A Nonsmooth Exclusion Test for Finding All Solutions of Nonlinear Equations

by

Vinay Kumar

Bachelor of Technology in Electrical Engineering
Indian Institute of Technology, Kharagpur 2006

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author ....[illegible]..........................
School of Engineering
August 16, 2007

Certified by ....[illegible]..............................
Paul I. Barton
Lammot du Pont Professor of Chemical Engineering
Thesis Supervisor

Accepted by ....[illegible].....................
Jaime Peraire
Professor of Aeronautics and Astronautics
Codirector, Computation for Design and Optimization Program

Dedicated

to

Mummy & Papa

You were, are and always will be
a daily reminder of things
that are ideal in life.

A Nonsmooth Exclusion Test for Finding All Solutions of Nonlinear Equations

by

Vinay Kumar

Bachelor of Technology in Electrical Engineering

Indian Institute of Technology, Kharagpur 2006

Submitted to the School of Engineering
on August 16, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

A new approach is proposed for finding all solutions of systems of nonlinear equations with bound constraints. The zero finding problem is converted to a global optimization problem whose global minima with zero objective value, if any, correspond to all solutions of the initial problem. A branch-and-bound algorithm is used with McCormick's nonsmooth convex relaxations to generate lower bounds. An inclusion relation between the solution set of the relaxed problem and that of the original nonconvex problem is established which motivates a method to generate automatically reasonably close starting points for a local Newton-type method. A damped-Newton method with *natural level functions* employing the *restrictive monotonicity test* is employed to find solutions robustly and rapidly. Due to the special structure of the objective function, the solution of the convex lower bounding problem yields a nonsmooth root exclusion test which is found to perform better than earlier interval based exclusion tests. The Krawczyk operator based root inclusion and exclusion tests are also embedded in the proposed algorithm to refine the variable bounds for efficient fathoming of the search space. The performance of the algorithm on a variety of test problems from the literature is presented and for most of them the first solution is found at the first iteration of the algorithm due to the good starting point generation.

Thesis Supervisor: Paul I. Barton
Title: Lammot du Pont Professor of Chemical Engineering

Acknowledgments

I would like to express my heartfelt thanks to a number of individuals for my time at MIT as a student in general and their contribution to this thesis in particular.

First, I would like to thank my advisor, Paul I. Barton, for his expert guidance and persistent encouragement throughout the development of this work. Given the short time frame of this thesis, I cannot imagine a timely finish without his valuable suggestions and critical remarks. Working under his supervision has been a rewarding experience and I will forever be thankful to him for this opportunity.

My gratitude extends further to the expert team of researchers in the Process Systems Engineering Laboratory. Ajay Selot and Mehmet Yunt have extended a helping hand whenever needed, especially during the implementation, to interface those complex C++ and Fortran codes. I also thank Benoit Chachuat for the C++ source code for computing the McCormick's relaxations and Alexander Mitsos, for the DAEPACK related help for computing the interval extension of the Jacobian. Cannot forget to thank Patricio Ramirez for his help with Jacobian and for his humorous chat sessions. I am also grateful to Sinapore-MIT-Alliance (SMA) for their financial support and for putting so much effort to make the life of SMA graduate fellows enjoyable at MIT.

Furthermore, I would like to thank my friends Ramendra, Priyanka, Amit, Shashi, Yong Ning and Joline with whom I had some wonderful experiences and whose friendship I will value and cherish for life. Their good sense of humor and the fun that we all together had, has made my life at MIT a memorable one.

No words can describe the support and encouragement that my parents have extended during this tough time. They were, are and always will be a daily reminder of things that are ideal in life. I am equally grateful to my elder brother Vikas for his unconditional support and his strong belief in me all these years. They all have done the best jobs as a family and were always there to back me up through the ups and downs of life. I owe them more than I would be able to express.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Chapter 1

Introduction

One of the most challenging problems arising in many science and engineering applications is the solution of systems of nonlinear equations. This is expressed mathematically as finding $\mathbf{x} \in \mathbb{R}^n$ such that

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{1.1}$$

where $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the function describing a certain model. Also, quite often the model under consideration is only valid on a subset of $\mathbb{R}^n$, usually in an n dimensional box formed by the physical bounds on the variables $\mathbf{x}$. Such a box $\mathbb{X} \subset \mathbb{R}^n$ is described as

$$\mathbb{X} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{x}^L \leq \mathbf{x} \leq \mathbf{x}^U\} \tag{1.2}$$

where $\mathbf{x}^L \in \mathbb{R}^n$ and $\mathbf{x}^U \in \mathbb{R}^n$ are, respectively, the lower and upper bounds on $\mathbf{x}$. The algorithm presented in this thesis finds all solutions of (1.1) in a given box $\mathbb{X} \subset \mathbb{R}^n$ using an approach similar to a branch-and-bound method for global optimization.

A vast literature exists on techniques for solving systems of nonlinear equations. Many of the existing methods can be broadly classified into following three major headings:

1. Newton type methods,

2. Interval methods, and

3. Continuation methods.

In this chapter relevant theoretical background and literature reviews about each of them will be presented highlighting their roles, if any, in the branch-and-bound algorithm proposed in this thesis.

1.1 Newton-type Methods

All Newton type methods for solving a system of equations defined by (1.1) require computation of the *Newton direction* $\mathbf{d}^k$ for iteration k, given by the solution of following system of linear equations:

$$\mathbf{J}(\mathbf{x}^k)\mathbf{d}^k = -\mathbf{f}(\mathbf{x}^k) \tag{1.3}$$

where $\mathbf{x}^k$ is the estimate for the solution at iteration k and $\mathbf{J}(\mathbf{x}^k)$ is the Jacobian matrix of $\mathbf{f}$ evaluated at $\mathbf{x}^k$. Newton's method takes the full Newton step at iteration k, giving the next iterate as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{d}^k. \tag{1.4}$$

1.1.1 Local Convergence of Newton's Method

In general, Newton's method achieves a superlinear convergence but the convergence is confined locally [20]. Local convergence means that corresponding to each isolated solution $\mathbf{x}^*$ of (1.1) there exists a scalar $\epsilon > 0$ defining a neighborhood $N_\epsilon(\mathbf{x}^*)$ of $\mathbf{x}^*$ as

$$N_\epsilon(\mathbf{x}^*) = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{x} - \mathbf{x}^*|| < \epsilon\} \tag{1.5}$$

such that all starting points located in $N_\epsilon(\mathbf{x}^*)$ will generate a sequence of iterates converging to the solution $\mathbf{x}^*$. In the Euclidean norm $N_\epsilon(\mathbf{x}^*)$ is a hypersphere in $\mathbb{R}^n$ centered at $\mathbf{x}^*$, having radius ϵ. Moreover, often the step $\mathbf{d}^k$ is "too large" making Newton's method unstable, eventually leading to convergence failure. Another potential disadvantage is that this step may totally ignore the domain of admissible solutions, which is not desirable.

1.1.2 Damped-Newton Method

Attempts have been made to increase the neighborhood of convergence by use of step length control strategies such as *line search* which leads to the *damped-Newton method.* This, instead of taking the full Newton step, calculates a stepsize $\alpha^k \in (0,1]$ at each iteration and the next iterate is given by

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k. \tag{1.6}$$

The stepsize α^k is chosen to decrease an appropriate *merit* or *level function* relative to $\mathbf{x}^k$. The overall effect is that taking a smaller stepsize rather than full Newton step almost eliminates the instability problem with Newton's method.
A common choice for merit function $T(\mathbf{x})$ is the squared Euclidean norm of $\mathbf{f}(\mathbf{x})$:

$$T(\mathbf{x}) = ||\mathbf{f}(\mathbf{x})||_2^2 = \sum_{i=1}^{n} f_i^2(\mathbf{x}). \tag{1.7}$$

Line search obtains the stepsize α^k by solving the following one dimensional minimization problem:

$$\min_{\alpha^k \in (0,1]} T(\mathbf{x}) = \min_{\alpha^k \in (0,1]} ||\mathbf{f}(\mathbf{x}^k + \alpha^k \mathbf{d}^k)||_2^2. \tag{1.8}$$

Using nonlinear optimization theory it can be shown that with the proper choice of stepsize, the damped-Newton method will converge for any initial guess in the *level set*

$$N_\beta = \{\mathbf{x} \in \mathbb{R}^n : ||\mathbf{f}(\mathbf{x})||_2^2 \leq \beta\} \tag{1.9}$$

provided β is chosen such that:

1. N_β is a compact subset of the domain of $\mathbf{f}$,

2. $\mathbf{f}$ is twice continuously differentiable on an open set containing N_β, and

3. the Jacobian matrix $\mathbf{J}(\mathbf{x})$ is nonsingular on N_β.

It is anticipated that the set N_β is much larger than the neighborhood of convergence for Newton's method, and hence this is called *global convergence.*

1.1.3 Natural Level Functions

Computational experience shows that even for mildly ill-conditioned problems the damped-Newton method produces extremely small stepsizes leading to very slow convergence. As pointed out by Deuflhard [3] this happens because, for ill conditioned problems, the Newton direction

$$\mathbf{d}^k = -\mathbf{J}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x}^k) \tag{1.10}$$

and the steepest descent direction of the merit function (1.7)

$$-\boldsymbol{\nabla} T(\mathbf{x}^k) = -2\mathbf{J}(\mathbf{x}^k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}^k) \tag{1.11}$$

are almost orthogonal so that enforcing descent of the merit function leads to very small stepsizes. This can be verified by computing the cosine of the angle between the two directions as,

$$\cos(\mathbf{d}^k, -\boldsymbol{\nabla} T(\mathbf{x}^k)) = \frac{\mathbf{f}(\mathbf{x}^k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}^k)}{||\mathbf{J}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x}^k)||\,||\mathbf{J}(\mathbf{x}^k)^{\mathrm{T}}\mathbf{f}(\mathbf{x}^k)||} \geq \frac{1}{\mathrm{cond}(\mathbf{J}(\mathbf{x}^k))}. \tag{1.12}$$

It is highly probable that above expression for the cosine of the angle between $\mathbf{d}^k$ and $-\boldsymbol{\nabla} T(\mathbf{x}^k)$ attains its lower bound of $(\mathrm{cond}\ \mathbf{J}(\mathbf{x}^k))^{-1}$, explaining the slow convergence of the damped-Newton method. This observation motivated Deuflhard [3] to propose the following merit function

$$T_J(\mathbf{x}) = ||\mathbf{J}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x})||_2^2 \tag{1.13}$$

known as the *natural level function*, for which the steepest descent direction is parallel to the Newton direction, avoiding the orthogonality problem with the damped-Newton method. Moreover, this merit function is invariant under affine transformations and hence convergence, when it occurs, is fast.

1.1.4 Restrictive Monotonicity Test

As is evident from (1.13), the natural level function is changing at each iteration and so descent arguments can no longer be used to prove global convergence. Indeed, it is quite easy to construct a counterexample in which the iteration will just move back and forth between two points for ever without converging [1]. This led Bock *et al.* [2] to propose the *restrictive monotonicity test* (RMT) which essentially is an alternative stepsize selection strategy to exact or approximate line search using natural level functions. To formalize RMT following proposition is needed.

Proposition 1.1.1 (Quadratic Upper Bound). *If* $\mathbf{d}^k$ *is the Newton direction, then*

$$||\mathbf{J}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x}^k + \alpha\mathbf{d}^k)|| \leq \Big(1 - \alpha + \frac{\alpha^2}{2}\omega(\alpha)||\mathbf{d}^k||\Big)||\mathbf{J}(\mathbf{x}^k)^{-1}\mathbf{f}(\mathbf{x}^k)|| \qquad (1.14)$$

where,

$$\omega(\alpha) = \sup_{0<s\leq\alpha} \frac{||\mathbf{J}(\mathbf{x}^k)^{-1}\Big(\mathbf{J}(\mathbf{x}^k + s\mathbf{d}^k) - \mathbf{J}(\mathbf{x}^k)\Big)||}{s||\mathbf{d}^k||}. \qquad (1.15)$$

In light of proposition (1.1.1), if we choose a stepsize $0 < \alpha^k \leq 1$ such that

$$\alpha^k||\mathbf{d}^k|| \leq \min\Big(\frac{\eta}{\omega(\alpha^k)}, ||\mathbf{d}^k||\Big) \qquad (1.16)$$

for some $\eta < 2$, then a descent condition similar to the Armijo rule holds for natural level functions. (1.16) is known as Restrictive Monotonicity Test.

The Fortran subroutine (NWTSLV) implementing the damped-Newton method has been developed [24] that combines the RMT method with sparse linear algebra [4], making it suitable for large-scale problems. On a wide range of test problems, this RMT code has demonstrated a dramatic improvement in robustness over the previous codes based on Newton's method. Although in terms of speed of convergence RMT often takes a large number of steps, convergence is slow and steady rather than grinding to halt as is the case for the basic damped-Newton method in face of ill-conditioning.

Although these efforts have significantly enlarged the region of convergence, finding a close enough starting point still remains a nontrivial task, as real industrial problems are large, highly nonlinear and ill-conditioned and often exhibit a very small neighborhood of convergence. In practice, a large amount of time on projects is spent by engineers trying to get a suitable starting point using a variety of ad hoc strategies. An important contribution of the algorithm proposed in this thesis is the development of a reliable technique to generate automatically starting points which are in a sense reasonably close to the solution sought.

1.2 Interval methods

This section will provide a brief introduction to interval arithmetic, with emphasis on the aspects relevant to the nonlinear equation solving addressed in this thesis. For a more detailed and complete discussion the reader is referred to the classic literature on interval based methods by Neumaier [19].

A real *interval number*, or simply, an *interval* X can be defined by $X = [x^L, x^U] = \{x \in \mathbb{R} : x^L \leq x \leq x^U\}$, where $x^L, x^U \in \mathbb{R}$ and $x^L \leq x^U$. The set of all such real intervals is denoted by $\mathbb{IR}$. A real number $x \in \mathbb{R}$ can also be represented as a degenerate (or thin) interval $X = [x, x] \in \mathbb{IR}$. An interval vector is analogous to real vectors where real numbers are replaced by intervals. Thus, an interval vector represents an n dimensional box and is denoted by $\mathbb{X} = (X_i)_{1 \leq i \leq n} = (X_1, X_2, \ldots, X_n) \in \mathbb{IR}^n$ where, $X_i \in \mathbb{IR}, 1 \leq i \leq n$.

Some useful definitions related to intervals are enumerated below. In all the definitions an interval number is denoted by $X = [x^L, x^U]$ and an interval vector is denoted by $\mathbb{X} = (X_i)_{1 \leq i \leq n}$.

1. (Midpoint): For the interval number $X = [x^L, x^U]$ the mid-point is the number $\overline{x} \in \mathbb{R}$ such that $\overline{x} = (x^L + x^U)/2$. For an interval vector $\mathbb{X} = (X_i)_{1 \leq i \leq n} \in \mathbb{IR}^n$ it is $\overline{\mathbf{x}} \in \mathbb{R}^n$ such that $\overline{\mathbf{x}} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$.

2. (Width): The width of an interval number X is $w(X) \in \mathbb{R}$ defined as $w(X) = x^U - x^L$. For an interval vector $\mathbb{X}$, width $w(\mathbb{X}) \in \mathbb{R}$, is $w(\mathbb{X}) = \max_{1 \le i \le n} w(X_i)$.

3. (Absolute Value): The absolute value $|X| \in \mathbb{R}$ of an interval is $|X| = \max(|x^L|, |x^U|)$.

1.2.1 Interval Arithmetic

Since interval analysis treats intervals as numbers, arithmetic operators can be defined with these numbers as an extension of real arithmetic. For two intervals $X = [x^L, x^U]$ and $Y = [y^L, y^U] \in \mathbb{IR}$ the elementary interval arithmetic operations $op \in \{+, -, \times, \div\}$ are defined as

$$X \; op \; Y = \{x \; op \; y : x \in X, y \in Y\}. \tag{1.17}$$

This leads to following formulae for elementary interval operations in terms of the corresponding end points

$$\begin{aligned}
X + Y &= [x^L + y^L, x^U + y^U],\\
X - Y &= [x^L - y^U, x^U - y^L],\\
X \times Y &= [\min(x^L y^L, x^L y^U, x^U y^L, x^U y^U), \max(x^L y^L, x^L y^U, x^U y^L, x^U y^U)],\\
X \div Y &= [x^L, x^U] \times [1/y^U, 1/y^L], 0 \notin [y^L, y^U].
\end{aligned}$$

When these operations are performed on a computer, rounding problems are likely to arise in exact computations of the end points. Steps can be taken to ensure that the result is a superset and not a subset of the accurate result. This is done by using rounded-interval arithmetic [19][Chapter 1].

1.2.2 Interval Valued Functions

Given a real-valued continuous function $f : \mathbb{R}^n \to \mathbb{R}$ the image of an interval $\mathbb{X} \in \mathbb{IR}^n$ is defined as,

$$f(\mathbb{X}) = \{f(\mathbf{x}) : \mathbf{x} \in \mathbb{X}\} = \left[\min_{\mathbf{x} \in \mathbb{X}} f(\mathbf{x}), \max_{\mathbf{x} \in \mathbb{X}} f(\mathbf{x})\right] \tag{1.18}$$

which is an interval. Hence this mapping can be viewed as an interval-valued mapping $F : \mathbb{IR}^n \longrightarrow \mathbb{IR}$. Note that f being a continuous function optimized over a compact set $\mathbb{X}$ guarantees the existence of max and min in (1.18) and also that every value between the two extrema is attained. However, exact computation of the RHS in (1.18) requires solving two global optimization problem which cannot in general be done with a finite computation. Hence, computationally inexpensive techniques are used to obtain an estimate of $f(\mathbb{X})$ using interval analysis. An example of an interval valued function that is cheap to compute is a *rational interval function* whose interval values are defined by a specific finite sequence of interval arithmetic operations.

