Adaptive use of extended systems for the efficient verified solution of nonlinear systems

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

am Fachbereich Mathematik und Naturwissenschaften der
Bergischen Universität Wuppertal eingereichte

Dissertation
von

Dipl.-Math. Elke Just

Referent:	Prof. Dr. Bruno Lang
Koreferent:	Prof. Dr. Andreas Frommer
Dissertation eingereicht am:	03.07.2013
Tag der Disputation:	23.09.2013

Abstract

This thesis discusses existing methods for the reliable solution of nonlinear systems of equations and presents various approaches to improve these methods. The reliability of all methods is ensured by the application of interval analysis.

In particular, the issue of utilizing extended systems is addressed. These systems are obtained from the given system of equations by the introduction of additional variables for suitable subterms.

An existing approach to control the usage of extended systems in an overall branch-and-bound scheme is presented. Subsequently, a new, adaptive strategy is developed. This adaptive strategy allows to exploit the advantages of the different extended systems well-aimed and effectively. We further give detailed considerations concerning the usage of the preconditioned interval Newton method on the extended systems.

Although the adaptive strategy provides the title for this work, the discussion of techniques for the reliable solution of nonlinear systems is interspersed with further deliberations for improvements.

Another significant part of this work is dedicated to the issue of verification tests. These tests can verify the existence and even uniqueness of solutions in a given bounded box. Existing verification methods are discussed and different modifications of the tests are examined. A general scheme to apply verification tests for square systems is given. For non-square systems we discuss the verification of square subsystems and especially possibilities to fix the variables of underdetermined systems.

Our theoretical considerations are supported by numerical studies within the framework of the software SONIC, in which the proposed algorithms and strategies have been implemented and tested.

Zusammenfassung

Im Rahmen dieser Arbeit werden verschiedene Ansätze zur Verbesserung existierender Methoden zum verifizierten Lösen nichtlinearer Gleichungssysteme diskutiert und diverse Ansätze zur Verbesserung dieser Methoden vorgestellt. Die Verlässlichkeit dieser Methoden wird durch die Anwendung der Intervallrechnung gewährleistet.

Besondere Aufmerksamkeit wird auf die Betrachtung erweiterter Systeme gelegt. Diese entstehen, wenn in ein vorgegebenes System zusätzliche Variablen für geeignete Teilterme eingeführt werden.

Es wird ein bekannter Ansatz zur Nutzung erweiterter Systeme innerhalb eines Branch-and-Bound-Verfahrens vorgestellt. Anschließend wird ein neuer, adaptiver Ansatz entwickelt. Dieser adaptive Ansatz erlaubt es, die Vorteile der verschiedenen erweiterten Systeme gezielter und effektiver auszunutzen. Des Weiteren werden detaillierte Überlegungen zur Nutzung des präkonditionierten Intervall-Newton-Verfahrens auf den erweiterten Systemen angestellt.

Obwohl die adaptive Strategie den Titel dieser Arbeit liefert, ist die Diskussion der Techniken für die verlässliche Lösung nichtlinearer Gleichungssysteme durchsetzt mit Überlegungen zu weiteren Verbesserungsmöglichkeiten.

Ein weiterer signifikanter Teil dieser Arbeit ist dem Thema der Verifikationstests gewidmet. Betrachtet werden Methoden zur Verifikation der Existenz und Eindeutigkeit von Lösungen nichtlinearer Gleichungssysteme. Es werden Tests zur Verifikation von Lösungen diskutiert und Modifikationen dieser Tests untersucht. Zudem wird ein Schema zum Einsatz der Verifikationstests für quadratische Systeme angegeben. Für nichtquadratische Systeme wird die Verifikation quadratischer Untersysteme besprochen, insbesondere im Hinblick auf die Fixierung von Variablen für unterbestimmte Systeme.

Die theoretischen Ausführungen werden unterstützt durch numerische Studien im Rahmen des Programms SONIC, in dem die vorgeschlagenen Algorithmen und Strategien umgesetzt und getestet wurden.

