^1 \sum_{|\beta|=k} \partial^\beta f(m+th)w_k(dt)h^\beta$$

which implies

$$\begin{aligned} f(m+h) &\geq f(m) - \sum_{0<|\alpha|<k} |\partial^\alpha f(m)||h^\alpha| - \int_0^1 \sum_{|\beta|=k} \partial^\beta g(|m| + |th|)w_k(dt)|h^\beta| \\ &\geq f(m) - \sum_{0<|\alpha|<k} |\partial^\alpha f(m)|d^\alpha - \int_0^1 \sum_{|\beta|=k} \partial^\beta g(|m| + td)w_k(dt)d^\beta \\ &= f(m) - \sum_{0<|\alpha|<k} |\partial^\alpha f(m)|d^\alpha - (g(|m| + d) - g(|m|) - \sum_{0<|\alpha|<k} \partial^\alpha g(|m|)d^\alpha) \\ &= f(m) - g(|m| + d) + g(|m|) + \sum_{0<|\alpha|<k} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|)d^\alpha . \end{aligned}$$

In conclusion,

$$f(m+h) \geq f(m) - g(|m| + d) + g(|m|) + \sum_{0<|\alpha|<k} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|)d^\alpha .$$

Thus,

$$f(m) - g(|m| + d) + g(|m|) + \sum_{0<|\alpha|<k} (\partial^\alpha g(|m|) - |\partial^\alpha f(m)|)d^\alpha > M$$

is sufficient for showing that f does not attain a value smaller than or equal to M on σ.

□

Corollary 5.4.1. *Let $\{M_n\}$ be the decreasing sequence of values generated by the minimization algorithm 5.1.1 using the Taylor series minimization condition* (5.2). *If f is C^2 on Λ, $f \prec_k g$ with $k \geq 2$, f attains its global minimum on the interior of a cell Λ, and $\mathcal{M}_f(\Lambda)$ consists of a finite subset of the regular zero points of the gradient, then there is a constant N_0 such that $\#(\Omega_n) \leq N_0$ for any $n \geq 0$.*

Proof. First, we establish the following assertion:

There exist constants $C, \delta > 0$ such that if $\sigma \in \Omega_n$ with midpoint m and mesh vector d such that $\|d\| \leq \delta$, then there exists a point $\tilde{x} \in \mathcal{M}_f(\Lambda)$ such that $\|m - \tilde{x}\| \leq C\|d\|$. In other words, all cells which satisfy the minimization condition must lie within some small neighborhood of a point where f attains its global minimum.

Then we will use the fact that F attains its global minimum at a finite number of points to complete the proof.

Assume the assertion is false. Then there exists a sequence $\{\sigma_i\}$, $\sigma_i \subset \Omega_i$, with mesh vectors d_i such that

$$\|d_i\| \leq 1/i \text{ and } \|m_i - \tilde{x}\| > i\|d_i\| \tag{5.3}$$

for all points $\tilde{x} \in \mathcal{M}_f(\Lambda)$.

Since Λ is compact, the m_i are bounded, and we can find a convergent subsequence $\{m_{i_j}\}_{j=1}^{\infty}$ such that

$$\lim_{j \to \infty} m_{i_j} = \tilde{x}\,.$$

By Theorem 5.2.1, it follows that $\tilde{x} \in \mathcal{M}_f(\Lambda)$. We may expand f in a Taylor series about $\tilde{x}$

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

where $H_f(\tilde{x})$ is the Hessian of f.

Now since $\tilde{x}$ is a regular point of the gradient, $\nabla f(\tilde{x}) = 0$ and $H_f(\tilde{x})$ is positive definite. Thus, there exists a $\lambda_* > 0$ such that

$$f(x) \geq f(\tilde{x}) + \frac{1}{2}\lambda_* \|x - \tilde{x}\|^2 + O(\|x - \tilde{x}\|^3) .$$

For small enough $\|x - \tilde{x}\|$ we have

$$f(x) - f(\tilde{x}) \geq K_1 \|x - \tilde{x}\|^2 \tag{5.4}$$

for some constant K_1 such that $\frac{1}{2}\lambda_* > K_1 > 0$.

Now we will use the minimization condition (5.2) to arrive at a contradiction.

First, we know that at a regular critical point of the gradient, $\tilde{x}$, there exist constants $\gamma_1, \gamma_2 > 0$ such that for $\|x - \tilde{x}\|$ small enough,

$$\gamma_1 \|\nabla f(x)\| \leq \|x - \tilde{x}\| \leq \gamma_2 \|\nabla f(x)\| . \tag{5.5}$$

The minimization condition is

$$f(m_n) - M_n \leq g(|m_n| + d_n) - g(|m_n|) - \sum_{0<|\alpha|<k} (\partial^\alpha g(|m_n|) - |\partial^\alpha f(m_n)|) d_n^\alpha$$

where M_n is the current minimum value of f on Λ determined by the algorithm.

Now,

$$g(|m_n| + d_n) = g(|m_n|) + \sum_{0<|\alpha|<k} \partial^\alpha g(|m_n|) d_n^\alpha + O(\|d_n\|^k) .$$

Using this fact, $k \geq 2$, and Lemma 5.2.1 the minimization condition becomes

$$f(m_n) - f(\tilde{x}) \leq \sum_{0<|\alpha|<2} |\partial^\alpha f(m_n)| d_n^\alpha + O(\|d_n\|^2) .$$

Hence, we have

$$f(m_n) - f(\tilde{x}) \leq K_2 \|\nabla(f(m_n))\| \|d_n\| + O(\|d_n\|^2)$$

for some $K_2 > 0$.