Inclusion Function

An interval valued function $F : \mathbb{IR}^n \longrightarrow \mathbb{IR}$ is an inclusion function for $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ over an interval $\mathbb{X} \subset \mathbb{R}^n$ if

$$f(\mathbb{Z}) \subset F(\mathbb{Z}), \forall \mathbb{Z} \in \mathbb{IR}^n : \mathbb{Z} \subset \mathbb{X}. \tag{1.19}$$

Hence, the interval valued inclusion function evaluated at $\mathbb{Z}$ contains the image of $\mathbb{Z}$ under f for all $\mathbb{Z} \subset \mathbb{X}$.

Inclusion Monotonic Function

An interval valued mapping $F : \mathbb{IR}^n \longrightarrow \mathbb{IR}$ is *inclusion monotonic* if

$$Y_i \subset X_i, \ \forall\ i = 1, 2, 3, \ldots, n \Rightarrow F(\mathbb{Y}) \subset F(\mathbb{X}) \tag{1.20}$$

i.e., the interval value of a subset is a subset of the interval value of the host set. The image function f is inclusion monotonic, as are all the interval arithmetic operators (as they are images), and rational interval functions by finite induction. However not all interval valued functions are inclusion monotonic.

1.2.3 Interval Extensions of Functions

Given a function $f : \mathbb{R}^n \to \mathbb{R}$ its *interval extension* is an interval valued function $F : \mathbb{IR}^n \to \mathbb{IR}$ with the property

$$f(\mathbf{x}) = \mathbf{y} = [\mathbf{y}, \mathbf{y}] = F([\mathbf{x}, \mathbf{x}]), \forall \mathbf{x} \in \mathbb{R}^n \tag{1.21}$$

where the interval valued function F is evaluated at the degenerate interval $[\mathbf{x}, \mathbf{x}]$ corresponding to the point $\mathbf{x}$. (The domain and property may be restricted to $\mathbb{X} \subset \mathbb{R}^n$). It is noteworthy that there is not a unique interval extension for a given function. Also, if $F : \mathbb{IR}^n \to \mathbb{IR}$ is an inclusion monotonic interval extension of $f : \mathbb{R}^n \to \mathbb{R}$, then

$$f(\mathbb{X}) \subset F(\mathbb{X}), \forall \mathbb{X} \in \mathbb{IR}^n. \tag{1.22}$$

Hence, inclusion monotonic interval extensions are of particular interests in interval analysis.

Natural Interval Extension

One of the easiest ways to compute an inclusion monotonic interval extension of real rational functions is by the *natural interval extension* which is obtained simply by replacing $\mathbf{x}$ by the interval $\mathbb{X}$ and the elementary real operations by the corresponding interval arithmetic operations. Also, if a unary continuous intrinsic function $\phi(x)$ appears in the sequence of elementary operations, the image of any interval X is given by

$$\Phi(X) = \{\phi(x) : x \in X\} = \left[\min_{x \in X} \phi(x), \max_{x \in X} \phi(x)\right]. \tag{1.23}$$

For most of the intrinsic functions supported by compilers, the min and max are easy to compute and so the image can be used in natural interval extensions. For instance, for a monotonically increasing function (e.g., $\exp(x), \log(x), \sqrt{x}$) if $X = [x^L, x^U]$

$$\Phi(X) = \{\phi(x) : x \in X\} = [\phi(x^L), \phi(x^U)] \tag{1.24}$$

and an obvious result also holds for monotonically decreasing functions. For a positive integer p, exponentiation of the interval $X = [x^L, x^U]$ is defined as

$$X^p = \begin{cases} [(x^L)^p, (x^U)^p] & \text{if } x^L > 0 \text{ or } p \text{ is odd} \\ [(x^U)^p, (x^L)^p] & \text{if } x^U < 0 \text{ and } p \text{ is even} \\ [0, |X|^p] & \text{if } 0 \in X \text{ and } p \text{ is even.} \end{cases} \tag{1.25}$$

It is not hard to confirm that this definition yields the exact range of the functions whose interval extensions are computed. However, this is not true in general. In fact, the interval extensions $F(\mathbb{X})$ encloses all values of $f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{X}$ but the quality (tightness) of this enclosure depends on the form in which $F(\mathbb{X})$ is expressed and evaluated. For example, consider the function $f(\mathbf{x}) = x_1(x_2 - x_3), \mathbf{x} \in \mathbb{R}^3$ defined over the three-dimensional box $\mathbb{X} = X_1 \times X_2 \times X_3 \in \mathbb{IR}^3$. The natural interval extension is $F(\mathbb{X}) = X_1 \times (X_2 - X_3)$ which for $X_1 = X_2 = X_3 = [1, 2]$ evaluates to [-2,2] and is precisely the image of $\mathbb{X}$ under f. However, expressing the same function as $f(\mathbf{x}) = (x_1x_2 - x_1x_3)$ results in the natural interval extension $F(\mathbb{X}) = X_1 \times X_2 - X_1 \times X_3$ evaluating to [-3,3] for the same box and so is an overestimate of the image of $\mathbb{X}$ under f. Such overestimations usually happen when an interval variable occurs more than once in an expression. This is called the *dependence problem* and occurs because interval arithmetic essentially treats each occurrence of a variable independently rather than recognizing their dependencies. Natural interval extensions are widely used to approximate (overestimate) the range of real-valued rational functions on a given box and it will be seen later that they serve as a key component in evaluation of the *convex relaxation* of functions.

1.2.4 Interval-Newton Operator

For an interval box $\mathbb{X} \subset \mathbb{R}^n$, a point $\mathbf{x} \in \mathbb{X}$ and a continuously differentiable function $\mathbf{f} : \mathbb{X} \rightarrow \mathbb{R}^n$, the interval-Newton operator $\mathbb{N}(\mathbf{x}, \mathbf{f}, \mathbb{X})$ is defined by the following system of linear interval equations:

$$\mathbb{J}(\mathbf{f},\mathbb{X})\Big(\mathbb{N}(\mathbf{x},\mathbf{f},\mathbb{X})-\mathbf{x}\Big)=-\mathbf{f}(\mathbf{x}) \tag{1.26}$$

where $\mathbb{J}(\mathbf{f},\mathbb{X})$ is the interval extension of the Jacobian matrix of $\mathbf{f}$ over $\mathbb{X}$. It can be shown [17] that a solution of $\mathbf{f}(\mathbf{x})=\mathbf{0}$ in $\mathbb{X}$, if any, will also be contained in $\mathbb{N}(\mathbf{x},\mathbf{f},\mathbb{X})$. This suggests the iteration

$$\mathbb{X}^{k+1}=\mathbb{X}^{k}\cap\mathbb{N}(\mathbf{x}^{k},\mathbf{f},\mathbb{X}^{k}), \tag{1.27}$$

known as the *interval-Newton iteration.* Different interval-Newton methods differ in the way $\mathbb{N}(\mathbf{x}^{k},\mathbf{f},\mathbb{X}^{k})$ is determined from equation (1.26) and thus in the tightness with which the solution set of (1.1) is enclosed in $\mathbb{N}(\mathbf{x}^{k},\mathbf{f},\mathbb{X}^{k})$. Schnepper and Stadtherr [22], for example, computed $\mathbb{N}^{k}(\mathbf{x},\mathbf{f},\mathbb{X})$ component by component using an interval Gauss-Seidel-like procedure. Various kinds of preconditioning is also done to obtain a tighter enclosure.

Root Inclusion and Exclusion Tests

While the iteration scheme given by (1.27) can be used to enclose a solution tightly, what is most significant is its power to provide an existence and uniqueness test popularly known as a *root inclusion test* [19]. It states that if $\mathbb{N}(\mathbf{x}^{k},\mathbf{f},\mathbb{X}^{k})\subset\mathbb{X}^{k}$ then $\mathbb{X}^{k}$ contains *exactly one* solution of $\mathbf{f}(\mathbf{x})=\mathbf{0}$ and furthermore Newton's method with real arithmetic will converge to that solution starting from any point in $\mathbb{X}^{k}$. Also if $\mathbb{N}(\mathbf{x}^{k},\mathbf{f},\mathbb{X}^{k})\cap\mathbb{X}^{k}=\emptyset$ then there is no root in $\mathbb{X}^{k}$. This so called *root exclusion test* is another significant result of the interval-based method which helps fathom a large part of the search space in the interval-Newton/generalized bisection method. However, If none of the two holds then no conclusion can be drawn and the inclusion test could be repeated on the next interval-Newton iterate $\mathbb{X}^{k+1}$, assuming it to be sufficiently smaller than $\mathbb{X}^{k}$. Or else, one could also bisect $\mathbb{X}^{k+1}$ and repeat the inclusion test on the resulting interval. This is the basic idea of the interval-Newton/generalized bisection method. Assuming that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ has finite number of real solutions in specified initial box, a properly implemented interval-Newton/generalized bisection method can find with mathematical certainty any and all such solutions to any pre-specified tolerance, or can determine that there is no solution in the given box [7].

1.2.5 Krawczyk's Operator

Although the interval-Newton operator has the potential to produce the tightest enclosures of solutions, its computation is often cumbersome due to the invertibility requirements for an interval matrix. If the interval extension of the Jacobian $\mathbb{J}(\mathbf{f}, \mathbb{X})$ contains a singular matrix, the interval-Newton operator $\mathbb{N}(\mathbf{x}, \mathbf{f}, \mathbb{X})$ becomes unbounded and lot of preconditioning and other strategies are required to avoid the involved computational instability. This motivated Krawczyk [19] to derive the following interval operator known as *Krawczyk's operator*

$$\mathbb{K}(\mathbf{x}, \mathbf{f}, \mathbb{X}) = \mathbf{x} - \mathbf{Y}\mathbf{f}(\mathbf{x}) + (\mathbf{I} - \mathbf{Y}\mathbb{J}(\mathbf{f}, \mathbb{X}))(\mathbb{X} - \mathbf{x}) \tag{1.28}$$

where, $\mathbf{Y} \in \mathbb{R}^{n \times n}$ is a linear isomorphism used for preconditioning, $\mathbf{I} \in \mathbb{R}^{n \times n}$ is a $n \times n$ identity matrix, and $\mathbb{J}(\mathbf{f}, \mathbb{X})$ is an interval extension of the Jacobian matrix of $\mathbf{f}$ over $\mathbb{X}$.

As per Neumaier [19] the invertibility of the interval matrix is avoided in Krawczyk's operator at the cost of a relaxed enclosure of solutions compared to the interval-Newton operator. Nevertheless, similar root inclusion and exclusion tests hold for the enclosures obtained using Krawczyk's operator, i.e., if $\mathbb{K}(\mathbf{x}, \mathbf{f}, \mathbb{X}) \subset \mathbb{X}$ then there is a unique zero of $\mathbf{f}(\mathbf{x})$ in $\mathbb{X}$ (Krawczyk root inclusion test). Furthermore, solutions of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in $\mathbb{X}$, if any, are contained in the intersection $\mathbb{K}(\mathbf{x}, \mathbf{f}, \mathbb{X}) \cap \mathbb{X}$ and if this intersection is the empty set ($\emptyset$) then no root is contained in $\mathbb{X}$ (Krawczyk root exclusion test). For the inclusion test an ideal pre-conditioner matrix $\mathbf{Y}$ is the inverse of the Jacobian matrix evaluated at the solution. However, in the interval type methods, the solution is not known a priori and hence $\mathbf{Y}$ is usually approximated by taking the inverse of the midpoint of the interval matrix $\mathbb{J}(\mathbf{f}, \mathbb{X})$. In the proposed algorithm, once a solution is found by a point Newton-type method, the Krawczyk operator is used only to check the uniqueness of the obtained solution in the present box $\mathbb{X}$. Hence, excellent preconditioning is achieved by using the inverted Jacobian matrix at a solution, making the inclusion test quite effective. Also, since $\mathbb{K}(\mathbf{x}, \mathbf{f}, \mathbb{X})$ is evaluated at a solution point the second term in equation (1.28) vanishes leaving the following simplified form:

$$\mathbb{K}(\mathbf{x}, \mathbf{f}, \mathbb{X}) = \mathbf{x} + (\mathbf{I} - \mathbf{Y}\mathbb{J}(\mathbf{f}, \mathbb{X}))(\mathbb{X} - \mathbf{x}). \tag{1.29}$$

If the root inclusion test is positive the current box can be fathomed based on the uniqueness result. Moreover, the intersection relation itself helps to fathom a good part of the search space not containing any solution. For the exclusion test, the inverse of mid-point of the Jacobian interval matrix $\mathbb{J}(\mathbf{f}, \mathbb{X})$ is used as the pre-conditioner $\mathbf{Y}$.

1.3 Continuation Methods

Another important class of methods for solving systems of nonlinear equations is *continuation* or *homotopy continuation methods.* This is an *incremental loading* type of method where a single parameter (say t) family of problems is created such that the solution for ($t = 0$) is known. Starting from $t = 0$, a sequence of problems is solved with t being incremented in small steps untill $t = 1$, when the solution sought is obtained.

For illustration consider the system of equations defined in (1.1). Embedding it into a convex linear global homotopy gives:

$$\mathbf{H}(\mathbf{x}, t) = t\mathbf{f}(\mathbf{x}) + (1 - t)\mathbf{g}(\mathbf{x}) = \mathbf{0} \tag{1.30}$$

Where $t \in \mathbb{R}$ is the scalar homotopy parameter, $\mathbf{H} : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n$, and $\mathbf{g} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a vector function selected such that the solution to $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ is known or easily determined.

The above choice of $\mathbf{g}(\mathbf{x})$ helps because solving $\mathbf{H}(\mathbf{x}, 0) = \mathbf{0}$ is the same as solving

Table 1.1: Different homotopies for various choices of $\mathbf{g}(\mathbf{x})$.

Homotopy	$\mathbf{g}(\mathbf{x})$	$\mathbf{H}(\mathbf{x}, t)$
Newton homotopy	$\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}^0)$	$\mathbf{f}(\mathbf{x}) - (1-t)\mathbf{f}(\mathbf{x}^0)$
Fixed-point homotopy	$\mathbf{x} - \mathbf{x}^0$	$t\mathbf{f}(\mathbf{x}) + (1-t)(\mathbf{x} - \mathbf{x}^0)$
Scale-invariant affine homotopy	$\nabla\mathbf{f}(\mathbf{x}^0)(\mathbf{x} - \mathbf{x}^0)$	$t\mathbf{f}(\mathbf{x}) + (1-t)\nabla\mathbf{f}(\mathbf{x}^0)(\mathbf{x} - \mathbf{x}^0)$

$\mathbf{g}(\mathbf{x}) = \mathbf{0}$ and solutions of $\mathbf{H}(\mathbf{x}, 1) = \mathbf{0}$ correspond to the solutions of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Hence, by incrementing t in small steps starting from $t = 0$ an $\mathbf{x} - t$ curve satisfying equation (1.30) is obtained and its points of intersection with $t = 1$ gives the solutions of (1.1) sought. Three main choices of $\mathbf{g}(\mathbf{x})$ discussed by Wayburn and Seader [10] lead to three respective homotopies as listed in Table 1.1 ($\mathbf{x}^0$ is the given starting point). Seader *et al.* [23, 8, 6] have successfully applied homotopy based continuation methods for solving nonlinear equations. Although this method is frequently used to find multiple solutions of arbitrary systems of nonlinear equations, mathematical guarantees that all solutions will be found exist only in some special cases. Furthermore, an important drawback of the continuation-based methods is that variable bounds cannot be handled directly and hence they have limited application for the class of bound constrained problems addressed in this thesis.

1.4 Thesis Summary

In recent years, a global optimization reformulation of the root finding problem has been used successfully by Maranas and Floudas [15] to find all solutions of systems of nonlinear equations. These ideas exploit developments in the deterministic global optimization literature. In the proposed algorithm, an approach similar to [15] is used and the original problem is reformulated to a nonconvex optimization problem with the objective function taken as a suitable norm of $\mathbf{f}(\mathbf{x})$. Chapter 2 will detail this global optimization reformulation and develop relevant theory leading to the proposed branch-and-bound algorithm. The algorithm will be formally stated in Chapter 3 with a brief discussion on its implementation. Test results of the performance of algorithm on a variety of problems from the literature are presented and analyzed in Chapter 4. The thesis ends with concluding remarks and a brief discussion on the scope of future work in Chapter 5.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Global Optimization Formulation

The root finding problem described by (1.1) can be reformulated as the following global optimization problem over the admissible domain $\mathbb{X} \subset \mathbb{R}^n$

$$\min_{\mathbf{x}\in\mathbb{X}} ||\mathbf{f}(\mathbf{x})|| \tag{2.1}$$

where $||.||$ can be any vector norm. Numerical considerations dictate that the 1-norm is the best choice in the presented algorithmic framework, as will be discussed later in this Chapter. This results in the following optimization problem:

$$\min_{\mathbf{x}\in\mathbb{X}} \sum_{i=1}^{n} |f_i(\mathbf{x})|. \tag{2.2}$$

If the optimal value of the above optimization problem is zero, any corresponding $\mathbf{x}$ will be a solution of (1.1). In fact, the global optimal solutions with zero optimal value of (2.2) are equivalent to the admissible solution set of (1.1). However, in general (2.2) is a nonconvex program and a local solver cannot guarantee convergence to a global optimal solution. Hence, for such problems, one strategy is to construct a *convex relaxation* of the objective function and solve the resulting convex program to generate a lower bound on the global optimal value. Likewise, for nonconcave maximization problems there is the concept of a *concave relaxation* of the objective function used to upper bound the global maximum. Convex and concave relaxations are quite commonly used in the global optimization algorithms and given an elementary function, a variety of techniques are known for constructing these relaxations as elaborated in the section that follows.