Acknowledgement

I want to thank my advisor Prof. Dr. Bruno Lang for introducing me to the field of interval analysis as well as for his constant support with advice and expertise. I would also like to thank Prof. Dr. Andreas Frommer for acting as a referee as well as Prof. Dr. Walter Krämer and Prof. Dr. Markus Reineke for their willingness to participate in my doctoral committee.

Furthermore, I thank the former and present members of the Applied Computer Science Group at the University of Wuppertal for providing a supportive and inspiring working environment. In particular, I am gratefully indebted to Dr. Paul Willems, for providing answers to my questions in the early stage of this work, concerning interval analysis as well as details of the implementation of SONIC.

A further dept of gratitude is owed to Patrick Bauermann, Anke Just and Tobias Kellner whose comments and suggestions to earlier draft of this work helped to improve its readability.

Finally a special thanks goes to Markus Filzhuth for all his personal support.

Wuppertal, July 2013 *Elke Just*

Contents

Introduction

In contemplation, if a man begins with certainties he shall end in doubts;
but if he will be content to begin with doubts, he shall end in certainties.

Francis Bacon

Employing computers for calculating solutions of mathematical problems enables us to resolve many problems much faster and/or more accurately than possible by hand. Although, everybody dealing with calculations on computers should know that care has to be taken to avoid computational errors whenever solving a problem numerically. Some of these errors can be alleviated by the application of numerically stable methods. We face this restriction by space limitation when storing information physically. In addition, computers cannot even store some seemingly innocuous numbers. An often-quoted example is $0.1 = 1/10$. Why can this number not be stored—although only two digits are needed for its decimal representation? The simple answer is that computers do not work with decimal numbers, but with binary ones. The decimal number 0.1 is represented by $0.0\overline{0011}$ in the binary system. To be stored exactly it would thus require an infinite number of digits.

However, there are ways to calculate rigorous solutions despite the shortcomings of computers in representing numbers exactly. One of them is to enclose all numbers and uncertain values by intervals. *Interval analysis* provides fundamental rules and universal methods to deal with these intervals. Every method takes into account all possible sources of errors and provides guaranteed results. By contrast to earlier approaches for utilizing intervals in calculations, interval analysis includes automated error handling. Its main principle is to validate the assumptions of mathematical claims with the help of interval calculations.

In summary, interval analysis computes with guaranteed enclosures of given values instead of numbers of finite precision. It enables us to calculate rigorously and provides trustworthy results. This makes it a perfect tool for reliable computations.

Since the need for reliable numerical calculations arose as recently as computers were used for the solution of mathematical problems, interval analysis is a relatively young subject. First works relating to interval operations seem to be published around the year 1930. Nevertheless, R. E. Moore's "Interval Analysis" [Moo66], published in 1966, is generally taken to be the first important work on this subject. Since its publication interval analysis has become a field of interest for a number of researchers, augmenting it with more methods and new areas of application.

In practice, however, interval computations did not gain full acceptance for a long time. One reason for this is that the naive approach to replace numbers and operations in a given algorithm by their interval counterparts is in most cases bound to fall short due to vast overestimation of the desired result.

Meanwhile, appropriate interval methods have been developed for many fields of interest. Basic examples for sophisticated interval methods are those for enclosing the values of a function for all arguments in an interval. Of course, also more complex tasks such as solving systems of linear and nonlinear equations as well as optimization problems have been discussed before (as in [Han92], [Kea96b] and [Neu90]). Further fields of interval analysis include the verification of solutions (see Chapter 4) and the solution of differential equations [Ebl07]. Applications of interval analysis can, e.g., be found in satisfiability problems [FHT+07], in robotics and robust control (see for example [JKDW01]), the analysis of chemical processes [BBLSA04,BLMM01], and even in the search for Nash equilibria [KW10] and computer assisted proofs [CFL09, BLUW09, Fro01].

Many interval-related publications can be found in the *Reliable Computing* journal [webb]. Even more publications, historical information, fields of applications and software employing interval computations are listed in publications such as [Kea96a] and online [weba].

For most practical calculations, still floating point procedures are employed. They have been under development for some decades and quite a few have been highly optimized. But floating point computations cannot grant guaranteed solutions. They suffer not only from rounding errors in the given numbers, but also from the accumulation of rounding and truncation errors within the line of computation.