Now by (5.5) for sufficiently large n,

$$f(m_n) - f(\tilde{x}) \leq \gamma \|m_n - \tilde{x}\| \|d_n\| + O(\|d_n\|^2)$$

for some $\gamma > 0$.

We conclude from (5.4) that for sufficiently large n,

$$K_1 \|m_n - \tilde{x}\|^2 \leq f(m_n) - f(\tilde{x}) \leq \gamma \|m_n - \tilde{x}\| \|d_n\| + O(\|d_n\|^2) .$$

This implies that for sufficiently large n,

$$K_1 \|m_n - \tilde{x}\|^2 \leq K_3 \|m_n - \tilde{x}\| \|d_n\|$$

for some $\gamma > K_3 > 0$.

Thus, dividing by $\|m_n - \tilde{x}\|$, which is positive by assumption (5.3), we get

$$K_1 \|m_n - \tilde{x}\| \leq K_3 \|d_n\| .$$

Using the assumption (5.3) again, this implies,

$$0 < \frac{K_1}{K_3} \leq \frac{\|d_n\|}{\|m_n - \tilde{x}\|} < \frac{\|d_n\|}{n\|d_n\|} = \frac{1}{n} \to 0 \text{ as } n \to \infty$$

As a result, $K_1 = 0$. This contradicts the fact that $K_1 > 0$. Thus, we have established the assertion.

This proof may now be completed in the same manner as that of Corollary 4.2.2.

□

Conclusion

In this dissertation we present the basic theory of zero-finding and optimization cell exclusion algorithms. We discuss the Lipschitz, monotonicity, and Taylor series root conditions. The improved Taylor series root condition is a localized root condition which is very effective, especially when working with systems possessing singular solutions. For optimization problems we develop monotonicity and Taylor series minimization conditions which promise to be effective when finding the global minimum of functions of relatively few variables with possibly many local minima.

Bibliography

[1] E. L. Allgower and K. Georg. Numerical Path Following. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of Numerical Analysis*, volume 5, pages 3–207. North-Holland, 1997.

[2] E. L. Allgower and K. Georg. Relationships Between Deflation and Global Methods in the Problem of Approximating Additional Zeros of a System of Nonlinear Equations. In B. C. Eaves, F. J. Gould, H.-O. Peitgen and M. J. Todd, editors, *Homotopy Methods and Global Convergence*, pages 31–42. Plenum Press, 1983.

[3] A. Arazyan and R. B. Kearfott. Taylor Series Models in Deterministic Global Optimization. In *Proceedings of AD 2000, the Third International Conference and Workshop on Automatic Differentiation, June 19-23, 2000, Nice, France.* SIAM. To be published by SIAM.

[4] F. H. Branin and S. K. Hoo. A Method for Finding Multiple Extrema of a Function of n Variables. In F. A. Lootsma, editor, *Numerical Methods for Non-linear Optimization*, pages 231–237. Academic Press, 1972.

[5] K. M. Brown and W. B. Gearhart. Deflation Techniques for the Calculation of Further Solutions of a Nonlinear System. *Numer. Math.*, 16:334–342, 1971.

[6] K. Georg. Improving the Efficiency of Exclusion Algorithms. To appear, 2000.

[7] E. R. Hansen. An Overview of Global Optimization Using Interval Analysis. In R. E. Moore, editor, *Reliability in Computing*, Academic Press, 1988.

[8] R. Horst and P. M. Pardalos. *Handbook of Global Optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1995.

[9] R. B. Kearfott. Empirical Evaluation of Innovations in Interval Branch and Bound Algorithms for Nonlinear Algebraic Systems. *SIAM Journal on Scientific Computing*, 18(2):574–594, 1997.

[10] R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[11] S. B. Maurer and A. Ralston. *Discrete Algorithmic Mathematics, 2nd edition.* AK Peters, 1998.

[12] R. E. Moore. *Methods and Applications of Interval Analysis.* SIAM, 1979.

[13] A. P. Morgan and C. W. Wampler. Solving a Planar Four-Bar Design Problem Using Continuation. *J. Mech. Design*, 112:544–550, 1990.

[14] A. P. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[15] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Wiley, 1988.

[16] J. B. Shoven. Applied General Equilibrium Modelling. *IMF Staff Papers.* pages 394–419, 1983.

[17] K.-F. Tang and J.-Z. Hu. Critical Properties of the Anisotropic Ising Model on Three-Node Hierarchical Lattices. *J. Phys. A: Math. Gen.*, 21:785-790, 1988.

[18] J. Verschelde. Algorithm 795: Phcpack: A General-Purpose Solver for Polynomial Systems by Homotopy Continuation. *ACM Transactions on Mathematical Software*, 25:251–276, 1999. A collection of polynomial systems analyzed with this algorithm is posted on the net: `http://www.math.uic.edu/~jan/demo.html`.

[19] Z.-B. Xu, J.-S. Zhang, and Y.-W. Leung. A General CDC Formulation for Specializing the Cell Exclusion Algorithms of Finding All Zeros of Vector Functions. *Appl. Math. Comput.*, 86:235–259, 1997.

[20] Z.-B. Xu, J.-S. Zhang, and W. Wang. A Cell Exclusion Algorithm for Determining All the Solutions of a Nonlinear System of Equations. *Appl. Math. Comput.*, 80:181–208, 1996.