2.1 Convex and Concave Relaxations

Let $C \subset \mathbb{R}^n$ be any convex set. The *convex relaxation* of a function $f : C \to \mathbb{R}$ on C is a convex function $u : C \to \mathbb{R}$ such that

$$u(\mathbf{x}) \leq f(\mathbf{x}), \forall \mathbf{x} \in C. \tag{2.3}$$

Analogously, the *concave relaxation* of $f : C \to \mathbb{R}$ on C is a concave function $o : C \to \mathbb{R}$ such that

$$o(\mathbf{x}) \geq f(\mathbf{x}), \forall \mathbf{x} \in C. \tag{2.4}$$

Hence the convex relaxation of a function is a pointwise underestimator on its domain of definition and the concave relaxation is a pointwise overestimator.

Minimizing the convex relaxation on C will yield a lower bound on the minimum and/or infimum of f on C and maximizing the concave relaxation on C provides an upper bound on the maximum and/or supremum of f on C. Both the minimization of a convex relaxation and maximization of a concave relaxations are convex programs and hence are of particular interest in global optimization algorithms. The quality of these bounds will depend on how tight the relaxations are. Exact bounds are provided by the so called *convex and concave envelopes* of f on C.

2.1.1 Convex and Concave Envelopes

The convex envelope $\check{f}_C(\mathbf{x}) : C \to \mathbb{R}$ of a function $f : C \to \mathbb{R}$ is the pointwise supremum of all possible convex relaxations of f on C and the concave envelope $\hat{f}_C(\mathbf{x}) : C \to \mathbb{R}$ of $f : C \to \mathbb{R}$ is the pointwise infimum of all possible concave relaxations of f on C. The convex envelope $\check{\phi}_X(x)$ of any univariate concave function $\phi(x)$ on an interval $X = [x^L, x^U]$ is given by the secant to the univariate concave function at the endpoints of the interval. Mathematically,

$$\check{\phi}_X(x) = \phi(x^L) + \frac{\phi(x^U) - \phi(x^L)}{x^U - x^L}(x - x^L). \tag{2.5}$$

In fact, the convex envelope of any multivariate concave function on a simplex is given by the affine function that coincides with the multivariate concave function at all the vertices of the simplex. However, in general the convex envelope of a multivariate concave function on a multi-dimensional interval is not an affine function. Bilinear functions are a frequently occurring nonconvex term in optimization problems. The convex and concave envelopes of a bilinear function $\phi(x, y) = xy$ on the interval $X \times Y$ are given, respectively, by the pointwise maximum and minimum of two planes as follows:

$$\check{\phi}_{X\times Y}(x, y) = \max\{y^L x + x^L y - x^L y^L, y^U x + x^U y - x^U y^U\} \tag{2.6}$$

$$\hat{\phi}_{X\times Y}(x, y) = \min\{y^L x + x^U y - x^U y^L, y^U x + x^L y - x^L y^U\} \tag{2.7}$$

2.1.2 Relaxations of the Sum & Difference of Two Functions

Given the convex and concave relaxations of two functions $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ on a nonempty convex set $C \subset \mathbb{R}^n$ it is possible to construct convex and concave relaxations for their sum, difference and product over C. Suppose that convex functions $c_1^u(\mathbf{x})$, $c_2^u(\mathbf{x})$ and concave functions $c_1^o(\mathbf{x})$, $c_2^o(\mathbf{x})$ on C satisfying

$$c_1^u(\mathbf{x}) \leq f_1(\mathbf{x}) \leq c_1^o(\mathbf{x}), \ \forall \mathbf{x} \in C \tag{2.8}$$

$$c_2^u(\mathbf{x}) \leq f_2(\mathbf{x}) \leq c_2^o(\mathbf{x}), \ \forall \mathbf{x} \in C \tag{2.9}$$

are known. The results for the relaxations of the sum and difference of the two functions follow immediately as:

$$u_+(\mathbf{x}) \equiv c_1^u(\mathbf{x}) + c_2^u(\mathbf{x}) \leq f_1(\mathbf{x}) + f_2(\mathbf{x}) \leq c_1^o(\mathbf{x}) + c_2^o(\mathbf{x}) \equiv o_+(\mathbf{x}), \ \forall \mathbf{x} \in C \tag{2.10}$$

$$u_-(\mathbf{x}) \equiv c_1^u(\mathbf{x}) - c_2^o(\mathbf{x}) \leq f_1(\mathbf{x}) - f_2(\mathbf{x}) \leq c_1^o(\mathbf{x}) - c_2^u(\mathbf{x}) \equiv o_-(\mathbf{x}), \ \forall \mathbf{x} \in C. \tag{2.11}$$

It can be easily seen that functions $u_+(\mathbf{x})$ and $u_-(\mathbf{x})$, being the sum of two convex functions, are convex and $o_+(\mathbf{x})$ and $o_-(\mathbf{x})$, being the sum of two concave functions, are concave. The inequality relations in (2.10) confirm that $u_+(\mathbf{x})$ and $o_+(\mathbf{x})$ are valid convex and concave relaxations, respectively, for the sum of the two functions over C. By symmetrical arguments, in light of (2.11), $u_-(\mathbf{x})$ and $o_-(\mathbf{x})$ are valid convex and concave relaxations, respectively, for the difference of the two functions.

2.1.3 Relaxations of the Product of Two Functions

To compute a relaxation of the product $f_1(\mathbf{x})f_2(\mathbf{x})$ on C in addition to $c_1^u(\mathbf{x})$, $c_2^u(\mathbf{x})$ and $c_1^o(\mathbf{x})$, $c_2^o(\mathbf{x})$ satisfying (2.8) and (2.9) we also need numbers $f_1^L, f_1^U, f_2^L, f_2^U$ such that

$$C \subset \{\mathbf{x} : f_1^L \leq f_1(\mathbf{x}) \leq f_1^U, f_2^L \leq f_2(\mathbf{x}) \leq f_2^U\}. \tag{2.12}$$

From the convex envelope of a bilinear function it is known that

$$f_1(\mathbf{x})f_2(\mathbf{x}) \geq \max\{f_2^L f_1(\mathbf{x}) + f_1^L f_2(\mathbf{x}) - f_1^L f_2^L, f_2^U f_1(\mathbf{x}) + f_1^U f_2(\mathbf{x}) - f_1^U f_2^U\}, \ \forall \mathbf{x} \in C. \tag{2.13}$$

Now defining,

$$\begin{aligned}
\alpha_1(\mathbf{x}) &\equiv \min\{f_2^L c_1^u(\mathbf{x}), f_2^L c_1^o(\mathbf{x})\} \\
\alpha_2(\mathbf{x}) &\equiv \min\{f_1^L c_2^u(\mathbf{x}), f_1^L c_2^o(\mathbf{x})\} \\
\beta_1(\mathbf{x}) &\equiv \min\{f_2^U c_1^u(\mathbf{x}), f_2^U c_1^o(\mathbf{x})\} \\
\beta_2(\mathbf{x}) &\equiv \min\{f_1^U c_2^u(\mathbf{x}), f_1^U c_2^o(\mathbf{x})\},
\end{aligned}$$

it can be verified that the functions $\alpha_1(\mathbf{x}), \alpha_2(\mathbf{x}), \beta_1(\mathbf{x})$ and $\beta_2(\mathbf{x})$ are convex on C. Also, it follows from (2.13) that

$$f_1(\mathbf{x})f_2(\mathbf{x}) \geq \max\{\alpha_1(\mathbf{x}) + \alpha_2(\mathbf{x}) - f_1^L f_2^L, \beta_1(\mathbf{x}) + \beta_2(\mathbf{x}) - f_1^U f_2^U\}, \ \forall \mathbf{x} \in C. \tag{2.14}$$

Each argument in the max function on the RHS of (2.14) is convex and the maximum of two convex functions is convex making it a valid convex relaxation of $f_1(\mathbf{x})f_2(\mathbf{x})$

on C. The concave relaxation is obtained from the concave envelope of the bilinear function in a similar manner:

$$f_1(\mathbf{x})f_2(\mathbf{x}) \leq \min\{f_2^L f_1(\mathbf{x}) + f_1^U f_2(\mathbf{x}) - f_1^U f_2^L, f_2^U f_1(\mathbf{x}) + f_1^L f_2(\mathbf{x}) - f_1^L f_2^U\}, \ \forall \mathbf{x} \in C. \tag{2.15}$$

Again defining,

$$\begin{aligned}
\gamma_1(\mathbf{x}) &\equiv \max\{f_2^L c_1^u(\mathbf{x}), f_2^L c_1^o(\mathbf{x})\} \\
\gamma_2(\mathbf{x}) &\equiv \max\{f_1^U c_2^u(\mathbf{x}), f_1^U c_2^o(\mathbf{x})\} \\
\delta_1(\mathbf{x}) &\equiv \max\{f_2^U c_1^u(\mathbf{x}), f_2^U c_1^o(\mathbf{x})\} \\
\delta_2(\mathbf{x}) &\equiv \max\{f_1^L c_2^u(\mathbf{x}), f_1^L c_2^o(\mathbf{x})\}
\end{aligned}$$

the functions $\gamma_1(\mathbf{x}), \gamma_2(\mathbf{x}), \delta_1(\mathbf{x})$ and $\delta_2(\mathbf{x})$ can be shown to be concave on C and it follows from (2.15) that

$$f_1(\mathbf{x})f_2(\mathbf{x}) \leq \min\{\gamma_1(\mathbf{x}) + \gamma_2(\mathbf{x}) - f_1^U f_2^L, \delta_1(\mathbf{x}) + \delta_2(\mathbf{x}) - f_1^L f_2^U\}, \ \forall \mathbf{x} \in C. \tag{2.16}$$

Each argument in the min function on the RHS of (2.16) is concave and the minimum of two concave functions is concave which makes it a valid concave relaxation of $f_1(\mathbf{x})f_2(\mathbf{x})$ on C.

2.2 McCormick's Convex Relaxation

McCormick [16] has proposed a method for constructing convex and concave relaxations of a function $F[f(\mathbf{x})]$ defined by the composition of a multivariate function $f(\mathbf{x})$ with a univariate function $F(z)$. The following theorem, known as McCormick's composition theorem, enables the construction of convex and concave relaxations of the composition.

Theorem 2.2.1 (McCormick's Composition Theorem). *Let $C \subset \mathbb{R}^n$ be any nonempty convex set. Consider the composite function $F \circ f$ where $f : C \longrightarrow \mathbb{R}$ is* *continuous, and let $f(C) \subset [a,b]$. Suppose that a convex function $c^u(\mathbf{x})$ and a concave function $c^o(\mathbf{x})$ satisfying*

$$c^u(\mathbf{x}) \leq f(\mathbf{x}) \leq c^o(\mathbf{x}), \ \forall \mathbf{x} \in C \tag{2.17}$$

are known. Let $e : [a,b] \longrightarrow \mathbb{R}$ be a convex relaxation of F on $[a,b]$ and let $E : [a,b] \longrightarrow \mathbb{R}$ be a concave relaxation of F on $[a,b]$. Let z_{min} be point at which e attains its infimum on $[a,b]$ and let z_{max} be point at which E attains its supremum on $[a,b]$. If the above conditions hold, then

$$u(\mathbf{x}) = e[\text{mid}\{c^u(\mathbf{x}), c^o(\mathbf{x}), z_{min}\}] \tag{2.18}$$

is a convex relaxation of $F \circ f$ on C and,

$$o(\mathbf{x}) = E[\text{mid}\{c^u(\mathbf{x}), c^o(\mathbf{x}), z_{max}\}] \tag{2.19}$$

is a concave relaxation of $F \circ f$ on C, where the mid *function selects the middle value of the three scalars.*

The theorem requires prior knowledge of a and b such that $f(C) \subset [a,b]$. This can be done by taking the natural interval extension of f on a box $\mathbb{X} \supset C$ and using $f^L$and f^U for a and b, respectively. This makes the relaxations dependent on the strength of the interval extensions and so a weak interval extension may result in weak relaxations. Hence, due care should be taken while writing the function expression to minimize the dependence problem discussed in section (1.2.3).

2.2.1 Factorable Functions

Let $C \subset \mathbb{R}^n$ be any nonempty convex set. McCormick defined a *factorable function* $f : C \longrightarrow \mathbb{R}$ as a function that can be formulated as a finite sequence of *factors* defined as binary additions, binary multiplications or univariate intrinsic functions of previously defined factors, so that

$$v_i = x_i, \ \forall i = 1, 2, 3, \ldots, n$$

and for $i = n+1, n+2, \ldots, N$

$$v_i = v_j + v_k, \;\; j,k < i$$

or

$$v_i = v_j v_k, \;\; j,k < i$$

or

$$v_i = F(v_j), \;\; j < i$$

where F is any univariate intrinsic function and f evaluates to v_N. It is similar to the class of functions for which natural interval extensions can be calculated. Subtraction and division operations can be handled by introducing univariate negative and reciprocal intrinsic functions. Most of the functions that are implementable as a computer program are factorable in the above sense. There are usually several different representations of a given function as factors, and different representations may yield different convex and concave relaxations. For example, x^2 can be treated as a univariate intrinsic function or as a binary multiplication to yield two different relaxations.

As discussed in section (2.1.1), if the interval extensions and convex and concave relaxations of two functions are known on a given interval, the corresponding relaxations for their sum, difference and product can be computed. In order to construct convex and concave relaxations of a factorable function f on C, given an interval vector (box) $\mathbb{X} \supset C$, set the first $i = 1, 2, 3, \ldots, n$ convex and concave relaxations as:

$$c_i^u = x_i$$

$$c_i^o = x_i$$

and the interval extensions $i = 1, 2, 3, \ldots, n$ as

$$V_i = X_i.$$

Assuming convex and concave relaxations are known for all univariate intrinsic functions from which f is composed, each factor $i = n+1, \ldots, N$ can be augmented with expressions defining its concave and convex relaxations, using the rules for univariate intrinsic functions, binary addition and binary multiplication. Each factor $i = n+1, \ldots, N$ can also be augmented with the expression for its natural interval extension in order to propagate the bounds needed in the expressions for the relaxations. This in fact defines a sequence of statements that can be executed by a computer program in order to evaluate simultaneously:

1. f at $\mathbf{x}$,
2. the required convex and concave relaxations at $\mathbf{x}$, and
3. the natural interval extension of f on the box $\mathbb{X}$.

Hence, the relaxations can be easily implemented as a computer program using the operator overloading features of modern programming languages. The running time required will be a small fixed multiple of the running time required to evaluate the original factorable representation of the function.

2.2.2 Nonsmoothness of McCormick's Relaxation

It is evident that, due to the frequent occurrence of min, max and mid terms in the expressions for evaluating McCormick's relaxations, these relaxations are usually nonsmooth and so convex optimization methods assuming differentiability cannot be applied to solve the resulting convex program. However, nonsmooth optimization techniques employing subgradients of the objective function can be applied. Recently, a method to compute subgradients of McCormick's convex relaxations has also been developed [21] which works in a very similar manner to the way in which automatic differentiation computes the gradient of a smooth function [5]. This enables the resulting convex program to be solved using nonsmooth optimization methods such as *bundle methods* [14, 11] and the *variable metric method* [12]. The Fortran codes [13] implementing the variable metric method (PVARS) and the proximal bundle method (PBUNS), work quite well on McCormick's nonsmooth functions and have been used in the implementation of the presented algorithm. The McCormick's convex relaxation of the objective function in (2.2) is computed to obtain the lower bounding convex program, the solution of which gives a root exclusion test as well as a reliable technique to generate a starting point for a point Newton-type iteration automatically.

2.3 Nonsmooth Root Exclusion Test

Consider the nonconvex minimization problem (2.2). By constructing McCormick's convex underestimator $u(\mathbf{x})$ of the objective function over $\mathbb{X} \subset \mathbb{R}^n$, the following convex program

$$\min_{\mathbf{x} \in \mathbb{X}} u(\mathbf{x}) \tag{2.20}$$

can be formulated and solved to lower bound the optimal value of (2.2). Since this convex program is obtained using McCormick's convex underestimators, it will be referred to as McCormick's convex program subsequently in this thesis. Before proceeding further the following result is needed.

Proposition 2.3.1. *Let* $\mathbf{f} : \mathbb{X} \subset \mathbb{R}^n \to \mathbb{R}^n$ *be any factorable function defined on an n-dimensional box* $\mathbb{X}$. *Then, the McCormick's convex relaxation* $u(\mathbf{x})$ *underestimating the vector 1-norm* $||\mathbf{f}(\mathbf{x})||_1$ *of* $\mathbf{f}(\mathbf{x})$ *is nonnegative on* $\mathbb{X}$.

Proof : Let $c_i^u(\mathbf{x})$ and $c_i^o(\mathbf{x})$ respectively denote the convex and concave relaxations of function $f_i(\mathbf{x})$ over $\mathbb{X}$ for $i = 1, 2, 3, \ldots, n$. Thus,

$$c_i^u(\mathbf{x}) \leq f_i(\mathbf{x}) \leq c_i^o(\mathbf{x}),\ \forall \mathbf{x} \in \mathbb{X},\ i = 1, 2, \ldots, n. \tag{2.21}$$

Also suppose that the numbers f_i^L and f_i^U are known (e.g. from natural interval extensions) such that

$$f_i^L \leq f_i(\mathbf{x}) \leq f_i^U,\ \forall \mathbf{x} \in \mathbb{X},\ i = 1, 2, \ldots, n. \tag{2.22}$$

To construct the convex relaxation $u_i(\mathbf{x})$ of $|f_i(\mathbf{x})|$ over $\mathbb{X}$ we observe that the univariate outer function is $F(z) = |z|$ which being convex is its own convex relaxation and attains its minimum value at $z_{i,min} = \text{mid}\{f_i^L, f_i^U, 0\}$ over $[f_i^L, f_i^U]$ for $i = 1, 2, 3, \ldots, n$. Hence, using McCormick's composition theorem the convex underestimator of $|f_i(\mathbf{x})|$ over $\mathbb{X}$ will be given by

$$u_i(\mathbf{x}) = |\text{mid}\{c_i^u(\mathbf{x}), c_i^o(\mathbf{x}), z_{i,min}\}| \geq 0, \; i = 1, 2, \ldots, n. \tag{2.23}$$

Now, the convex relaxation $u(\mathbf{x})$ for the 1-norm of $\mathbf{f}(\mathbf{x})$ can be obtained by adding the individual convex underestimators $u_i(\mathbf{x})$, as summation of convex functions preserves convexity. Hence,

$$u(\mathbf{x}) = \sum_{i=1}^{n} u_i(\mathbf{x}) \leq \sum_{i=1}^{n} |f_i(\mathbf{x})|, \; \forall \mathbf{x} \in \mathbb{X}.$$

Using (2.23), all the terms involved in the left summation above are nonnegative thereby making $u(\mathbf{x})$ nonnegative. □

The proposition above is proved only for 1-norm of $\mathbf{f}(\mathbf{x})$ but can be easily generalized to any norm. It asserts that the optimal value of the convex program (2.20) is nonnegative. Thus, if (2.20) is solved and its optimal value is found to be positive, then based on the underestimating property of the convex relaxation, it is concluded that no solution to (1.1) exists in $\mathbb{X}$. This so called *root exclusion test* provides a rigorous method to verify that no admissible solution to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ exists and proves extremely useful to debug poorly formulated process models and/or simulation problems. As will be seen in the next chapter, this test is used to fathom a large part of the search space in the proposed branch-and-bound algorithm for solving systems of nonlinear equations.