[21] P. J. Zufiria and R. S. Guttalu. A Computational Method for Finding All Roots of a Vector Function. *Applied Mathematics and Computation*, 35(1):13–59, 1990.

[22] P. J. Zufiria and R. S. Guttalu. On an Application of Dynamical Systems Theory to Determining All Zeros of a Vector Function. *J. Math. Anal. Appl.*, 152:269–295, 1990.

Appendix A

COMPUTER CODE

In this appendix we present computer code which we use to investigate various numerical examples. First, we present Matlab code which I wrote. Dr. Kurt Georg wrote some Matlab software which implements the improved Taylor series root condition. In an effort to write faster algorithms, I wrote some programs in C^{++} which implement the basic Lipschitz and monotonicity root conditions and find components. A component corresponds to the many cells which cluster about a zero. These cells may be linked into one component, and the midpoint of this component is the only necessary output. We include neither Dr. Georg's Matlab software nor my C^{++} programs in this appendix.

With the prospect of implementing the improved Taylor series root condition and the linking of components in C^{++} seeming quite daunting, we decided to use Java for this task. Java is faster than Matlab, and Java does not involve pointers. Dr. Georg wrote some Java software that implements the improved Taylor series root condition for polynomial systems and links cells into components. This software is easy to use and very effective. We present this code next.

In this dissertation we do not implement the minimization algorithm. The implementation of the minimization algorithm 5.1.1 differs from that of the zero-finding algorithm 1.2.1 in one crucial way. The zero-finding algorithm is recursive. Starting with a large cell, it will subdivide again and again until a cell either

fails the root condition or becomes small enough. After either of these two things occur, the algorithm then jumps back to a point in the recursion where it begins to subdivide a larger cell. This is what is called a depth first search algorithm [11].

On the other hand, the minimization algorithm is not purely recursive. At every level of partitioning k, we must use the minimization condition to test all cells of this level's size. Then we arrive at a current minimum value of the algorithm, M_k, on the kth level of partitioning. Thus, on each level all cells of the same size must be stored in a queue. Then the minimization condition is applied to each of these cells in turn. Those that satisfy the minimization condition are then subdivided and stored in a new queue for the next level. This is what is referred to as a breadth first search algorithm [11].

A.1 Matlab Code

First, we present the Matlab code used for sequential and simultaneous bisection with the Lipschitz root condition. Then we present the Matlab code used for sequential bisection, sequential trisection, and simultaneous bisection with the monotonicity root condition.

Lipschitz Root Condition: Sequential Bisection

```
function seq_lipschitz(F, a1, a2, c, mu, k)
% Finds all solutions of F(x) = 0
% in the N-box [a1, a2] where F, x, c, a1, and a2 are N-dim.
% c is the vector of Lipschitz constants.
% Uses recursive bisection and the Lipschitz root condition:
% F(a1+a2/2) <= lipschitz. const * norm(a1-a2, inf)/2
% k = axis along which we will be dividing.  ** Call seq_lipschitz with k=1.

global ROOTCHECKS
global CELLS
N=length(a1); % N = dimension of the space in which we are.
m = (a1+a2) ./ 2;% Define the midpoint of the cell.
ROOTCHECKS = ROOTCHECKS + 1;

if (min (abs(feval(F, m)') <= abs(c .* (m-a1)))' == 1)
        if norm(a2-a1, inf) < mu
                CELLS = CELLS + 1;
                [a1; a2];
                return;
        end
        % a1 = a1
        b2 = a2;
        a2(k) = (a1(k)+a2(k))/2;
        b1 = a1;
        b1(k) = a2(k);
        if (k < N)
                k=k+1;
        else
                k=1;
        end
        seq_lipschitz(F,a1,a2,c,mu,k)
        seq_lipschitz(F,b1,b2,c,mu,k)
end
```

Lipschitz Root Condition: Simultaneous Bisection

```
function simult_lipschitz(F, a1, a2, c, mu)
% Finds all solutions of F(x) = 0
% in the N-box [a1, a2] where F, x, c, a1, and a2 are N-dim.
% c is the vector of Lipschitz constants.
% Uses recursive bisection and the Lipschitz root condition:
% F(a1+a2/2) <= lipschitz. const * norm(a1-a2, inf)/2
% ********** Bisects along all axes simultaneously!
% place = variable that marks the current position in a row of A
% trigger =  0 then fill kth position with a1(k)
%            1 then fill kth position with a1(k)+(a2(k)-a1(k))/2
% times = keeps track of the # of times one has progressed through the
%         i-loop.  Used for updating place.

global CELLS
global ROOTCHECKS

N = length(a1); % N = dimension of the space in which we are.
A = zeros(N, 2^(N+1));

for k=1:N
        place=0;
        times=0;
        trigger=0;
    while (place < (2^(N+1)))
        for i=1:2:(2^(N-k+1)-1)
                if (trigger == 0)
                        A(k, (i+place)) = a1(k);
                end
                if (trigger == 1)
                        A(k, (i+place)) = a1(k)+(a2(k)-a1(k))/2;
                end
        end
        trigger = mod(trigger+1, 2);
        times = times + 1;
        place = times*(2^(N-k+1));
    end
end

for i=2:2:2^(N+1)
        A(:,i) = A(:,(i-1)) + (a2-a1)'/2;
end
```

```
for i=1:2:(2^(N+1)-1) % Steps of 2 to hit all new cells which were formed.
    m = (A(:,i)+A(:,i+1)) ./2;
    ROOTCHECKS = ROOTCHECKS + 1;
    if (min (abs(feval(F, m)') <= abs(c .* (m-A(:,i))')) == 1)
          if norm(A(:,i+1) - A(:,i), inf) < mu
                  [A(:,i)'; A(:,i+1)'];
                  CELLS = CELLS + 1;
          else
                simult_lipschitz(F, A(:,i)', A(:,i+1)', c, mu);
          end
    end
end
```

Monotonicity Root Condition: Sequential Bisection

```
function seq_mon(G, H, a1, a2, mu, k)
% Finds all solutions of F(x) = G(x) - H(x) = 0
% in the N-box [a1, a2] where F, G, H, x, a1, and a2 are N-dim.
% vectors and all components of G and H are increasing inside
% [a1, a2].  G and H are vector fields.
% Uses recursive bisection and the monotonicity root condition:
% G(a1) <= H(a2) and H(a1) <= G(a2).
% k = axis along which we will be dividing.  ** Call seq_mon with k=1.

global CELLS
global ROOTCHECKS

N=length(a1); % N = dimension of the space in which we are.
ROOTCHECKS = ROOTCHECKS + 1;

if  ((min(feval(G,a1)<= feval(H,a2)) == 1) &
                                    (min(feval(H,a1) <= feval(G,a2)) == 1))
        if norm(a2-a1, inf) < mu
                [a1; a2]
                CELLS = CELLS + 1;
                return;
        end
        % a1 = a1
        b2 = a2;
        a2(k) = (a1(k)+a2(k))/2;
        b1 = a1;
        b1(k) = a2(k);
        if (k < N)
                k=k+1;
        else
                k=1;
        end
        seq_mon(G,H,a1,a2,mu,k)
        seq_mon(G,H,b1,b2,mu,k)
end
```

Monotonicity Root Condition: Sequential Trisection

```
function seq_trisect_mon(G, H, a1, a2, mu, k)
% Finds all solutions of F(x) = G(x) - H(x) = 0
% in the N-box [a1, a2] where F, G, H, x, a1, and a2 are N-dim.
% vectors and all components of G and H are increasing inside
% [a1, a2].  G and H are vector fields.
% Uses recursive trisection and the monotonicity root condition:
% G(a1) <= H(a2) and H(a1) <= G(a2).
% k = axis along which we will be dividing.  ** Call seq_trisect_mon with k=1.

N=length(a1); % N = dimension of the space in which we are.

if  ((min(feval(G,a1)<= feval(H,a2)) == 1) &
                                    (min(feval(H,a1) <= feval(G,a2)) == 1))
        if norm(a2-a1, inf) < mu
                [a1; a2]
                return;
        end
        % a1 = a1
        c2 = a2;
        b2 = a2;
        b2(k) = a1(k)/3 + 2*a2(k)/3;
        b1 = a1;
        b1(k) = 2*a1(k)/3 + a2(k)/3;
        c1 = a1;
        c1(k)= a1(k)/3 + 2*a2(k)/3;
        a2(k) = 2*a1(k)/3 + a2(k)/3;
        if (k < N)
                k=k+1;
        else
                k=1;
        end
        seq_trisect_mon(G,H,a1,a2,mu,k)
        seq_trisect_mon(G,H,b1,b2,mu,k)
        seq_trisect_mon(G,H,c1,c2,mu,k)
end
```

Monotonicity Root Condition: Simultaneous Bisection

```
function simult_mon(G, H, a1, a2, mu)
% Finds all solutions of F(x) = G(x) - H(x) = 0
% in the N-box [a1, a2] where F, G, H, x, a1, and a2 are N-dim.
% vectors and all components of G and H are increasing inside
% [a1, a2].  G and H are vector fields.
% Uses recursive bisection and the monotonicity root condition:
% G(a1) <= H(a2) and H(a1) <= G(a2).
% ************ Bisects along all axes simultaneously!
% place = variable that marks the current position in a row of A
% trigger = 0 then fill kth position with a1(k)
%           1 then fill kth position with a1(k)+(a2(k)-a1(k))/2
% times = keeps track of the # of times one has progressed through the
%         i-loop.  Used for updating place.

N = length(a1); % N = dimension of the space in which we are.
global CELLS
global ROOTCHECKS

A = zeros(N, 2^(N+1));

for k=1:N
        place=0;
        times=0;
        trigger=0;
    while (place < (2^(N+1)))
        for i=1:2:(2^(N-k+1)-1)
                if (trigger == 0)
                        A(k, (i+place)) = a1(k);
                end
                if (trigger == 1)
                        A(k, (i+place)) = a1(k)+(a2(k)-a1(k))/2;
                end
        end
        trigger = mod(trigger+1, 2);
        times = times + 1;
        place = times*(2^(N-k+1));
    end
end

for i=2:2:2^(N+1)
        A(:,i) = A(:,(i-1)) + (a2-a1)'/2;
end
```

```
for i=1:2:(2^(N+1)-1) % Steps of 2 to hit all new cells which were formed.
        ROOTCHECKS = ROOTCHECKS + 1;
        if ((min(feval(G, A(:,i))<= feval(H, A(:, i+1))) == 1) &
                            (min(feval(H, A(:,i)) <=feval(G,A(:,i+1)))==1))
                if norm((A(:,i)-A(:,i+1)), inf) < mu
                          [A(:,i)'; A(:,i+1)'];
                          CELLS = CELLS +1;
                else
                        simult_mon(G, H, A(:,i)',A(:, i+1)',mu)
                end
        end
end
```

A.2 Java Code

Here we present Java code written by Dr. Kurt Georg. This is a black box algorithm which implements the improved Taylor series root condition for polynomial systems. The only input required is a system of polynomials entered in a standard way (see the following example) and an initial cell, Λ. Where appropriate, special features of this code are explained.