2.4 Automatic Generation of Starting Points

If the optimal value of (2.20) is found to be zero, it does not necessarily imply that an admissible solution of (1.1) exists. Nevertheless, the solution point found can be used as an automatically generated starting point for a local Newton-type iteration to find a solution of (1.1), if one exists. The following proposition motivates this choice of starting point.

Proposition 2.4.1. *Let $\mathbf{f} : \mathbb{X} \subset \mathbb{R}^n \to \mathbb{R}^n$ be any continuous function defined on an n-dimensional box $\mathbb{X}$ and let the sets S and U be defined as*

$$S = \{\mathbf{x} \in \mathbb{X} : \mathbf{f}(\mathbf{x}) = \mathbf{0}\} \tag{2.24}$$

$$U = \{\mathbf{x} \in \mathbb{X} : u(\mathbf{x}) = 0\} \tag{2.25}$$

where, $u(\mathbf{x})$ is the McCormick convex relaxation of the 1-norm $||\mathbf{f}(\mathbf{x})||_1$ of $\mathbf{f}(\mathbf{x})$ over $\mathbb{X}$. Then, $S \subset conv(S) \subset U \subset \mathbb{X}$, where $conv(S)$ denotes the convex hull of the set S.

Proof : Let $\hat{\mathbf{x}} \in \text{conv}(S)$. Using Caratheodory's theorem, there exist scalars $\lambda_j \geq 0$ and $\mathbf{x}_j \in S$ for $j = 1, 2, 3, \ldots, n+1$, such that

$$\hat{\mathbf{x}} = \sum_{j=1}^{n+1} \lambda_j \mathbf{x}_j \qquad \text{and} \qquad \sum_{j=1}^{n+1} \lambda_j = 1.$$

Again, any $\mathbf{x}^* \in S$ will also be a zero optimal solution of the optimization problem

$$\min_{\mathbf{x}\in\mathbb{X}} ||\mathbf{f}(\mathbf{x})||_1 \;=\; \min_{\mathbf{x}\in\mathbb{X}} \sum_{i=1}^{n} |f_i(\mathbf{x})|.$$

Using the nonnegativity and underestimating property of $u(\mathbf{x})$, it follows that

$$0 \leq u(\mathbf{x}^*) \leq \sum_{i=1}^{n} |f_i(\mathbf{x}^*)| = 0, \ \forall \mathbf{x}^* \in S$$

$$\Rightarrow u(\mathbf{x}^*) = 0, \ \forall \mathbf{x}^* \in S.$$

Again nonnegativity and convexity of $u(\mathbf{x})$ gives

$$0 \leq u(\hat{\mathbf{x}}) = u\Big(\sum_{j=1}^{n+1} \lambda_j \mathbf{x}_j\Big) \leq \sum_{j=1}^{n+1} \lambda_j u(\mathbf{x}_j) = 0$$

$$\Rightarrow u(\hat{\mathbf{x}}) = 0 \Rightarrow \hat{\mathbf{x}} \in U \Rightarrow \text{conv}(S) \subset U.$$

Hence, the desired set inclusion relation is obtained. □

As per the above proposition, the automatically generated starting point will lie in the set U which contains the convex hull of the desired solution set S. If the number of admissible solutions to (1.1) is small as compared to the space dimension n (a reasonable expectation for engineering problems), then conv(S) will be a smaller set relative to $\mathbb{X}$. Also, if U is not much larger than conv(S), any of the points in U is likely to be close to an admissible solution of (1.1). In fact, the difference of these two sets bear a close relation with the tightness of the convex relaxation $u(\mathbf{x})$. Continuing with the notations used in the proof of Proposition 2.3.1, let $c_i^u(\mathbf{x})$ and $c_i^o(\mathbf{x})$ be convex and concave relaxations of $f_i(\mathbf{x})$, respectively, and $u_i(\mathbf{x})$ denotes the McCormick convex relaxation of $|f_i(\mathbf{x})|$ for $i = 1, 2, 3, \ldots, n$. The convex relaxation $u(\mathbf{x})$ of $||\mathbf{f}(\mathbf{x})||_1$ will be

$$u(\mathbf{x}) = \sum_{i=1}^{n} u_i(\mathbf{x}). \tag{2.26}$$

By definition $\mathbf{x}^* \in U \Rightarrow u(\mathbf{x}^*) = 0$. Nonnegativity of $u_i(\mathbf{x})$ together with (2.26) imply that for $i = 1, 2, 3, \ldots, n$

$$u_i(\mathbf{x}^*) = 0 \Rightarrow |\text{mid}\{c_i^u(\mathbf{x}^*), c_i^o(\mathbf{x}^*), z_{i,min}\}| = 0 \tag{2.27}$$

Now, assuming that at least one admissible solution of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ exists, the following necessary condition

$$f_i^L \leq 0 \leq f_i^U, \forall i = 1, 2, 3, \ldots, n. \tag{2.28}$$

will always hold true over $\mathbb{X}$. This implies,

$$z_{i,min} = \text{mid}\{f_i^L, f_i^U, 0\} = 0, \forall i = 1, 2, 3, \ldots, n. \tag{2.29}$$

Also, by construction $c_i^u(\mathbf{x}^*) \leq c_i^o(\mathbf{x}^*)$ and in light of above assumption only one of the following holds true:

$$c_i^u(\mathbf{x}^*) \leq 0 \leq c_i^o(\mathbf{x}^*) \tag{2.30}$$

$$0 < c_i^u(\mathbf{x}^*) \leq c_i^o(\mathbf{x}^*) \tag{2.31}$$

$$c_i^u(\mathbf{x}^*) \leq c_i^o(\mathbf{x}^*) < 0 \tag{2.32}$$

The last two inequality relations (2.31) and (2.32) imply $u_i(\mathbf{x}^*) \neq 0$ leaving (2.30) which clearly asserts $u_i(\mathbf{x}^*) = 0$ as per (2.27). Hence , the set U can be alternatively described as

$$U = \{\mathbf{x}^* \in \mathbb{X} : c_i^u(\mathbf{x}^*) \leq 0, -c_i^o(\mathbf{x}^*) \leq 0, i = 1, 2, 3, \ldots, n\} \tag{2.33}$$

This characterization of U further advocates the choice of starting point to be a point lying inside it. The concave $c_i^o(\mathbf{x}^*)$ and convex $c_i^u(\mathbf{x}^*)$ relaxations of the functions $f_i(\mathbf{x})$ at any point in $\mathbf{x}^* \in U$ are opposite in sign for $i = 1, 2, 3, \ldots, n$, making it likely for $\mathbf{x}^*$ to be close to a point in the solution set S. Moreover, U being the solution set of a convex program, it is expected to be a convex set, as confirmed by the above characterization (2.33), where it is represented by convex inequality constraints involving convex functions.

Furthermore, even if no admissible solution of equation (1.1) exists, there is the possibility that in the computed natural interval extension of $\mathbf{f}(\mathbf{x})$ over $\mathbb{X}$, $f_i^L \leq 0$ and $f_i^U \geq 0$ $\forall i = 1, 2, 3, \ldots, n$. Hence, the natural interval extension will not be able to detect the nonexistence of a root. Moreover, in this case $z_{i,min} = 0$, $\forall i = 1, 2, 3, \ldots, n$. Assume that for any $1 \leq i \leq n$ either $c_i^u(\mathbf{x}) > 0$ or $c_i^o(\mathbf{x}) < 0$, $\forall \mathbf{x} \in \mathbb{X}$. Thus, $u(\mathbf{x}) > 0$, $\forall \mathbf{x} \in \mathbb{X}$ and the nonsmooth exclusion test can detect nonexistence of root.

In the proposed branch-and-bound algorithm, the nonsmooth solver PVARS [13] is used to solve the McCormick's convex program. Assuming that S is nonempty over the current box $\mathbb{X}$, the solver is supposed to find a point in set U with an optimal value of zero. It can be easily deduced that, in theory, the set U is invariant for any choice of 1, 2 or ∞ norms of $\mathbf{f}(\mathbf{x})$. However, numerical solvers rely on pre-specified tolerances and thresholds for termination and so, numerically there is a larger set $\hat{U}$ enclosing the real set U within which the nonsmooth solver is likely to converge. The convex underestimators are expected to be flat around the convex hull of the solution set S and will be further flattened on squaring thereby making the enclosing set $\hat{U}$ larger. Hence, the most obvious choice of squared Euclidean norm of $\mathbf{f}(\mathbf{x})$ as the objective function is not suitable on account of above numerical consideration. Out of the other two choices, the 1-norm is preferred over the infinity-norm to make the exclusion test more effective. In the infinity-norm only one of the functions out of $f_i(\mathbf{x}), i = 1, 2, 3, \ldots, n$ will be contributing to the optimal value of the underestimating convex program, making it a poor choice compared to the 1-norm where all the n functions will make their contribution.

Before proceeding to the next Chapter, which describes and formalizes the steps of the proposed branch-and-bound algorithm, the theoretical developments made so far in this thesis will be illustrated with the help of the following example.

2.5 Illustrative Example

Example 1. *Global minima of the Himmelblau function:*

$$f_1(x_1, x_2) = x_1^2 + x_2 - 11 = 0,$$
$$f_2(x_1, x_2) = x_1 + x_2^2 - 7 = 0,$$
$$(x_1, x_2) \in \mathbb{X} = [-6, 6]^2.$$

The Himmelblau function is defined as the squared-Euclidean norm of the above system of equations and hence is nonnegative. Thus, the global minima of the Himmelblau function are equivalent to the roots of above system of equations. In order to construct the McCormick's convex relaxation of the 1-norm of $\mathbf{f}(\mathbf{x})$ it is observed that the natural interval extensions of both f_1 and f_2 over $\mathbb{X}$ contain 0 and hence the minimum of the outer modulus function $|.|$ is attained at 0. The following intermediate

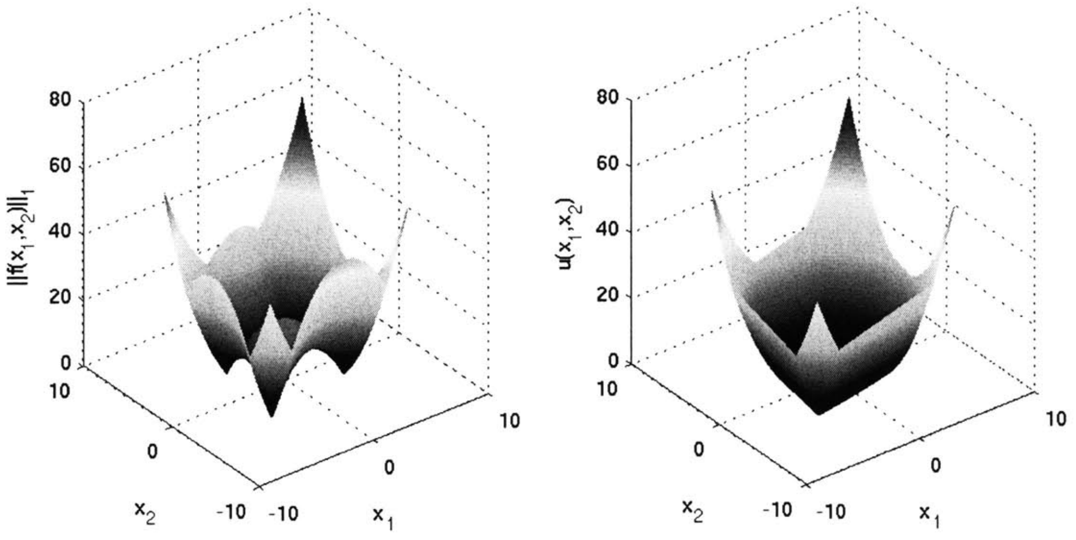

Figure 2-1: Plot of $||\mathbf{f}(\mathbf{x})||_1$ (Left) of Example 1 and its McCormick convex relaxation (Right)

.

steps can also be easily verified:

$$\begin{aligned}
c_1^u(x_1,x_2) &= x_1^2 + x_2 - 11,\\
c_1^o(x_1,x_2) &= x_2 + 25,\\
u_1(x_1,x_2) &= |\text{mid}\{x_1^2 + x_2 - 11, x_2 + 25, 0\}|,\\
c_2^u(x_1,x_2) &= x_1 + x_2^2 - 7,\\
c_2^o(x_1,x_2) &= x_1 + 29,\\
u_2(x_1,x_2) &= |\text{mid}\{x_1 + x_2^2 - 7, x_1 + 29, 0\}|.
\end{aligned}$$

Hence the McCormick's convex relaxation $u(x_1, x_2)$ of $||\mathbf{f}(x_1, x_2)||_1$ over $\mathbb{X}$ is given by

$$u(x_1,x_2) = |\text{mid}\{x_1^2 + x_2 - 11, x_2 + 25, 0\}| + |\text{mid}\{x_1 + x_2^2 - 7, x_1 + 29, 0\}|. \quad (2.34)$$

Figure 2-1 shows the plot of $||\mathbf{f}(\mathbf{x})||_1$ and its McCormick's convex relaxation $u(x_1, x_2)$. There are four points at which $||\mathbf{f}(\mathbf{x})||_1$ touches the $z = 0$ plane marking the four roots of the system of equations in Example 1. The convex relaxation constructed using

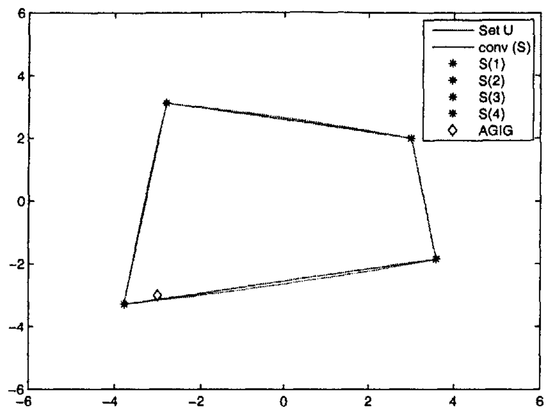

Figure 2-2: Plot of conv (S) and set U corresponding to the system of equations in Example 1

McCormick's procedure is nonnegative on $\mathbb{X}$ and is flat in the convex hull defined by the four roots. The convex hull of the solution set S for above system of equations and the solution set U of the associated McCormick convex program are shown in Figure 2-2. U contains the conv(S) with a close overlap as stated in Proposition 2.4.1. The automatically generated starting point (AGIG) $(-3, 3)$ obtained by solving the non-soomth convex program is quite close to one of the solutions $(-3.77931, -3.283185)$ and RMT based damped-Newton iterations starting from the former easily converges to the latter. All four roots (Table B.1) of this problem were found in 31 iterations of the B&B algorithm (first row in Table A.1). Another related problem, addressed in Chapter 4 is to find all stationary points of the Himmelblau function.

None of the algorithms proposed in the literature, with the exception of Wilhem and Swaney [25] whose algorithm finds a single solution, has embedded Newton's method within a branch-and-bound framework for solving systems of nonlinear equations. The proposed algorithm does this by integrating RMT based damped-Newton iterations with the convex lower bounding strategy. This is motivated by the automatic generation of a reasonably close starting point as a result of solving the convex lower bounding problem and hence often leads to fast convergence to a solution using the point Newton-type method. The efficacy of this feature is evidenced from the fact that the first solution to most of the test problems is found at the very first iteration of the algorithm. This approach is particularly helpful when only one admissible root of $\mathbf{f}(\mathbf{x})$ is required. In the next chapter the branch-and-bound algorithm for solving systems of nonlinear equations is detailed based on the theoretical developments made so far in the previous chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Branch-and-Bound Algorithm for Systems of Nonlinear Equations

This chapter describes the branch-and-bound algorithm for finding all solutions of systems of nonlinear equations (1.1) in a given box $\mathbb{X} \in \mathbb{R}^n$ (defined by the variable bounds). To exploit the interval algebra discussed in Section 1.2, the box $\mathbb{X} \in \mathbb{R}^n$ is often denoted alternatively as an interval vector of n interval variables as

$$\mathbb{X} = (X_1, X_2, X_3, \dots, X_n) \tag{3.1}$$

where, $X_i = [x_i^L, x_i^U]$ is the i^{th} interval with x_i^L and x_i^U, respectively, being the lower and upper bound on the i^{th} variable x_i for $i = 1, 2, 3, \dots, n$. The i^{th} interval-element X_i of the interval-vector $\mathbb{X}$ is also denoted by $\mathbb{X}(i)$. Likewise, the i^{th} element of a real vector $\mathbf{x}$ is denoted by $\mathbf{x}(i)$.