Example: Boon Problem

Here is the Boon problem from the field of neurophysiology which we consider in Section 4.3.

The domain of this problem is taken as

$$\Lambda = \prod_{i=1}^{5} [-2, 2]$$

where $F(x)$ is defined by

$$f_1(x) = x_5^2 + x_3^2 - 1$$

$$f_2(x) = x_6^2 + x_4^2 - 1$$

$$f_3(x) = x_1 x_3^3 + x_2 x_4^3 - 1.2$$

$$f_4(x) = x_1 x_5^3 + x_2 x_6^3 - 1.2$$

$$f_5(x) = x_1 x_3^2 x_5 + x_2 x_4^2 x_6 - .7$$

$$f_6(x) = x_1 x_3 x_5^2 + x_2 x_4 x_6^2 - .7$$

The following is the file "boon.poly" which represents this system in the standard way. Each line represents a monomial. The first column is the coefficient and the following columns are the powers of the variables which appear in the monomial. An equation is formed by summing the monomials which appear between the blank lines.

```
boon.poly
1 2 0 0 0 0 0
1 0 0 2 0 0 0
-1 0 0 0 0 0 0

1 0 2 0 0 0 0
1 0 0 0 2 0 0
-1 0 0 0 0 0 0

1 0 0 3 0 1 0
1 0 0 0 3 0 1
-1.2 0 0 0 0 0 0

1 3 0 0 0 1 0
1 0 3 0 0 0 1
-1.2 0 0 0 0 0 0

1 1 0 2 0 1 0
1 0 1 0 2 0 1
-.7 0 0 0 0 0 0

1 2 0 1 0 1 0
1 0 2 0 1 0 1
-.7 0 0 0 0 0 0
```

Driver

The following class is the main algorithm. It is the driver for the cell exclusion algorithm, i.e., this is the class which is compiled. In this class one enters the file containing the system of polynomials. Here this file is "boon.poly". See the preceding pages for this example file and an explanation. One also enters the midpoint and mesh vector for the initial cell, Λ. Currently, the midpoint is (0,0,0,0,0,0) and the mesh vector is (2,2,2,2,2,2).

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class ggg {
    public static void main(String[] args)
            throws IOException, java.io.FileNotFoundException {
        PolyCEAtest ex = new PolyCEAtest("boon.poly");
        CellExcl alg =
            new CellExcl(ex,
                         new cell(new double[] {0,0,0,0,0,0}, new
                        double[] {2,2,2,2,2,2}), 10);
        alg.go();
        int cnt[] = alg.getCellCnt();
        int len = cnt.length;
        for(int k=0;k<len;k++){
            System.out.println(cnt[k]);
        }
        Vector sol = alg.getSol();
        System.out.println("Number of Components :  " + sol.size());
        int tc = alg.getTotalTestCnt();
        System.out.println("Total number of tests :  " + tc);
        for(int k = 0; (k < sol.size()) && !sol.isEmpty(); k++){
            System.out.print((k+1) + ") ");
            System.out.println(((cell)(sol.elementAt(k))).toString());
//            ex.printValue(((cell)(sol.get(k))).getCenter());
        }
    }
}
```

Polynomial Methods

The following class contains methods which operate on polynomial systems. It reads a system of polynomials from a file, generates the derivatives needed for the improved Taylor series root condition (4.2), and performs the root condition.

We now give some background and explain how derivatives of polynomial systems are generated for the improved Taylor series root condition (4.2).

Given a polynomial of degree k, $p(x) = \sum_{|\alpha| \leq k} c_\alpha x^\alpha$, the natural dominating function to choose is $\hat{p}(x) = \sum_{|\alpha| \leq k} |c_\alpha| x^\alpha$. We note $p \prec_\infty \hat{p}$. The improved Taylor series root condition now reads

$$|p(m)| \leq \hat{p}(|m| + d) - \hat{p}(|m|) - \sum_{0<|\alpha|<q} (\partial^\alpha \hat{p}(|m|) - |\partial^\alpha p(m)|) d^\alpha \qquad \text{(A.1)}$$

for any $q > 0$.

The terms in the above sum are nonnegative, and often they are zero. Naturally, we do not wish to compute terms which are zero. This is the motivation for the following definition:

Definition A.2.1 (Monotone). *A polynomial p is monotone if and only if all non-zero coefficients of p have the same sign.*

The following lemmas are straightforward consequences of this definition:

Lemma A.2.1. *A polynomial p is monotone if and only if $\hat{p}(m) = |p(m)|$ for all positive m.*

Lemma A.2.2. *If p is monotone, then $\partial^\beta p$ is monotone for all β.*

Notice if p is monotone, then the root condition simplifies to

$$|p(m)| \leq \hat{p}(|m| + d) - \hat{p}(|m|).$$

Notice that Lemma A.2.1 is only relevant when dealing with positive cells. The case when all values in Λ are nonnegative is important. Often in systems with physical significance, variables only take on nonnegative values. For example, if the variables represent molarities in a chemistry equilibrium problem, they must be nonnegative.

The following recursion generates the multi-indices needed for the improved Taylor series root condition when we are considering an initial cell Λ with only nonnegative values.

Notation: $\alpha_k = k$th component of α, $\beta_k = k$th component of β

The recursion is started with $\alpha = (0, 0, \ldots, 0)$.

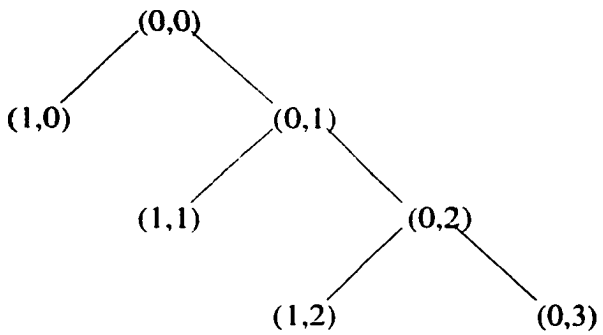
```
function GenerateMultiIndices(α)
set n = |α|
if ∂^α(p) is monotone
        return
print(α)
set β = α
set β_1 = β_1 + 1
GenerateMultiIndices(β)
for k = 1 : (n − 1)
        if α_k ≠ 0
            return
        set β = α
        set β_{k+1} = β_{k+1} + 1
        GenerateMultiIndices(β)
```

Now we consider the following example:

$$p = xy^2 + x - y^3 + 2xy + 3$$

The recursion will generate the following tree of derivatives which are necessary for the improved Taylor series root condition (4.2):

Derivative Tree



We also notice that for a polynomial $\hat{p}$ of degree k,

$$\hat{p}(|m| + d) = \hat{p}(|m|) + \sum_{0<\alpha\leq k} \partial^\alpha \hat{p}(|m|) d^\alpha.$$

Thus, for polynomial systems the root condition (A.1) simplifies to

$$|p(m)| \leq \sum_{0<|\alpha|\leq k} |\partial^\alpha p(m)| d^\alpha.$$

This root condition is valid for all m, not just $m \geq 0$. All relevant multi-indices can be obtained in a recursion similar to that above. The line "if $\partial^\alpha(p)$ is monotone" only needs to be replaced by "if $\partial^\alpha(p) = 0$". This latter recursion and root condition are used in the Java code.

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class polynomial{
    private static final DecimalFormat fmt1 = new DecimalFormat("0");
```

```
private static final double RelaxFac = 1. + 1.E-12; // to avoid numerical
                                                    //instabilities in test()
private monomial[] terms;
private int[] derivPattern;
private int numberOfTerms;
private int numberOfVar;
private polynomial[] next;
private int fac; // = factorial in Taylor Polynomial
private static double r_r1, r_r2; // for exclusion test
private static double[] m, d;     // of a current cell
public polynomial(Vector Poly){
    if (Poly.isEmpty()){
        return;
    }
    numberOfTerms = Poly.size();
    terms = new monomial[numberOfTerms];
    for(int k=0; k<numberOfTerms; k++){
        terms[k] = (monomial)(Poly.get(k));
        if(numberOfVar==0){
            numberOfVar = terms[k].getNumOfVar();
        }
        else{
            if(numberOfVar!=terms[k].getNumOfVar()){
                throw new Error("monomials have varying number of variables")
            }
        }
    }
    derivPattern = new int[numberOfVar];
    fac = 1;
    generateTree();
}
public polynomial(Vector Poly, int[] pat){
    if (Poly.isEmpty()){
        return;
    }
    numberOfTerms = Poly.size();
    terms = new monomial[numberOfTerms];
    for(int k=0; k<numberOfTerms; k++){
        terms[k] = (monomial)(Poly.get(k));
       if(numberOfVar==0){
            numberOfVar = terms[k].getNumOfVar();
        }
        else{
            if(numberOfVar!=terms[k].getNumOfVar()){
```

```
                    throw new Error("monomials have varying number of variables")
                }
            }
        }
        if (pat.length!=numberOfVar){
            throw new Error("derivative pattern does not fit number of variables"
        }
        derivPattern = pat;
        fac = 1;
        for(int k=0; k<numberOfVar; k++){
            for(int j=2; j<=pat[k]; j++){
                fac *=j;
            }
        }
    }
    public polynomial getDeriv(int i){
        Vector derPoly = new Vector();
        monomial p;
        for(int k=0; k<numberOfTerms; k++){
            p = terms[k];
            if(p.isNotConst(i)){
                derPoly.add(p.generateDeriv(i));
            }
        }
        if(derPoly.isEmpty()){
            return null;
        }
        int[] newDerivPattern = general.arrayCopy(derivPattern);
        newDerivPattern[i]++;
        return new polynomial(derPoly,newDerivPattern);
    }
    public String toString(){
        StringBuffer str = new StringBuffer();
        str.append("Polynomial has derivative pattern ");
        for(int k=0; k<numberOfVar; k++){
            str.append(general.padStr(fmt1.format(derivPattern[k]),2));
        }
        str.append(" and Taylor factor ");
        str.append(fac);
        for(int k=0; k<numberOfTerms; k++){
      str.append("\n");
            str.append(terms[k].toString());
        }
        return str.toString();
```

```
    }
    public double eval(double[] x){
        if(x.length!=numberOfVar){
            throw new Error("vector length different from number of variables");
        }
        double sm = 0.;
        for(int k=0; k< numberOfTerms; k++){
            sm += terms[k].eval(x);
        }
        return sm / fac;
    }
    public void generateTree(){
        Vector derPol = new Vector();
        polynomial pp;
        int len = 0;
        for(int k=0; k<numberOfVar; k++){
            if(k>0 && derivPattern[k-1]!=0){
                break;
            }
            pp = getDeriv(k);
            if(pp!=null){
                derPol.add(pp);
            }
        }
        if(!derPol.isEmpty()){
            len = derPol.size();
            next = new polynomial[len];
            for(int k=0; k<len; k++){
                next[k] = (polynomial)(derPol.get(k));
//                System.out.println(next[k].toString());
            }
        }
        for(int k=0; k<len; k++){
            (next[k]).generateTree();
        }
    }
    public boolean test(cell sig){
        m = sig.getCenter();

      d = sig.getMeshv();
        r_r1 = Math.abs(eval(m));
        r_r2 = 0.;
        accumulate(next);
        if(r_r1 <= r_r2*RelaxFac){
```

```
            return true;
        }
        else{
            return false;
        }
    }
    private void accumulate(polynomial[] pols){
        if(pols==null){
            return;
        }
        int len = pols.length;
        for(int k=0; k<len; k++){
            r_r2 += Math.abs(pols[k].eval(m)) * pols[k].dEval(d);
            accumulate(pols[k].next);
        }
    }
    private double dEval(double[] dd){
        if(dd.length!=numberOfVar){
            throw new Error("mesh vector length different from
                                number of variables");
        }
        double pr = 1.;
        for(int k=0; k<numberOfVar; k++){
            for(int j=0; j<derivPattern[k]; j++){
                pr *= dd[k];
            }
        }
        return pr;
    }
    public static Vector readPolySys(String fname)
            throws java.io.FileNotFoundException, java.io.IOException{
        BufferedReader in
            = new BufferedReader(new FileReader(fname));
        Vector polySys = new Vector();
        Vector eqn = new Vector();
        StringTokenizer t;
        String line;
        boolean eqnEnded = true;
        double coeff;
        monomial mon;
        int[] powers = new int[30];
        int[] index;
        int k;
        while((line = in.readLine()) != null){
```

```
                t = new StringTokenizer(line);
                if(t.hasMoreTokens()){
                    if(eqnEnded){
                        eqn = new Vector();
                        eqnEnded = false;
                    }
                    coeff = (new Double(t.nextToken())).doubleValue();
                    k = 0;
                    while(t.hasMoreTokens()){
                        powers[k] = Integer.parseInt(t.nextToken());
                        k++;
                    }
                    index = general.arrayCopy(powers, k);
                    mon = new monomial(coeff, index);
                    eqn.add(mon);
                }
                else{
                    if(!eqnEnded){
                        polySys.add(eqn);
                    }
                    eqnEnded = true;
                }
            }
            if(!eqnEnded){
                polySys.add(eqn);
            }
            in.close();
            return polySys;
        }
    }
```