The intuitive idea of the branch-and-bound algorithm is to search $\mathbb{X} \subset \mathbb{R}^n$ exhaustively for solutions using a bisection strategy and fathom portions of $\mathbb{X}$ based on certain criteria.

3.1 Overview

The algorithm starts with a stack N of nodes initialized to the given box $\mathbb{X}$ (in which the solutions of (1.1) are sought), the solution set S which is initially empty and the iteration counter k set to 1. Each node $\mathbb{X}_i \in N$ will have two associated attributes, namely, the solution field $\mathbb{X}_i.\mathbf{s}$ and the solution flag $\mathbb{X}_i.f$. By default, $\mathbb{X}_i.\mathbf{s}$ will be set to a point lying inside the node $\mathbb{X}_i$ (usually the mid-point) and $\mathbb{X}_i.f$ is set to zero. If a solution has been found in the box $\mathbb{X}_i \in N$ it will be stored in its solution field $\mathbb{X}_i.\mathbf{s}$ when its solution flag $\mathbb{X}_i.f$ is set to one, indicating that a solution has been found in this node. The algorithm requires two tolerance parameters as input, namely (1) the size tolerance ϵ_{size} and (2) the feasibility tolerance ϵ_f. The size tolerance ϵ_{size} limits the smallest size of the box to be examined, which essentially means that this algorithm can distinguish solutions which are ϵ_{size} apart in $\mathbb{R}^n$ in terms of the distance metric chosen to define the box size. Also, the nonsmooth solver will terminate at a certain tolerance and hence may not locate the exact zero value of the convex program even if it exists. The feasibility tolerance ϵ_f limits the highest optimal value of the convex program below which the node will not be fathomed based on the root exclusion test.

At each iteration, a node is popped off of the stack. Since the stack is a LIFO container, the popped node will be the last one to be pushed in. If its size is smaller than ϵ_{size}, it is fathomed. Now, there are various ways of measuring the box size. The simplest one is the length of diagonal given by

$$d(\mathbb{X}) = ||\mathbf{x}^U - \mathbf{x}^L||_2. \tag{3.2}$$

Another approach suggested by Schnepper [22] is to use the scaled diameter of the box as a size metric defined as

$$d(\mathbb{X}) = \max_{1\leq i\leq n} \{(x_i^U - x_i^L)/\max(|x_i^U|, |x_i^L|, 1)\}. \tag{3.3}$$

Using this size metric the algorithm may not be able to distinguish between roots which are ϵ_{size} apart in some dimension. One may decide based on the problem which size metric to choose.

If a node is not fathomed due to small size its solution flag is checked. If a solution has been found, the Krawczyk root inclusion test is applied to check the uniqueness of the contained solution. If the inclusion test is positive the current box is fathomed. Otherwise, the box is bisected along a suitably chosen coordinate into two disjoint sub-boxes, such that the solution lies exactly in one of them (i.e., not on the boundary of division). If the solution happens to lie on the bisection boundary, the bisecting plane is shifted either upwards or downwards along the bisected coordinate by a suitable amount (usually by a small chosen fraction of the interval width along the bisection coordinate of the box) to ensure that the solution is contained in exactly one of the sub-boxes. When a box $\mathbb{X} = (X_1, X_2, X_3, \ldots, X_n)$ is bisected, the resulting sub-boxes are $\mathbb{X}_L = (X_1, X_2, \ldots, [x_q^L, \overline{x}_q], \ldots, X_n)$ and $\mathbb{X}_U = (X_1, X_2, \ldots, [\overline{x}_q, x_q^U], \ldots, X_n)$, where $X_q = [x_q^L, x_q^U]$, $\overline{x}_q$ is the mid-point of the interval X_q and q is the coordinate chosen for bisection. The bisection coordinate can be chosen in two ways. A simpler and more intuitive way is to choose the coordinate with the widest interval. Another approach is to choose the direction having largest scaled diameter [22] such that q satisfies $d(X_q) = d(\mathbb{X})$, where $d(\mathbb{X})$ is defined according to (3.3). The latter scheme performs better (especially when the initial box widths vary widely in different coordinates) and has been used in the proposed algorithm.

To facilitate the bisection scheme discussed above, the algorithm uses a subroutine *Divide*. This subroutine takes as input a parent node $\mathbb{X}$ and returns two disjoint child nodes $\mathbb{X}_L$ and $\mathbb{X}_U$ obtained by division (usually bisection) of $\mathbb{X}$ such that the point in its solution field $\mathbb{X}$.s is contained in exactly one of them. It also sets the solution field and flag of the child nodes to their default values. The subroutine *maxdim* returns the bisection coordinate of the parent interval vector $\mathbb{X}$ in q using an user defined size metric. Also, equating any two nodes $\mathbb{Y}$ and $\mathbb{X}$ using $\mathbb{Y} = \mathbb{X}$ copies information stored in all the fields of $\mathbb{X}$ to the respective fields of $\mathbb{Y}$. Using these notations and those discussed at the beginning of the chapter, a pseudo code for the subroutine *Divide* can be written as:

```
$[\mathbb{X}_L, \mathbb{X}_U] = Divide(\mathbb{X})\{$
  $\mathbb{X}_L = \mathbb{X},\ \mathbb{X}_U = \mathbb{X},\ q = maxdim(\mathbb{X})$
  $\overline{x}_q = (x_q^L + x_q^U)/2$
  $\mathbb{X}_L(q) = [x_q^L, \overline{x}_q],\ \mathbb{X}_U(q) = [\overline{x}_q, x_q^U]$
  $if\ (\mathbf{x}(q) = \overline{x}_q)$
    $\eta = 0.1(x_q^U - x_q^L)$
    $\mathbb{X}_L(q) = [x_q^L, \overline{x}_q + \eta],\ \mathbb{X}_U(q) = [\overline{x}_q + \eta, x_q^U]$
    $\mathbb{X}_U.\mathbf{s} = \mathrm{mid}(\mathbb{X}_U),\ \mathbb{X}_U.f = 0$
  $end\ if$
$\}$
```

Once bisected, the sub-box not containing the solution is pushed first, followed by the one which contains the solution and the iteration counter is increased by two to account for the newly generated two nodes. This ensures that in the next iteration the node containing the solution is again popped and this process will continue unless the solution containing node is fathomed either based on the inclusion test or the size metric. With the decreasing box size due to bisection at each iteration, the inclusion test will become more effective in the subsequent iterations and eventually the solution containing node will be fathomed based on the root inclusion test. Otherwise, even in the worst case it cannot escape the size-based fathoming, though after a much larger number of iterations.

If the current node does not contain a known solution, a simple interval-based root exclusion test is performed which is positive if the natural interval extension $\mathbb{F}$ of $\mathbf{f}$ over the current node $\mathbb{X}_i$ does not contain $\mathbf{0}$ and the node is fathomed. Otherwise, Krawczyk operator based interval root exclusion test, discussed in Section 1.2.5, is applied and if positive the current node is fathomed. If both these tests fail to fathom the current node $\mathbb{X}_i$, then McCormick convex relaxation of $||\mathbf{f}(\mathbf{x})||_1$ is constructed over

$\mathbb{X}_i$ and the lower bounding convex program is solved using any nonsmooth solver (viz. PVARS [13]) and the obtained optimal point $\mathbf{x}^*$ is stored in its solution field. If the optimal value of the convex program is positive the current node is fathomed based on the nonsmooth root exclusion test. If the optimal value of the nonsmooth convex program is zero, then starting from $\mathbf{x}^*$ RMT based damped-Newton iterations are applied. The RMT solver is set to a convergence tolerance of ϵ_{Newton} and will converge if $||\mathbf{f}(\mathbf{x})||_2 < \epsilon_{Newton}$ in a given maximum number of iterations. If a solution is found the bisection process explained in the previous paragraph for a solution containing node is performed. Otherwise, the node is bisected by calling the subroutine *Divide* with the current node such that the automatically generated starting point $\mathbf{x}^*$ lies in exactly one of the two nodes obtained after the bisection. The resulting nodes are pushed onto the stack N with the one containing $\mathbf{x}^*$ being the last node to be pushed in and the iteration counter is increased by two.

This heuristic ensures that at any iteration of the algorithm, there will be only one, if any, solution containing node which lies at the top of the stack. Also, due to the bisection of nodes at each iteration, the McCormick convex relaxations become tighter and tighter and even "closer" starting points are obtained, resulting in quick convergence of the RMT based damped Newton method. As stated earlier, except for some of the test problems, a solution is obtained at the very first iteration of the algorithm and a partial, if not full, credit to this does go to the generation of good starting points. Algorithm 3.2.1 formalizes the steps of the proposed branch-and-bound algorithm for finding all solutions of nonlinear system of equations.

3.2 Branch-and-Bound Algorithm

Based on the description of various stages of the branch-and-bound (B&B) algorithm for solving systems of nonlinear equations, the algorithm can be formalized as follows:

Algorithm 3.2.1. Branch-and-Bound Algorithm for Solving Systems of Nonlinear Equations

1. **(Initialization)**: Set $\mathbb{X}.f := 0, \mathbb{X}.\mathrm{s} = \mathrm{mid}(\mathbb{X}), N = \{\mathbb{X}\}, S = \emptyset, k = 1$.

2. **(Termination)**: If $(N = \emptyset)$ then print the solution set S. Terminate.

3. **(Node Selection)**: Pop and delete the node $\mathbb{X}_i$ from the top of stack N.

4. **(Fathoming Based on Size)**: If $(d(\mathbb{X}_i) < \epsilon_{size})$ then goto 2.

5. **(Krawczyk Root Inclusion Test)**: If $(\mathbb{X}_i.f = 1)$ then [$\mathbb{X}_i$ contains a known solution]

   - $\bar{\mathbf{x}}^* := \mathbb{X}_i.\mathrm{s}$. Compute $\mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i)$.
   - If $(\mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i) \subset \mathbb{X}_i)$ then goto 2.
   - $\mathbb{X}_i = \mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i) \cap \mathbb{X}_i$.
   - $[\mathbb{X}_k\ ,\ \mathbb{X}_{k+1}] = \mathrm{Divide}(\mathbb{X}_i)$.
   - If $(\bar{\mathbf{x}}^* \in \mathbb{X}_k)$ then
     - Push $\mathbb{X}_{k+1}$ followed by $\mathbb{X}_k$ onto the stack N.
   - Else
     - Push $\mathbb{X}_k$ followed by $\mathbb{X}_{k+1}$ onto the stack N.
   - $k = k + 2$. Goto 2.

6. **(Krawczyk Root Exclusion Test)**: Compute the natural interval extension $\mathbb{F}(\mathbf{f}, \mathbb{X}_i)$ of $\mathbf{f}$ over $\mathbb{X}_i$.

   - If $(\mathbf{0} \notin \mathbb{F}(\mathbf{f}, \mathbb{X}_i))$ then goto 2 [$\mathbb{X}_i$ does not contain a solution].
   - Else
     - $\bar{\mathbf{x}}^* := \mathbb{X}_i.\mathrm{s}$. Compute $\mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i)$.
     - If $(\mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i) \cap \mathbb{X}_i = \emptyset)$ then goto 2. [$\mathbb{X}_i$ does not contain a solution]
     - $\mathbb{X}_i = \mathbb{K}(\bar{\mathbf{x}}^*, \mathbf{f}, \mathbb{X}_i) \cap \mathbb{X}_i$.
     - Set $\mathbb{X}_i.\mathrm{s} = \mathrm{mid}(\mathbb{X}_i)$.

7. **(Automatic Starting Point Generation)**: Construct the McCormick convex relaxation $u(\mathbf{x})$ of $||\mathbf{f}(\mathbf{x})||_1$ over $\mathbb{X}_i$ and solve the resulting nonsmooth convex program using any nonsmooth solver. Let,

- $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathbb{X}_i} u(\mathbf{x})$.
- Set $\mathbb{X}_i.\mathbf{s} = \mathbf{x}^*$.

8. **(Nonsmooth Root Exclusion Test)**: If $(u(\mathbf{x}^*) > \epsilon_f)$ then goto 2 [$\mathbb{X}_i$ does not contain a solution].

9. **(RMT Based Damped-Newton Iterations)**: Apply a maximum of *maxiter* RMT iterations (NWTSLV) starting from $\mathbf{x}^*$. Let *niter* ($\leq$ *maxiter*) be the number of iterations taken by NWTSLV so that $||\mathbf{f}(\bar{\mathbf{x}}^*)||_2 \leq \epsilon_{Newton}$ where, $\bar{\mathbf{x}}^*$ stores the resulting solution.
   [$\bar{\mathbf{x}}^*$, *niter*] = NWTSLV($\mathbf{x}^*$, $\mathbf{f}$, *maxiter*, ϵ_{Newton}).
   If (*niter* $\leq$ *maxiter*) [NWTSLV Converged] then set $\mathbb{X}_i.f = 1$ and $\mathbb{X}_i.\mathbf{s} = \bar{\mathbf{x}}^*$, $S = S \cup \{\bar{\mathbf{x}}^*\}$, goto 5.

10. **(Branching)**:

   - [$\mathbb{X}_k$, $\mathbb{X}_{k+1}$] = Divide($\mathbb{X}_i$).
   - If ($\mathbf{x}^* \in \mathbb{X}_k$) then
     - First push $\mathbb{X}_{k+1}$ followed by $\mathbb{X}_k$ onto the stack N.
   - Else
     - First push $\mathbb{X}_k$ followed by $\mathbb{X}_{k+1}$ onto the stack N.
   - $k = k + 2$. Goto 2.

3.3 Implementation

This section will highlight the significant implementation details of the branch-and-bound algorithm presented in the previous section. The implementation is primarily done in C++ with extensive interfacing to Fortran subroutines already available for intermediate steps of the algorithm.

3.3.1 McCormick Convex Relaxation and its Subgradinets

C++ classes for the computation of McCormick's relaxations and AD tools for computing its subgradients need to be developed. An algorithm for subgradient propagation in McCormick's function has recently been developed at the PSEL and a C++ code for the same has been written by Benoit Chachuat [21] which is available as a shared library *libMC*. This shared library has a main class called McCormick with associated bounds, the convex and concave relaxations and their subgradients as its members which are propagated as discussed in Section 2.2.1. This shared library forms a key component of the algorithm and is used extensively for the final implementation. Given a function, an interval and a point inside the interval, its McCormick relaxation and subgradient can be computed using the above library as discussed below.

Calculation of a McCormick Relaxation

Suppose one is interested in calculating the value of the McCormick relaxation of the real-valued function $f(x, y) = x(\exp(x) - y)^2$ for $(x, y) \in [-2, 1]^2$, at the point $(x, y) = (0, 0)$. First, the variables x and y are defined. This is done as follows:

```
McCormick X( -2., 1., 0. );
McCormick Y( -2., 1., 0. );
```

Essentially, the first line means that X is a variable of class McCormick, that it belongs to the interval $[-2, 1]$, and that its current value is 0. The same holds for the McCormick variable Y. Once x and y have been defined, the McCormick's convex and concave relaxations of $f(x, y)$ at $(0, 0)$ are simply calculated as

```
McCormick Z = X*pow(exp(X)-Y,2);
```

In particular, the value of the McCormick's convex underestimator and the McCormick's concave overestimator of $f(x, y)$ on $[-2, 1]^2$ at $(0, 0)$ are obtained as

```
double Zcvx = Z.cv();
double Zccv = Z.cc();
```

Calculation of a Subgradient of a McCormick Relaxation

The calculation of a subgradient of a McCormick relaxation requires that the number of variables be specified. For the previous example, the problem has two variables (x and y), so we shall define

```
McCormick::np(2);
```

Then, the variables x and y are declared as before, except that the component index is now specified for the variables. For example, if x and y are considered to be components 0 and 1, respectively, we write

```
McCormick X( -2., 1., 0., 0 );
McCormick Y( -2., 1., 0., 1 );
```

The McCormick's convex and concave relaxations of $f(x, y)$ at $(0, 0)$, as well as a subgradient of these relaxations, are simply calculated as

```
McCormick Z = X*pow(exp(X)-Y,2);
```

Finally, a subgradient of the McCormick's convex underestimator of $f(x, y)$ on $[-2, 1]^2$ at $(0, 0)$ is obtained as

```
const double* dZcvx = Z.dcvdp();
```

Alternatively, the components of this subgradient can be obtained separately as

```
double dZcvx_X = Z.dcvdp(0);
double dZcvx_Y = Z.dcvdp(1);
```

Analogously, a subgradient of the McCormick's concave overestimator of $f(x, y)$ on $[-2, 1]^2$ at $(0, 0)$ is obtained as

```
const double* dZccv = Z.dccdp();
double dZccv_X = Z.dccdp(0);
double dZccv_Y = Z.dccdp(1);
```

Note that whenever a McCormick relaxation is differentiable at a point, then the components of the subgradient correspond to the partial derivatives of the relaxation at that point.

3.3.2 Nonsmooth Convex Solver

As already stated, the Fortran subroutine for the variable metric method PVARS written by Luskan and Vlcek [13] has been used to solve the nonsmooth convex program obtained by computing the McCormick convex relaxation of the objective function. Apart from the other arguments that PVARS requires, it also requires a Fortran subroutine called FUNDER with the following syntax:

```
SUBROUTINE FUNDER(N,XP,F,G)
```

where,

```
N     =  Space dimension
XP(N) =  Double precision vector specifying an estimate of the solution
F     =  Double precision value of the objective function at point XP
G(N)  =  Double precision vector of subgradients at the point XP
```

This is required to input the value of objective function and its subgradient at the required points to the solver PVARS.