Monomial Methods

The following class contains methods which operate on monomials. The method "generateDeriv" returns the derivative of a monomial. The method "eval" returns the value of the monomial evaluated at the N-dimensional point x.

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class monomial{
    private double coeff;
    private int[] index;
    private int len;
    private static final DecimalFormat fmt1 = new DecimalFormat("0");
//    private static final String fmt2str = ".0000000E0";
    private static final String fmt2str = "#####.########";
    private static final DecimalFormat fmt2 = new DecimalFormat(fmt2str);
    private static final int fmt2pad = fmt2str.length() + 3;
    public monomial(double c, int[] ind){
        index = ind;
        coeff = c;
        len = ind.length;
    }
    public int getNumOfVar(){
            return len;
    }
    public boolean isNotConst(int i){ // Is derivative with respect to i != 0?
        if(index[i]>0){
            return true;
        }
        else{
            return false;
        }
    }
    public monomial generateDeriv(int i){
        int[] newIndex = new int[len];
        for(int k=0; k<len; k++){
            newIndex[k] = index[k];
        }
        newIndex[i]--;
        return(new monomial(coeff*index[i],newIndex));
    }
```

```
    public String toString(){
        StringBuffer str = new StringBuffer();
        str.append("coeff: ");
        str.append(general.padStr(fmt2.format(coeff),fmt2pad));
        str.append("   exponents: ");
        for(int k=0; k<len; k++){
            str.append(general.padStr(fmt1.format(index[k]),2));
        }
        return str.toString();
    }
    public double eval(double[] x){
        if(x.length!=len){
        throw new Error("vector length does not agree with number of variables");
        }
        double pr = coeff;
        for(int k=0; k<len; k++){
            for(int j=0; j<index[k]; j++){
                pr *=x[k];
            }
        }
        return pr;
    }
}
```

Monitoring of Cells in Ω_k and Root Condition Checks

The following class contains methods which call root conditions. It monitors the number of cells which satisfy the root condition on each level and the number of root condition checks which have been performed.

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class CellExcl{
    private Vector sol;
    private int[] cell_cnt;
    private int total_cnt;
    private CeaTest t;
    private cell initCell;
    private int mxLev;
    // Constructor:
    public CellExcl(CeaTest test, cell sigma, int maximal_lev){
```

```
    sol = new Vector();
    cell_cnt = new int[maximal_lev+1];
    t = test;
    initCell = sigma;
    mxLev = maximal_lev;
}
public Vector getSol(){
    return sol;
}
public int[] getCellCnt(){
    return cell_cnt;
}
public int getTotalTestCnt(){
    return total_cnt;
}
// Main Procedure:
public void go(){
    go(initCell, 0, 0);
}
// Recursive CEA:
private void go(cell sig, int lev, int axis){
    // Discard?
    total_cnt++;
    if(!t.test(sig)){
        return;
    }
    // Count this cell for given level?
    if(axis==0){
        cell_cnt[lev]++;
    }
    // Collect Good Cell:
    if(lev == mxLev){
        int ln = sig.getLength();
        double[]  C = sig.getCenter();
        for(int k=0; k<ln; k++)
            System.out.print(C[k] + " ");
        System.out.println();
        sig.putInto(sol);
        return;
    }
    // Recursive Bisection:
    int ln = sig.getLength();
    cell[] newSig = sig.bisect(axis);
    if(axis==ln-1){
```

```
            axis = 0;
            lev++;
        }
        else{
            axis++;
        }
        go(newSig[0], lev, axis);
        go(newSig[1], lev, axis);
        return;
    }
}
```

Root Condition Implementation

The following class implements the improved Taylor series root condition (4.2) for polynomial systems.