However, unlike other nonsmooth convex functions, McCormick's convex functions do require, apart from the point of computation, the interval box over which it is computed for evaluation of its value and subgradients. Hence the objective function cannot be evaluated only by the arguments specified in FUNDER and indeed the lower and upper bounds on the variable needs to be specified. Furthermore, things become even more complicated because the McCormick classes are written in C++ while FUNDER needs to be a Fortran subroutine. To overcome this a C++ function *objmcc* is written with the following syntax:

```
objmcc(N,XP,F,G,XL,XU);
```

where, the lower and upper bounds on the variable point XP[N] is passed in the double precision array XL[N] and XU[N] respectively. The subroutine FUNDER is called from C++ code with the desired syntax which in turn calls *objmcc* with two additional variables specifying the bounds. The double precision array specifying the bounds are contained in two globally declared arrays whose values can modified only in the main program. Hence, the information contained in them can be used by FUNDER to call *objmcc* with the required additional variables.

For instance, for the problem in example 1, these functions can be written as:

```
Global double* XL = new double[N];
Global double* XU = new double[N];
extern "C" void FUNDER(N,XP,F,G){
objmcc(N,XP,F,G,XL,XU);
}
objmcc(N,XP,F,G,XL,XU)
{
 McCormick:np(N);
 McCormick* X = new McCormick[N];
 McCormick* f = new McCormick[N];
 for(i=0;i<N;i++){
 McCormick Element = X(XL[i],XU[i],XP[i],i);
 X[i] = Element;
 }
 f[0] = pow(X[0],2)+X[1]-11.0;
 f[1] = pow(X[1],2)+X[0]- 7.0;
 McCormick Z = 0.0;
 for(i=0;i<N;i++) Z = Z+abs(f[i]);
 F = Z.cv();
 G = Z.dcvdp();
}
```

3.3.3 RMT-Based Damped-Newton Solver

The RMT-based damped-Newton solver NWTSLV coded in Fortran is used in the implementation of the algorithm. A detailed discussion on the input parameters and on how to use this solver is documented in the DAEPACK manual on NWTSLV. Apart from other parameters that NWTSLV requires, it also requires a residue evaluator subroutine to compute the residuals of the original system of equations. The residual evaluator subroutine has the following syntax:

```
SUBROUTINE RESO(ICODE,N,X,F,RPARAMS,IPARAMS)
```

where,

```
ICODE    =  Integer parameter to be used by NWTSLV (both input and output)
N        =  Space dimension
X(N)     =  Real array of dimension N containing the estimate of solution
F(N)     =  Real array of dimension N containing the residual evaluated at X.
RPARAMS  =  Real parameter array
IPARAMS  =  Integer parameter array
```

Nevertheless, once this residue evaluator is provided, there are symbolic components in DAEPACK [24] which can be used to compute Jacobian matrix, its sparsity pattern and the interval extensions, automatically, making it practically suitable for the proposed implementation. For the problem in example 1, the residual evaluator is written as:

```
SUBROUTINE RESO(ICODE,N,X,F,RPARAMS,IPARAMS)
IMPLICIT NONE
INTEGER ICODE, N, IPARAMS(1)
DOUBLE PRECISION X(N), F(N), RPARAMS(1)
F(1) = X(1)**2+X(2)-11.0
F(2) = X(2)**2+X(1)- 7.0
RETURN
END
```

3.3.4 Interval Computation Tools

To take into account the involved interval analysis, C++ classes for interval computation were developed separately and the required arithmetic operators (+, −, ×, etc.) were overloaded for interval computation. DAEPACK is used to compute the Jacobian matrix and also its interval extension, required for the root inclusion test. Since the current version of DAEPACK cannot compute the interval extension of the Jacobian with its sparsity pattern being taken into account, the full Jacobian matrix is used for interval computations. As detailed in the DAEPACK manual on Automatic Code Generation it requires a specification file for computing the Jacobian of a system of equations defining the residual. A forward mode automatic differentiation is used with the post multiplier matrix set to the unity matrix. To enforce the dimension of Jacobian matrix to be $N \times N$ the argument AD_SEED_NUMBER is set equal to N in the specification file used for computing the derivative by DAEPACK. This form of storage of the output Jacobian matrix facilitates its direct manipulation from the calling C++ program for inversion and other interval arithmetic computations.

To solve a system of nonlinear equations by the proposed B&B algorithm the user needs to modify the following files as per the instructions given below:

res0.f : Code the equation describing the nonlinear system in this Fortran file as described in Section 3.3.3.

objmcc.cc : Code the same nonlinear system to compute the McCormick's convex relaxation of the objective function (1-norm of $\mathbf{f}$) and its subgradients, as detailed in Section 3.3.2.

jie_ad.spec : This is the specification file used by DAEPACK for computing the Jacobian matrix of the system of equation in *res0.f*. Ensure that the parameter AD_SEED_NUMBER in this specification file is set to N i.e., the number of equations in the system. ($N = 2$ for the system of equation in example 1).

main.cc : This is the main calling program where all initialization of the stack and other variables are done. The global variables containing the lower and upper bounds can be set and modified only from the main program to be used by all other participating functions.

A *makefile* is also written to run the program by typing *make* at the command prompt.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Computational Results

In this chapter, a number of test problems from the literature are addressed to measure the efficacy of the proposed branch-and-bound algorithm in finding all solutions of systems of nonlinear equations. The computational times reported are on a Intel Pentium 4 (3.4 GHz) processor with a size-tolerance of 10^{-4} and feasibility tolerance of 10^{-6}. For the RMT code NWTSLV parameters *maxiter* and ϵ_{Newton} were set to 20 and 10^{-8} respectively. The performance of the algorithm is judged on the basis of the performance parameters tabulated and explained in Table 4.1. The test problems are presented in the next section and the performance of the algorithm on them is shown in Table A.1 of the Appendix A.

Table 4.1: Performance parameters of the branch-and-bound algorithm.

Parameter	Description
n	Space dimension
$\|S\|$	Cardinality of the solution set S
NIT	Branch-and-bound iterations before termination of the algorithm
NITF	Branch-and-bound iterations before the first solution is found
SZ	Number of nodes fathomed based on node size
INCLT	Number of nodes fathomed by the Krawczyk root inclusion test
KEXT	Number of nodes fathomed by the Krawczyk root exclusion test
NSEXT	Number of nodes fathomed by nonsmooth root exclusion test
NWTF	Number of times RMT based damped-Newton method failed
MSD	Maximum stack depth reached prior to termination of the algorithm
CPU	CPU time taken by the algorithm in seconds

4.1 Test Problems

Example 2. *Stationary points of the Himmelblau function [15].*

$$
\begin{aligned}
4x_1^3 + 4x_1x_2 + 2x_2^2 - 42x_1 - 14 &= 0 \\
4x_2^3 + 4x_1x_2 + 2x_1^2 - 26x_2 - 22 &= 0 \\
-5.0 \leq x_1 &\leq 5.0 \\
-5.0 \leq x_2 &\leq 5.0.
\end{aligned}
$$

This system of equations results from the problem of finding the stationary points of the Himmelblau function discussed in Chapter 2. All nine solutions (Table B.2) were obtained in 113 B&B iterations with the first solution being reported at the first iteration. The iteration count is only a fraction (almost one third) of those reported in [15], taking into account the fact that here the iterations count the actual number of nodes visited in the branch-and-bound tree, unlike in [15] where it is increased by one (instead of two) at each bisection.

Example 3. *Multiple steady states of a CSTR reactor [8].*

$$
\begin{aligned}
x - \frac{1}{150}(T - T_f) &= 0 \\
x - \frac{1.34 \times 10^9 \times exp(-\frac{62800}{8.314T})}{1 + 1.34 \times 10^9 \times exp(-\frac{62800}{8.314T})} &= 0 \\
0 \leq x &\leq 1.0 \\
100 \leq T &\leq 500.
\end{aligned}
$$

This example solves the energy and mass balance equations governing the operation of a CSTR in terms of fractional conversion x and reactor effluent temperature T, with the reactor feed temperature T_f as a parameter. Three solutions were found in 43 B&B iterations as shown in Table B.3. Again the first solution was reported at the very first iteration.

Example 4. *Production of synthesis gas in an adiabatic reactor [8].*

$$\frac{x_1}{2} + x_2 + \frac{x_3}{2} - \frac{x_6}{x_7} = 0$$
$$x_3 + x_4 + 2x_5 - \frac{2}{x_7} = 0$$
$$x_1 + x_2 + x_5 - \frac{1}{7} = 0$$
$$x_1 + x_2 + x_3 + x_4 + x_5 - 1 = 0$$
$$400x_1x_4^3 - 178370x_3x_5 = 0$$
$$x_1x_3 - 2.6058x_2x_4 = 0$$
$$-28837x_1 - 139009x_2 - 78213x_3 + 18927x_4 + 8427x_5 + \frac{13492}{x_7} - 10690\frac{x_6}{x_7} = 0$$
$$0 \le x_i \le 1.0 \;, i = 1,2,3,4,5.$$
$$0 \le x_6 \le 5.0$$
$$0 \le x_7 \le 5.0.$$

The above system of equations represent three atom balances, a mole fraction constraint, two equilibrium relations, and an energy balance equations. The only solution to this problem, as shown in Table B.4, was found at the first iteration. However, it took 57 iterations to fathom the whole search space.

Example 5. *Badly scaled system of equations [15].*

$$10000x_1x_2 - 1 = 0$$
$$exp(-x_1) + exp(-x_2) - 1.001 = 0$$
$$5.490 \times 10^{-6} \le x_1 \le 4.553$$
$$2.196 \times 10^{-3} \le x_2 \le 18.210.$$

Only one solution (Table B.5) to this problem was found in 17 iterations which is nearly a quarter of the number of iterations reported in [15]. Once again the solution was found in the first iteration of the algorithm and the remaining 16 iterations were taken to fathom the whole search space based on the various root exclusion/inclusion tests.

Example 6. *Robot kinematic problem [15].*

$$
\begin{aligned}
0.004731x_1x_3 - 0.3578x_2x_3 - 0.1238x_1 + x_7 - 0.001637x_2 - 0.9338x_4 - 0.3571 = 0 \\
0.2238x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - x_7 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0 \\
x_6x_8 + 0.3578x_1 + 4.731 \times 10^{-3}x_2 = 0 \\
-0.7623x_1 + 0.2238x_2 + 0.3461 = 0 \\
x_1^2 + x_2^2 - 1 = 0 \\
x_3^2 + x_4^2 - 1 = 0 \\
x_5^2 + x_6^2 - 1 = 0 \\
x_7^2 + x_8^2 - 1 = 0 \\
-1.0 \le x_i \le 1.0, \;\; i = 1, 2, \ldots, 8.
\end{aligned}
$$

All 16 solutions to this problem were found in 2235 iterations which is almost half of the number of iterations reported in [15] to solve this problem. Again the first solution was found in one iteration of the algorithm and the total iteration count is less than that reported in [15]. The obtained solution is shown in Table B.6.

Example 7. *Brown's almost linear system [7].*

$$
\begin{aligned}
2x_1 + x_2 + x_3 + x_4 + x_5 - 6 = 0 \\
x_1 + 2x_2 + x_3 + x_4 + x_5 - 6 = 0 \\
x_1 + x_2 + 2x_3 + x_4 + x_5 - 6 = 0 \\
x_1 + x_2 + x_3 + 2x_4 + x_5 - 6 = 0 \\
x_1x_2x_3x_4x_5 - 1 = 0 \\
-2.0 \le x_i \le 2.0, \;\; i = 1, 2, 3, 4, 5.
\end{aligned}
$$

Two solutions (Table B.7) to this problem were found in 523 iterations with one of them being obtained at the first iteration. For this problem, the minimum value of α choosen for the convex lower bounding the last nonconvex expression using the αBB procedure [15] should be 16 as dictated by the range of eigenvalues of the corresponding Hessian matrix. Hence, iteration counts reported for α less than 16 in [15] are not matched by the proposed algorithm. Nevertheless, the results are comparable with those in [15] for appropriate values of α .

Example 8. *Problem 4 from [9].*

$$x \log(x) + 0.36787 = 0$$
$$0.2 \leq x \leq 0.5.$$

As shown in Table B.8, the two admissible solutions obtained after 37 B&B iterations are quite close to each other. Had the roots not been so close, even lesser number of iterations is expected because, in this case, the root inclusion test cannot successfully fathom the solution containing nodes unless it is smaller than 0.01 (approx. separation between the roots) units in size. This requires more bisections thereby increasing the iteration count.

Example 9. *Problem 5 from [9].*

$$\frac{481.6282 - x(533.2807 - x(166.197 - x(21.1115 - x(1.1679 - 0.023357))))}{\exp(x)} = 0$$
$$0 \leq x \leq 20.$$

Five real solutions (Table B.9) are found in 137 iterations which is only a fraction of the number of iterations reported in [9] to solve this problem using interval-Newton/generalized bisection method with natural interval extensions. Writing the expression in nested form (*Horner's Scheme*), instead of expanded power form, improves the quality of interval computations and hence the effectiveness of the exclusion and inclusion tests.

Example 10. *Chemical equilibrium under a stoichiometric feed condition [6].*

$$-1.674 + x(2.511 + x(14.7445 + x(x - 7.79075))) = 0$$
$$0.1 \leq x \leq 0.9.$$

This equation represents the equilibrium conversion of nitrogen and hydrogen to ammonia starting with 3 moles of hydrogen and 1 mole of nitrogen at the reaction conditions of 250 atm and 500^{0}C. The variable x is the number of moles of nitrogen reacted at equilibrium, which, for the basis of 1 mole of nitrogen also represents, the fractional conversion of nitrogen. Only one real solution 0.27759 is found in 7 iterations.

Example 11. *Chemical equilibrium under a non-stoichiometric feed condition [6].*

$$0.00850239 + x(-0.1816915 + x(0.699096 + x(x - 1.3))) = 0$$
$$0.0 \leq x \leq 0.5.$$

Similar to the preceding example, the above equation results from a chemical equilibrium problem involving synthesis of ammonia from nitrogen and hydrogen, this time though, under a non-stoichiometric feed condition. The feed gas contains 0.5 moles of nitrogen, 0.8 moles of hydrogen and 0.3 moles of ammonia. The temperature and pressure is maintained at 298.15 K and 0.00243 atm respectively, such that the chemical equilibrium constant is 604500 atm^{-2}. The variable x represents the number of moles of nitrogen reacted at equilibrium which can never be more than 0.5 as stated by the associated variable bound. Only one real solution 0.0586546 is found in 7 iterations.

Example 12. *Kinetics in a stirred reactor [6].*

$$2T\log(T) + 25.432796T - 21000 = 0$$
$$500 \leq T \leq 600.$$

This equation results after applying a logarithmic transformation to a badly scaled equation governing the operation of a CSTR. T is the reactor temperature measured in the absolute scale. The only feasible root 551.7738K is found in just 3 iterations.

Example 13. *Adiabatic flame temperature [6].*

$$\Delta H + \alpha(T-298) + \frac{\beta}{2}(T^2 - 298^2) + \frac{\gamma}{3}(T^3 - 298^3) = 0$$
$$\Delta H = -57798, \alpha = 7.256, \beta = 2.298 \times 10^{-3}, \gamma = 0.283 \times 10^{-6}$$
$$3000 \leq T \leq 5000.$$

This cubic equation in temperature T expressing the adiabatic energy balance, computes the temperature of an adiabatic flame. 4305.31K is the only feasible root found in 3 iterations of the B&B algorithm.

Example 14. *Gas volume using Beattie-Bridgeman equation [6].*

$$1.251 \times 10^{-6} + x(-5.422539 \times 10{-}4 + x(0.011658361 + x(x - 0.22411958))) = 0$$
$$0 \leq x \leq 0.9.$$

This fourth order polynomial equation is obtained by the application of Beattie-Bridgeman equation of state to study the molar volume of gaseous methane at different temperature and pressure. Two feasible solutions 0.00242793 and 0.17496723 are obtained in 23 iterations of the B&B algorithm.

Example 15. *Fractional conversion in a reactor [6].*

$$x_2(1.0 - x_1) - (1.0 - 1.25x_1) = 0$$
$$x_2 - \exp(0.8x_2 - 1.691954) = 0$$
$$0 \leq x_1 \leq 0.9$$
$$0 \leq x_2 \leq 0.9.$$

This system of equations results while trying to determine the fractional conversion of the limiting reactant in a reactor. At the very first iteration of the algorithm, the only feasible solution (0.7573962471, 0.2195130556) is found. It took two more iterations to fathom the whole search space making the total iteration count 3.

Example 16. *Flow in a smooth pipe (A) [6].*

$$ax_1^2 + bx_2 - c = 0$$
$$x_1^7 - x_2^4 = 0$$
$$a = 240, b = 40, c = 200.$$
$$0 \le x_1 \le 1.0$$
$$0 \le x_2 \le 1.0.$$

The above system of equation represents the mechanical energy balance for a fluid flowing in a smooth pipe and finds the average velocity x_1. The second equation is just to eliminate the fractional power term $x_1^{7/4}$, representing the skin friction, from the original equation by introducing an extra variable x_2. The x_1^2 term in the first equation reflects the changes in kinetic energy of the fluid. The only feasible solution found in 13 iterations of the B&B algorithm is (0.8425243284, 0.7409165365).

Example 17. *Flow in a smooth pipe (B) [6].*

This solves the previous problem with a different parameter value of a, b and c. The result is tabulated in Table 4.1.

Table 4.2: Parameter value and solution for Example 17.

a	b	c	x_1	x_2	NIT
46.64	67.97	40	0.5656503851	0.368942896	21

Example 18. *Batch distillation at infinite reflux [6].*

$$x - \left(\frac{1 + 0.05x}{0.95}\right)^{64} (0.90)^{63} = 0$$
$$0 \le x \le 18.$$

The above equation is derived from an equation governing the batch distillation of a binary mixture using total reflux. The original equation is transformed using a change of variables. Two feasible solutions 1.111111481 and 0.03962623025 are found in 17 iterations.