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class PolyCEAtest implements CeaTest{
    private polynomial[] polyArray;
    int len;
    public PolyCEAtest(String fname) throws IOException,
                                        java.io.FileNotFoundException{
        Vector polyS = polynomial.readPolySys(fname);
        len = polyS.size();
        polyArray = new polynomial[len];
        for(int k=0; k<len; k++){
            polyArray[k] = new polynomial((Vector)(polyS.get(k)));
        }
    }
    public boolean test(cell sig){
        for(int k=0; k<len; k++){
            if(!polyArray[k].test(sig)){
                return false;
            }
        }
        return true;
    }
    public void printValue(double[] x){
        for(int k=0; k<len; k++){
            System.out.println(polyArray[k].eval(x));
        }
    }

}
```

Partitioning Cells and Linking Components

The following class contains methods which partition cells and link groups of cells into components.

A typical feature of a cell exclusion algorithm is that several cells which satisfy the root condition correspond to one zero. We wish to link all of the cells which represent a single zero into one component. This is possible since we consider problems with a finite number of isolated solutions. Two cells are considered to be close if their midpoints m_1 and m_2 satisfy $|m_1 - m_2| \leq C2^{-k}d$ on the kth level of bisection. Ideally $C = 2$, but practically we must choose C somewhat greater than 2 to link components properly. A larger C helps to account for the effects of round-off error. In the following code FAC $= C = 7.99$.

```
import java.lang.*;
import java.text.*;
import java.io.*;
import java.util.*;
public class cell{
    private static final double FAC = 7.99; // factor for measuring closeness
    private static String formatStr = " ##.00000000";
    private static final DecimalFormat FMT = new DecimalFormat(formatStr);
    private static final int FMTL = formatStr.length()+1; // for padding
                                                          // the above format
    private double[] center;
    private double[] meshv;
    private int len;
    public cell(double[] c, double[] d){
        center = c;
        meshv = d;
        len = c.length;
    }
    public double[] getCenter(){
        return center;
        }
    public double[] getMeshv(){
        return meshv;
        }
    public int getLength(){
```

```
    return len;
    }
public boolean isClose(cell a){
    double[] ac = a.getCenter();
    double[] ad = a.getMeshv();
    for(int k=0;k<len;k++){
        if( Math.abs(center[k] - ac[k])
               > FAC * (meshv[k] + ad[k])){
            return false;
        }
    }
    return true;
}
public cell[] bisect(int axis){
    cell[] out = new cell[2];
    double[] c0 = new double[len];
    double[] d0 = new double[len];
    double[] c1 = new double[len];
    for(int k = 0;k<len;k++){
        c0[k] = center[k];
        c1[k] = center[k];
        d0[k] = meshv[k];
    }
    d0[axis] = 0.5*d0[axis];
    c0[axis] = center[axis]-d0[axis];
    c1[axis] = center[axis]+d0[axis];
    out[0] = new cell(c0,d0);
    out[1] = new cell(c1,d0);
    return out;
}
public String toString(){
    StringBuffer str = new StringBuffer();
    str.append("center=");
        for(int k=0;k<len;k++){
        str.append(general.padStr(FMT.format(center[k]),FMTL));
    }
    str.append("\n meshv=");
        for(int k=0;k<len;k++){
        str.append(general.padStr(FMT.format(meshv[k]),FMTL));
    }
    return str.toString();
}
public void putInto(Vector sol){
    if(sol.isEmpty()){
```

```
                sol.addElement(this);
                return;
            }
            for(int k = 0;k < sol.size();k++){
                if(this.isClose((cell)(sol.elementAt(k)))){
                    ((cell)(sol.elementAt(k))).gulp(this);
                    for(int l=k+1;l<sol.size();){
                        if(this.isClose((cell)(sol.elementAt(l)))){
                            ((cell)(sol.elementAt(k))).gulp((cell)(sol.elementAt(l)))
                            sol.removeElementAt(l);
                        }
                        else{
                            l++;
                        }
                    }
                    return;
                }
            }
            sol.addElement(this);
            return;
        }
        public void gulp(cell tau){
            double[] tau_center = tau.getCenter();
            double[] tau_meshv = tau.getMeshv();
            double a, b;
            for(int k=0;k<len;k++){
                a = Math.min(center[k]-meshv[k],tau_center[k]-tau_meshv[k]);
                b = Math.max(center[k]+meshv[k],tau_center[k]+tau_meshv[k]);
                center[k] = 0.5*(a+b);
                meshv[k] = 0.5*(b-a);
            }
        }
    }
```

Interfaces

Here are the interfaces required for linking all of the classes within this cell exclusion algorithm.

```
public interface CeaTest{
    public boolean test(cell sig);
}

public interface SimpleFunction{
    public double f(double v);
}

public interface VectorField{
    public double[] vf(double[] v);
    public int getDimensionIn();
    public int getDimensionOut();
}
```