Example 19. *Volume from the virial equation of state [6].*

$$-4242149.1 + V(74944.6341 + V(V - 471.34)) = 0$$
$$100 \leq V \leq 500.$$

This cubic equation results from the calculation of the specific volume V of a gas using the virial equation of state. The only feasible solutions is 212.9580159 which is reported at the very first iteration of the B&B algorithm. The algorithm terminated after 21 iterations and the remaining 20 iterations were taken to fathom the search space.

Example 20. *Volume from the Redlich-Kwong equation of state [6].*

$$-0.00064834321 + V(0.015878415 + V(V - 0.172326)) = 0$$
$$0 \leq V \leq 1.0.$$

Like the virial equation of state, the Redlich-Kwong equation of state also relates the state variables namely pressure (P), volume (V) and temperature (T). The above cubic equation results from the calculation of the specific volume V of a gas using the Redlich-Kwong equation of state. Though, the only feasible solutions 0.075709 is reported at the very first iteration of the B&B algorithm, termination occurred only after the iteration count reached 15.

Example 21. *Sinkage depth of a sphere in water [6].*

$$x^3 - 3x^2 + 2.4 = 0$$
$$0 \leq x \leq 5.0.$$

This third order polynomial equation is obtained as a result of finding the sinkage depth x of a sphere in water. Two feasible solutions 1.1341378 and 2.6610819 are obtained in 13 iterations. By decreasing the lower bound on the variable x the third solution -0.7952197 reported in [6] is also obtained.

Example 22. *Rosenbrock functions [18].*

$$\begin{aligned} 10(x_2 - x_1^2) &= 0 \\ 1 - x_1 &= 0 \\ (x_1, x_2) &\in [-2, 2]^2. \end{aligned}$$

This function has only one zero at $(1, 1)$. The solution is located in the first iteration and another 4 B&B iterations were required to fathom the entire search space. Due to the simple form of the functions, natural interval extensions were able to compute the exact range of the functions, which made the interval extension based exclusion test very effective. As a result the nonsmooth exclusion test was not performed at all.

Example 23. *Freudenstein and Roth function [18].*

$$\begin{aligned} -13 + x_1 + ((5 - x_2)x_2 - 2)x_2 &= 0 \\ -29 + x_1 + ((x_2 + 1)x_2 - 14)x_2 &= 0 \\ (x_1, x_2) &\in [0, 6]^2. \end{aligned}$$

The algorithm terminated in 39 iterations while the only feasible solution $(5, 4)$ is found in 17 iterations.

Example 24. *Beale function [18].*

$$\begin{aligned} 1.5 - x_1(1 - x_2) &= 0 \\ 2.25 - x_1(1 - x_2^2) &= 0 \\ 2.625 - x_1(1 - x_2^3) &= 0 \\ (x_1, x_2) &\in [0, 3]^2. \end{aligned}$$

This system of equations has three equations in two unknowns. So three distinct pairs of equations can be formed and solved. The three test cases A, B and C are studied using the pairs formed by first and second, second and third and third and first equations respectively. In all the cases only one feasible solution $(3, 0.5)$ is found.

Example 25. *Powell's singular function [18].*

$$\begin{aligned} x_1 + 10x_2 &= 0 \\ \sqrt{5}(x_3 - x_4) &= 0 \\ (x_2 - 2x_3)^2 &= 0 \\ \sqrt{10}(x_1 - x_4)^2 &= 0 \\ (x_1, x_2, x_3, x_4) &\in [0, 1]^4. \end{aligned}$$

The only root lies at the origin where the Jacobian is singular. This weakens the inclusion test and hence the node which contains the solution cannot be fathomed unless its size becomes less than the minimum box-size limit. This explains the comparatively larger number of B&B iterations taken to solve this problem.

Example 26. *Wood's function [18].*

$$\begin{aligned} 10(x_2 - x_1^2) &= 0 \\ \sqrt{90}(x_4 - x_3^2) &= 0 \\ \sqrt{10}(x_2 + x_4 - 2) &= 0 \\ \frac{1}{\sqrt{10}}(x_2 - x_4) &= 0 \\ (x_1, x_2, x_3, x_4) &\in [0, 2]^4. \end{aligned}$$

The original Wood's function is an over defined system of six equations in four variables. Four of them are linear and two are nonlinear equations. Two out of the four linear equations in the system are removed to form the above system of four equations in four unknowns. The B&B algorithm terminated in 13 iterations and successfully found the only feasible solution $(1, 1, 1, 1)$.

Example 27. *Broyden tridiagonal function [18].*

$$\begin{aligned} (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1 &= 0 \\ x_0 = x_{n+1} &= 0 \\ x_i \in [-2, 2],\ i = 1, 2, 3, \ldots, n&. \end{aligned}$$

This system of equations is solved for $n = 2, 4$ and 6 (designated by cases A, B and C respectively) and in each of these cases two feasible solutions are found. The iteration count for termination of the algorithm increases with the increase in the dimensionality n. Solutions obtained for each of the three cases are shown in Table B.10, Table B.11 and Table B.12, respectively.

Example 28. *Broyden banded function [18].*

$$
\begin{aligned}
x_i(2+5x_i^2)+1-\sum_{j\in J_i} x_j(1+x_j) &= 0 \\
J_i = \{j : j \neq i, \max(1, i-m_l) \leq j \leq \min(n, i+m_u)\} \\
m_l = 5, m_u = 1 \\
x_i \in [-1,1],\ i = 1,2,3,\ldots,n.
\end{aligned}
$$

The B&B algorithm is applied on the above system of equations with $n = 2, 5$ and 9 (designated by cases A, B and C respectively). One feasible solution is found in all the three cases and are shown in Table B.13, Table B.14 and Table B.15, respectively.

Example 29. *Extended Rosenbrock function [18].*

$$
\begin{aligned}
10(x_{2i}-x_{2i-1}^2) &= 0 \\
1-x_{2i-1} &= 0 \\
i &= 1,2,3,\ldots,n/2 \\
x_i \in [-2,2],\ i &= 1,2,3,\ldots,n.
\end{aligned}
$$

The above system of equation generalizes the system of two equations shown in Example 22 for any even n. This is solved for $n = 10, 50$ and 100 using the B&B algorithm. In all the cases the algorithm converged in 5 iterations and only one feasible solution is found where all the variables are unity.

Example 30. *Circuit design problem with extraordinary sensitivities to small pertur-*

bations [15].

$$(1 - x_1x_2)x_3\big[\exp\big\{x_5(g_{1k} - g_{3k}x_7 10^{-3} - g_{5k}x_8 10^{-3})\big\} - 1\big]$$
$$-g_{5k} + g_{4k}x_2 = 0,$$
$$(1 - x_1x_2)x_4\big[\exp\big\{x_6(g_{1k} - g_{2k} - g_{3k}x_7 10^{-3} + g_{4k}x_9 10^{-3})\big\} - 1\big]$$
$$-g_{5k}x_1 + g_{4k} = 0,$$
$$k = 1, 2, 3, 4.$$
$$x_1x_3 - x_2x_4 = 0$$
$$x_i \in [0, 10],\ i = 1, 2, 3, \ldots, 9.$$

where,

$k =$	1	2	3	4
g_{1k}	0.4850	0.7520	0.8690	0.9820
g_{2k}	0.3690	1.2540	0.7030	1.4550
g_{3k}	5.2095	10.0677	22.9274	20.2153
g_{4k}	23.3037	101.7790	111.4610	191.2670
g_{5k}	28.5123	111.8467	134.3884	211.4823

This set of equation results while trying to solve a circuit design problem with extraordinary sensitivities to small perturbations. With *maxiter* of NWTSLV set to 20 no solutions were reported to be found for this problem. However, on increasing the *maxiter* to 100 resulted in convergence of NWTSLV to the desired solution (Table B.16) at the ninth B&B iteration. The value of other output parameters for this problem, shown in Table A.1 is for *maxiter* set to 100.

Although, with this change, the B&B algorithm was able to find the solution, the nonsmooth exclusion test does not seem to work well on this example and the search space was not fathomed fully even after 20000 B&B iterations. McCormick convex relaxation happens to be flat over a sizable part of the domain and even when the box size becomes small, the relaxation does not becomes positive over the box. Because of the exponential terms and multiple occurrence of the same variable the natural interval extensions are very loose resulting in the flat relaxation using McCormick's procedure.

In [15] the problem is solved using αBB convex relaxation with α value as low as 0.1. It can be easily deduced by computing the natural interval extension of the Hessian of the last equation (which is the least nonconvex among the equations in the above system) that the value of α should at least be 0.5. Furthermore, rigorous αBB yields an α value of the order of 10^7. So, the iteration counts reported in [15] will be much more higher when an appropriate value for α is used.

Example 31. *Hydrocarbon combustion process [15].*

$$\begin{aligned}
x_1x_2 + x_1 - 3x_5 &= 0\\
2x_1x_2 + x_1 + 3R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4 + R_8x_2 - Rx_5 &= 0\\
2x_2x_3^2 + R_7x_2x_3 + 2R_5x_3^2 + R_6x_3 - 8x_5 &= 0\\
R_9x_2x_4 + 2x_4^2 - 4x_5 &= 0\\
x_1x_2 + x_1 + R_{10}x_2^2 + x_2x_3^2 + R_7x_2x_3 + R_9x_2x_4 + R_8x_2 + R_5x_3^2&\\
+R_6x_3 + x_4^2 - 1 &= 0\\
x_i \in [0.0001, 100],\ i &= 1, 2, 3, 4, 5.
\end{aligned}$$

where,

R	R_5	R_6	R_7	R_8	R_9	R_{10}
10	0.1930	4.106×10^{-4}	5.451×10^{-4}	4.497×10^{-7}	3.407×10^{-5}	9.615×10^{-7}

This example addresses the equilibrium of the products of a hydrocarbon combustion process with the problem being formulated in the element variable space. The B&B algorithm found the only feasible solution (Table B.17) at the first iteration. However, like the previous example the McCormick's convex relaxation happens to be flat over a big portion of the search space. This happens because of loose natural interval extension due to the frequent occurrence of the same variable in the equations which enhances the dependence problem of natural interval extension. As discussed in

Chapter 1, the dependence problem of natural interval extensions lead to overly large bound estimation of the function range even for smaller boxes leading to flattening of the convex relaxation. Even expressing the equations in Horner's form does not achieve any significant improvement in the natural interval extension. Hence, the search space could not be fathomed fully even in 20000 B&B iterations.

4.2 Performance Analysis

Table A.1 shows the performance of the proposed branch-and-bound (B&B) algorithm on all the test problems discussed in the preceding section. Some of the test problems (e.g., 27, 28, 29, etc.) can be decomposed into smaller blocks and can be very easily solved block-by-block. However, for testing the performance of the algorithm, they were not block decomposed at any stage to preserve the difficulty level.

As evident from Table A.1 the first solution is found at the very first iteration of the algorithm for most of the test problems. This is due to the combined effect of the good starting point generation and the increased region of convergence of the damped-Newton method through the use of RMT with natural level function. Also, the efficacy of the proposed nonsmooth exclusion test is evidenced by the fact that it successfully fathoms nodes which are left unfathomed by Krawczyk root exclusion test, since the former is performed only if the latter fails. The Krawczyk operator based root inclusion test also has an important contribution in reducing the number of B&B iterations, in lack of which, the only way to fathom the solution containing node is based on the size metric, leading to a significant increase in the number of iterations as well as processing time.

In the proposed algorithm two root exclusion tests and one root inclusion test are operating simultaneously to extract the best out of them in cutting down the iteration count and hence the CPU time. However, to illustrate their individual effects, all the test problems were solved by switching off each of these two exclusion tests and the inclusion test. B&B iterations (NIT) and the CPU times taken for all the three cases are presented in Table A.2. To force convergence in finite time, an upper limit of 20000 on the maximum number of B&B iterations is imposed. NIT values for problems on which the algorithm terminated without converging are mentioned as NC (Not Converged) in the results table.

4.2.1 B&B Without Krawczyk Root Exclusion Test

When the Krawczyk root exclusion test is switched off, the iterations count remains almost the same for most of the problems except for Example 25 where it is nearly doubled. This means that the nodes which were fathomed by the Krawczyk exclusion test are also successfully fathomed by the proposed nonsmooth exclusion test. For some of the problems, there is a small increase in the iterations because the nonsmooth exclusion test was not positive for some of the nodes which were earlier filtered by the Krawczyk exclusion test. Hence, subsequent bisections are required to have a tight enough convex relaxation before successful fathoming by the nonsmooth exclusion test. Nevertheless, a small increase in the iterations indicates that such nodes are small in number and usually convex relaxation on the nodes were tight enough not to require too many subsequent bisections to be fathomed by the proposed exclusion test.

Barring a few exceptions, the CPU times taken by the algorithm increases on switching off this exclusion test and the increase is quite significant for some of the problems (e.g., 6, 13, 14, 23, 24, 25 etc.). In some cases (e.g., 5, 8, 9, 12, 17 and 19) the CPU time decreased also but by a small amount. If the nonsmooth solver takes a larger number of iterations to solve the lower bounding convex program it will overwhelm the time saved in avoiding the computation of Krawczyk operator. As a result, the overall processing time will increase and this increment will further depend on the number of nodes on which the convex program is solved. In most of the presented test cases, as the node count (B&B iterations) does not increase much on switching off the Krawczyk exclusion test, it can be inferred that the additional number of nonsmooth convex program solved were nearly the same as the number of nodes fathomed by the Krawczyk exclusion test when it was turned on. Hence, for most practical cases, the proposed nonsmooth exclusion test is nearly as effective as the Krawczyk exclusion test and is computationally preferable as well.

4.2.2 B&B Without Nonsmooth Root Exclusion Test

Turning off the nonsmooth root exclusion test leads to a dramatic increase in the number of B&B iterations as well as the CPU time (except for the examples 30 and 31 for which the algorithm did not converge in 20000 iterations). If NSEXT denotes the number of nodes fathomed by the nonsmooth exclusion test when all features of the algorithm were turned on, then on turning it off, the increase in the B&B iteration (NIT) will at least be 2×NSEXT. However, the observed increment in NIT is much more than this lower limit for most of the test problems. This demonstrates the prowess of the proposed nonsmooth root exclusion test as compared to the Krawczyk exclusion test. The increase in NIT is specially significant in problems of higher dimension and/or those with higher cardinality of the solution set S (e.g., 4, 5, 6, 7, 9, 27 and 28). In fact, without the nonsmooth exclusion test, for the robot kinematic problem having as many as 16 solutions in a small volume of $[-1, 1]^8$, the whole search space could not be fathomed and the algorithm terminated when the maximum limit of 20000 on NIT is achieved.

Consider those NSEXT nodes that were fathomed by the nonsmooth exclusion test. With the latter being turned off, all these nodes will now filter down to that stage of the B&B algorithm where RMT iterations are started, escaping the interval-based Krawczyk exclusion tests. As no solution is contained in those nodes the RMT iterations will eventually fail to find a solution and will end up only in increasing the processing time. Furthermore, since the convex lower bounding problem is no longer solved, the starting points are also not generated automatically for the RMT iterations. Hence, in this case, the RMT iterations are started from the mid-point of the box which also leads to more failures of NWTSLV for some of the test problems. These increased failure of the RMT iteration (NWTSLV), leads to a dramatic increase in the CPU time for most of the test cases. However, for some problems (e.g., 12, 13, 22 and 29) there is no change in the iteration count because the nonsmooth exclusion test was not performed on any of the nodes and so switching it off does not effect the performance parameters.

For examples 30 and 31 the fall in CPU time has nothing to do with the efficacy of the nonsmooth exclusion test. Because of the upper ceiling on the number of iterations (NIT = 20000), nodes generated by the bisection of the nodes which were earlier fathomed by the nonsmooth exclusion test, are not processed fully by the algorithm anymore. For these problems, the volume of the original search space being processed by the algorithm with the nonsmooth exclusion test turned on is considerably higher than what is being processed when it is turned off. This is reflected in a dramatic decrease in the CPU time in the latter case. Had the algorithm been allowed to run till whole of the domain being fathomed, there would have been a tremendous increase in the number of iterations leading to a proportionate increase in the CPU time as in all the other cases.

With the nonsmooth exclusion test switched off, the remaining steps of the B&B algorithm closely approximates the Krawczyk-interval/bisection method for solving nonlinear equations but with the difference that RMT iterations are also applied on the boxes. Thus, even if it takes too many iterations to fathom the search space, solutions, if any, were found robustly and rapidly. As emphasized earlier, an added advantage of having the solution known a priori is excellent preconditioning of the Krawczyk's operator which makes the root inclusion test quite effective. Moreover, the intersection relation in the Krawczyk exclusion and inclusion tests leads to better refinement of the regions where the solutions are likely to lie.

4.2.3 B&B Without Krawczyk Root Inclusion Test

When the root inclusion test is switched off, there is a significant increase in the B&B iterations. However, the increment is fixed depending upon the dimensionality n of the problem, the cardinality of its solution set S and last, but not the least, on the initial box-size. If a solution is found in a box, then with the inclusion test being switched off, the box can only be fathomed when its size becomes smaller than the threshold box-size limit (ϵ_{size}) by subsequent bisection. This leads to the generation of an exponential number of nodes and so the iteration count will increase dramatically.

The increase in iteration count will also depend on how large a box the inclusion test fathomed when it was switched on. For example, in problems 24, 25 and 26, the nodes in which a solution was found was fathomed based on the size metric and not by the inclusion test even when the test was turned on. Hence, one may expect no change in the number of iterations when the inclusion test is turned off. But, the test also helps in reducing the box size by intersecting it with the Krawczyk operator as discussed in Chapter 3 which explains the increment in NIT for these problems.

The CPU time shows an increase for all the test problems. The nodes generated by bisection of the solution containing nodes are not easily fathomed by the exclusion tests, especially when the solution lies near the boundary of bisection. Hence, such nodes will experience the relatively expensive computational steps of the algorithms such as calculation of the Krawczyk operator, solution of nonsmooth convex program and even the RMT iterations (in case both these tests fail to fathom the node). Hence, the increase in the processing time will depend on the number of nodes generated and the amount of computationally expensive steps that each such node undergoes. The increase in B&B iterations as well as the CPU time is particularly significant in the solution of the *extended Rosenbrock function* (Examples 29 A, B and C) where the problems are relatively bigger in size and the inclusion test, when on, was able to fathom a large volume of the search space.

Having seen the effect of switching off various inclusion and exclusion tests from the proposed B&B algorithm, it seems that the combined effect of all of them working together is considerably better than any of them used in isolation, both in terms of iteration counts and processing time. Furthermore, the obtained test results also illustrate that the proposed nonsmooth exclusion test outperforms the Krawczyk operator based exclusion test when used in isolation. However, their combined effect yields the best performance and is better than either of the two acting independently.

Chapter 5

Conclusion and Future Work

McCormick's convex relaxation seems very promising for convex lower bounding of nonconvex programs because of its capability to produce relatively tighter relaxations. This is particularly helpful in the solution of systems of nonlinear equations using a global optimization approach as investigated in this thesis. In this formulation, due to the special structure of the objective function, nonnegativity of the McCormick convex relaxation can be established leading to a root exclusion test. This nonsmooth root exclusion test has a significant edge in performance over the interval-based Krawczyk root exclusion test as demonstrated by a number of test problems discussed in Chapter 4. Another important contribution is the set inclusion relation asserted by proposition 2.4.1 providing a technique for automatic generation of starting points for point Newton-type iterations.

A distinguishing feature of the proposed algorithm is that by embedding the RMT based Newton-type method in a branch-and-bound framework, a solution can be located at the very first iteration for most of the test problems. This is further exploited to fathom the search space more efficiently using the Krawczyk's root inclusion test by checking the uniqueness of the known solution in the given box.

Rigorous global convergence is guaranteed by the design of the algorithm itself. The bisection strategy combined with various inclusion and exclusion tests and the allowed minimum box-size limit ensures that the algorithm will convergence finitely. However, the key question is the high computational effort required for large problems where special concerns arise for solving the nonsmooth convex program and the amount of branching and partitioning required. The efficiency of the algorithm also depends on the range specification of the variables. Overly large bounds on variables not only lead to weaker interval extensions and bad relaxations, but also make the interval based inclusion and exclusion tests ineffective until sufficient contraction on variable bounds is achieved by successive bisection.

Natural interval extensions are used for estimating the function ranges required for constructing the convex relaxations using McCormick's method. As seen in some of the test examples, the dependency problem associated with natural interval extensions leads to overly large interval bounds on the function ranges. This causes the McCormick's convex relaxation to be almost flat over a large part of the domain of its definition and weakens the nonsmooth exclusion test. Hence, one is motivated to use the Taylor-model interval extensions [9] instead of the natural interval extensions for better estimation of the function ranges leading to comparatively tighter convex relaxations. However, unlike with natural interval extensions, bounds propagation using the Taylor-model interval extension for computing McCormick's relaxation will require the knowledge of the function derivatives and hence will be computationally more expensive. However, the overall performance may be faster if it can cut down the iteration counts significantly.

Another important scope for future work is motivated by the convexity of set U. As established earlier that, for zero optimal value of the lower bounding convex program, the solution set U is a convex set which contains all solutions of (1.1). Hence it would be desirable to prevent the point Newton-type iterations from leaving the set U. This can be done utilizing the convexity of U and using the separating hyperplane theorem. If the Newton-type iterate generates a point outside U then a separating hyperplane which separates this point from U can be constructed by solving another convex program and it can be imposed as a constraint or "cutting plane" to further restrict the admissible domain. This will potentially cut down the iteration counts for Newton-type methods significantly though at the expense of the involved computational efforts.

Appendix A

Performance Tables for the Branch-and-Bound Algorithm

Table A.1: Performance of the B&B algorithm on the test problems.

Ex.	n	$\|S\|$	NIT	NITF	SZ	INCLT	KEXT	NSEXT	NWTF	MSD	CPU(S)
1	2	4	31	1	0	4	10	2	0	6	0.01503
2	2	9	113	1	0	9	31	17	10	9	0.08087
3	2	3	43	1	0	3	7	12	4	9	0.03568
4	7	1	57	1	0	1	9	19	2	27	0.38239
5	2	1	17	1	0	1	5	3	2	7	0.03384
6	8	16	2235	1	32	0	413	723	51	49	3.93381
7	5	2	523	1	2	1	50	209	214	32	1.16246
8	1	2	37	1	2	2	3	12	4	11	0.02197
9	1	5	137	1	0	5	11	53	33	10	0.10800
10	1	1	7	1	0	1	3	0	0	4	0.00246
11	1	1	7	1	0	1	0	3	1	3	0.00415
12	1	1	3	1	0	1	1	0	0	2	0.00200
13	1	1	3	1	0	1	1	0	0	2	0.00187
14	1	2	23	1	0	2	8	2	1	8	0.00839
15	2	1	3	1	0	1	1	0	0	2	0.00191
16	2	1	13	1	0	1	6	1	2	6	0.00856
17	2	1	21	1	0	1	7	3	1	10	0.01457
18	1	2	17	1	0	2	6	1	0	8	0.00446
19	1	1	21	1	0	1	2	8	3	8	0.01302
20	1	1	15	1	0	1	3	4	0	8	0.00402
21	1	2	13	1	0	2	4	1	0	5	0.00448
22	2	1	5	1	0	1	2	0	0	3	0.00321
23	2	1	39	17	0	1	7	6	14	6	0.04899
24A	2	1	27	17	2	0	8	4	8	14	0.02917
24B	2	1	35	1	2	0	10	4	3	16	0.01982
24C	2	1	23	15	2	0	8	2	7	12	0.02230
25	4	1	123	1	4	0	47	11	3	59	0.11957
26	4	1	13	9	2	0	5	0	4	6	0.02003
27A	2	2	27	1	0	2	4	8	3	6	0.03113
27B	4	2	121	9	0	2	10	49	50	8	0.30441
27C	6	2	477	47	0	2	35	202	223	16	1.37043
28A	2	1	15	1	0	1	5	2	1	7	0.00641
28B	5	1	35	1	0	1	6	11	6	12	0.09240
28C	9	1	61	1	0	1	14	16	10	21	0.54033
29A	10	1	5	1	0	1	2	0	0	3	0.00739
29B	50	1	5	1	0	1	2	0	0	3	0.14038
29C	100	1	5	1	0	1	2	0	0	3	0.82233
30	9	1	NC	9	2	0	1938	8028	9839	166	199.205
31	5	1	NC	1	0	1	4975	4998	9838	63	50.5877

Table A.2: Performance of the B&B algorithm with one of its different features switched off.

	NIT			CPU(S)		
Ex.	KEXT OFF	NSEXT OFF	INCLT OFF	KEXT OFF	NSEXT OFF	INCLT OFF
1	31	39	271	0.02273	0.02157	0.08718
2	117	161	589	0.08922	0.09839	0.16702
3	43	79	215	0.06802	0.06866	0.08733
4	57	9675	223	0.47569	11.7817	0.49463
5	17	111	73	0.01982	0.10864	0.06693
6	2353	NC	4097	5.18078	28.6136	5.32686
7	545	10527	739	1.31617	10.3111	1.25520
8	43	67	51	0.02551	0.04981	0.02690
9	137	257	229	0.09176	0.15303	0.12845
10	7	7	27	0.00343	0.01984	0.00648
11	7	13	29	0.00435	0.00904	0.00885
12	3	3	31	0.00179	0.01708	0.00705
13	3	3	47	0.00283	0.00236	0.01013
14	35	29	51	0.02418	0.01454	0.01419
15	5	3	51	0.00307	0.01844	0.01598
16	13	19	61	0.01122	0.01348	0.02779
17	21	33	59	0.01364	0.02079	0.02694
18	17	23	59	0.00577	0.00971	0.01463
19	21	39	51	0.01165	0.02784	0.02110
20	15	25	29	0.00537	0.01242	0.00789
21	13	19	63	0.00452	0.00790	0.01664
22	5	5	67	0.00408	0.00707	0.02548
23	51	53	101	0.05537	0.06434	0.07441
24A	27	53	65	0.02702	0.03997	0.11748
24B	35	47	69	0.02807	0.03901	0.02824
24C	27	83	69	0.19316	0.30068	0.02824
25	287	173	123	0.30730	0.10276	0.10074
26	15	17	93	0.02062	0.01916	0.08846
27A	27	51	135	0.03011	0.05628	0.07100
27B	125	537	353	0.33809	0.48194	0.35421
27C	489	5703	829	1.22779	5.87245	1.45260
28A	15	19	63	0.00796	0.01085	0.02135
28B	35	681	171	0.08664	0.66921	0.13175
28C	61	3049	315	0.62469	4.49951	0.84279
29A	5	5	341	0.01633	0.00505	0.22033
29B	5	5	1825	0.16394	0.07638	10.6647
29C	5	5	3759	0.86887	0.40601	135.078
30	NC	NC	NC	200.852	106.720	203.328
31	NC	NC	NC	55.6053	18.3440	49.8362

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B

Solutions of the Test Problems

Table B.1: Solutions of Example 1

$\#S$	x_1	x_2
1	3	2
2	−3.77931025	−3.28318599
3	−2.80511808	3.13131251
4	3.58442834	−1.84812652

Table B.2: Solutions of Example 2

#S	x_1	x_2
1	3	2
2	-3.77931025	-3.28318599
3	-2.80511808	3.13131251
4	3.58442834	-1.84812652
5	-3.07302575	-0.08135304
6	-0.27084459	-0.92303855
7	-0.12796134	-1.95371498
8	3.38515418	0.07385187
9	0.08667750	2.88425470

Table B.3: Solutions of Example 3

$\#S$	x	$T\ (K)$
1	0.001580	300.37
2	0.3326	347.89
3	0.98302	445.45

Table B.4: Solution of Example 4

$\#S$	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0.32287	0.0092235	0.046017	0.61817	0.0037168	0.57671	2.977863451

Table B.5: Solution of Example 5

$\#S$	x_1	x_2
1	0.0000145067	6.89335287

Table B.6: Solutions of Example 6

$\#S$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	0.67155	0.74095	0.95189	−0.30643	−0.96381	0.26658	0.40464	−0.91447
2	0.67155	0.74095	0.95189	−0.30643	−0.96381	−0.26658	0.40464	0.91447
3	0.67155	0.74095	0.95189	−0.30643	0.96381	0.26658	0.40464	−0.91447
4	0.67155	0.74095	0.95189	−0.30643	0.96381	−0.26658	0.40464	0.91447
5	0.67155	0.74095	−0.65159	−0.75857	−0.96254	−0.27112	−0.43757	0.89918
6	0.67155	0.74095	−0.65159	−0.75857	−0.96254	0.27112	−0.43757	−0.89918
7	0.67155	0.74095	−0.65159	−0.75857	0.96254	−0.27112	−0.43757	0.89918
8	0.67155	0.74095	−0.65159	−0.75857	0.96254	0.27112	−0.43757	−0.89918
9	0.16443	−0.98638	−0.94706	−0.32104	−0.99823	0.05941	0.41103	−0.91162
10	0.16443	−0.98638	−0.94706	−0.32104	−0.99823	−0.05941	0.41103	0.91162
11	0.16443	−0.98638	−0.94706	−0.32104	0.99823	0.05941	0.41103	−0.91162
12	0.16443	−0.98638	−0.94706	−0.32104	0.99823	−0.05941	0.41103	0.91162
13	0.16443	−0.98638	0.718452	−0.69557	−0.99796	0.06377	−0.52780	−0.84936
14	0.16443	−0.98638	0.718452	−0.69557	−0.99796	−0.06377	−0.52780	0.84936
15	0.16443	−0.98638	0.718452	−0.69557	0.99796	−0.06377	−0.52780	0.84936
16	0.16443	−0.98638	0.718452	−0.69557	0.99796	0.06377	−0.52780	−0.84936

Table B.7: Solutions of Example 7

#S	x_1	x_2	x_3	x_4	x_5
1	1	1	1	1	1
2	0.916	0.916	0.916	0.916	1.418

Table B.8: Solutions of Example 8

#S	x
1	0.3652473891
2	0.3705177763

Table B.9: Solutions of Example 9

#S	x
1	5.214649094
2	1.440978037
3	10.00823395
4	14.75389597
5	18.58437612

Table B.10: Solutions of Example 27A

#S	x_1	x_2
1	−0.4532892	−0.3854050
2	1.6455966	0.2604066

Table B.11: Solutions of Example 27B

#S	x_1	x_2	x_3	x_4
1	−0.5545767	−0.6394204	−0.5907007	−0.4152683
2	1.8002386	−0.0405013	−0.4625116	−0.3874338

Table B.12: Solutions of Example 27C

$\#S$	x_1	x_2	x_3	x_4	x_5	x_6
1	−0.5685882	−0.6761748	−0.6871807	−0.6649009	−0.5958543	−0.4163735
2	1.8284628	−0.1005822	−0.5752215	−0.6434210	−0.5915113	−0.4154422

Table B.13: Solution of Example 28A

$\#S$	x_1	x_2
1	−0.4273046	−0.42730462

Table B.14: Solution of Example 28B

$\#S$	x_1	x_2	x_3	x_4	x_5
1	−0.4283028	−0.4765965	−0.5196377	−0.5588619	−0.5588619

Table B.15: Solution of Example 28C

Variable	Value
x_1	−0.42830
x_2	−0.47659
x_3	−0.51965
x_4	−0.55809
x_5	−0.59250
x_6	−0.62450
x_7	−0.62321
x_8	−0.62226
x_9	−0.58799

Table B.16: Solution of Example 30

Variable	Value
x_1	0.899999
x_2	0.449987
x_3	1.000005
x_4	2.000064
x_5	7.999975
x_6	7.999710
x_7	5.000035
x_8	0.999988
x_9	2.000050

Table B.17: Solution of Example 31

$\#S$	x_1	x_2	x_3	x_4	x_5
1	0.0034301	31.32700	0.0683498	0.8595291	0.0369624

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] U. Ascher and M. R. Osbourne. A note on solving nonlinear equations and the natural criterion function. *Journal of Optimization Thoery and Applications*, 55(1):147–152, 1987.

[2] H. G. Bock, E. Kostina, and Johannes P. Schloder. On the role of natural level functions to achieve global convergence for damped-Newton method. *System Modelling and Optimization: Methods, Theory and Applications*, 2000.

[3] P. Deuflhard. A modified newton method for the solution of ill-conditioned systems of equations with applications to multiple shooting. *Numer. Math.*, 22:289–315, 1974.

[4] I. S. Duff and J. K. Reid. A Fortran code for direct solution of sparse unsymmetric linear systems of equations. Technical report, Rutherford Appleton Laboratory, October 1993.

[5] A. Griewank. *Evaluating Derivatives : Principles and Techniques of Algorithmic Differentiation.* SIAM, Philadelphia, 1987.

[6] Kenneth S. Gritton, J. D. Seader, and Wen-Jing Lin. Global homotopy continuation procedures for seeking all roots of a nonlinear equation. *Computers & Chem. Engg.*, 25:1003–1019, 2001.

[7] R. B. Kearfott and M. Novoa. INTBIS, a portable interval-Newton/bisection package. *ACM Transactions on Math. Soft.*, 16:152, 1990.

[8] M. Kuno and J. D. Seader. Computing all real solutions to systems of nonlinear equations with global fixed-point homotopy. *Industrial & Engineering Chemistry Research*, 27:1320–1329, 1988.

[9] Hongkun Liang and Mark A. Stadtherr. Computation of interval extensions using Berz-Taylor polynomial models. Los Angeles, CA, November 2000. AIChE Annual Meeting.

[10] W. J. Lin, J. D. Seader, and T. L. Wayburn. Computing multiple solutions to systems of interlinked separation columns. *AIChE*, 33(6):886–897, 1987.

[11] L. Luksan and J. Vlcek. A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming*, 83(3):373–391, November 1998.

[12] L. Luksan and J. Vlcek. Globally convergent variable metric method for convex nonsmooth unconstrained minimization. *Journal of Optimization Theory Application*, 102(3):593–613, September 1999.

[13] L. Luksan and J. Vlcek. Algorithm for non-differentiable optimization. *ACM Transactions on Mathematical Software*, 2(2):193–213, 2001.

[14] M. M. Makela. Survey of bundle methods for nonsmooth optimization. *Optimization Methods and Software*, 17(1):1–29, 2002.

[15] C. D. Maranas and C. A. Floudas. Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182, 1995.

[16] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I-convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[17] R. E. Moore. *Interval Analysis.* Prentice Hall, Engelwood Cliffs, NJ, 1966.

[18] Jorge J. More, Burton S. Garbow, and Kenneth E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Softwares*, 7(1):17–41, March 1981.

[19] A. Neumaier. *Interval methods for systems of equations.* Cambridge University Press, 1990.

[20] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, San Diego, California, 1970.

[21] B. Chachuat P. I. Barton and A. Mitsos. Subgradient propagation for McCormick relaxations. *SIAM Journal on Optimization*, 2007. submitted.

[22] C. A. Schnepper and M. A. Stadtherr. Robust process simulation using interval methods. *Computers & Chem. Engg.*, 20:187–199, 1996.

[23] J. D. Seader, M. Kuno, W. J. Lin, S. A. Johnson, K. Unsworth, and J. W. Wiskin. Mapped continuation methods for computing all solutions to general systems of nonlinear equations. *Computers & Chem. Engg.*, 14(1):71–85, 1990.

[24] J. E. Tolsma and P. I. Barton. DAEPACK; an open modeling environment for legacy models. *Industrial & Engineering Chemistry Research*, 39(6):1826–1839, 2000.

[25] C. E. Wilhelm and R. E. Swaney. Robust solution of algebraic process modelling equations. *Computers and chem. Engg.*, 18(6):511–531, 1994.