A common approach to check floating point computations is to calculate with different precisions of the utilized numbers. The results of those computations are assumed to be correct if their first significant digits coincide. This approach may be less time-consuming than to use interval methods, but it is not sufficient to ensure the correctness of the result. Indeed, it cannot even guarantee that the result approximates the solution. In an example given by Rump (published, e.g., in [Rum10, p. 9]) computations in three different precisions seem to lead to the same result. Thus, one would have to trust in the results if there is no possibility to check the results by other means. However, the computed results for this example

are not only plainly wrong, they do not even have the right sign! Interval analysis may not lead to a better approximation of the exact solution, but it provides us with an interval guaranteed to contain the solution and may draw attention to numerical instabilities.

Reliable computations are especially important for safety-critical applications. For them one should not rely on error-prone floating point calculations. This still holds true if reliable computations are more expensive with regard to computation time or memory.

Furthermore, with more research in interval analysis, interval methods will most probably become faster and provide more accurate results. And there already exist interval methods that are faster than their real-valued counterparts (for references see [Kea96a, p. 6]).

◇

This work aims at promoting the scientific progress in interval analysis. Its focus is the development and improvement of methods to solve nonlinear systems of equations rigorously and efficiently and of tests to verify solutions of those systems.

To be able to present these methods, basic definitions are introduced and the principles of interval analysis are discussed (see Chapter 1). Subsequently the problem of solving nonlinear systems of equations in an interval context is introduced formally. Afterwards, we give an overview of the methods utilized to solve this problem class (see Chapter 2). We augment this overview by several suggestions for improvements.

The idea and usage of extended systems is introduced in Chapter 3. Emphasis is put on strategies to solve nonlinear systems of equations even faster with the help of extended systems. As a result of the conducted studies, a new, adaptive strategy is proposed that improves the utilization of these systems. The new strategy is analyzed in detail.

Chapter 4 is dedicated entirely to verifying the existence and uniqueness of solutions in a predefined area. Various modifications of existing verification tests are investigated.

Most strategies proposed in this work have been integrated into the verified solver SONIC. Chapter 5 gives a general introduction of this program. Although this chapter only provides an overview and cannot serve as manual, users of SONIC may find helpful advice and an outline of the features of this tool.

Eventually, a review states the achievements of this work and gives pointers for possible future work.

Chapter 1

Notations and the basics of interval analysis

Information is the resolution of uncertainty.

Claude Shannon

We define and use most technical terms and notations as suggested in [KNN+02] and recent books about interval analysis. Experts in interval analysis may therefore safely skip this chapter. Note, however, that we utilize not only bounded intervals but also unbounded ones. If necessary, further information can be found in the list of used symbols in the appendix.

Most of the definitions and descriptions in this chapter are given regarding functions already implemented in the solver SONIC. For this reason, they mainly follow earlier work related to this program (especially [BBLW04], [Wil04] and [Bee06]). (In his dissertation Beelitz gave a comprehensive description of all functions implemented up to then. Unfortunately this work is written in German. For completeness and the reader's convenience we thus decided to give descriptions of the most important functions nonetheless.) For a comprising survey of interval analysis the reader may want to consult further literature such as [Moo66], [AH83], [Neu90], [Han92] or [JKDW01].

This chapter will introduce some basic definitions and claims concerning intervals and the arithmetic on intervals. Since the theoretical claims require exact computations, section 1.9 deals with the issue of transferring theory into practice and real numbers to machine intervals.

Notations that are not introduced in this chapter will be are announced where they are needed.

1.1 Algorithms, numerical results and fonts

1.1.1 Notation of algorithms

All algorithms that occur in this work are denoted in pseudo code and use common instructions. An example for the appearance is given by Algorithm 1.1. Further we often shorten constructs for more concise notations and paraphrase specific instructions in words.

Algorithm 1.1 NAME OF THE ALGORITHM ($parameter_1, \dots$)

```
1: for elements for which the body of the for loop is executed do
2:    instructions {comment}
3: end for
4: while condition do
5:    instructions
6:    continue {ends current pass of a loop}
7:    exit {terminates the algorithm}
8: end while
9: if condition then
10:   instructions
11: else
12:   instructions
13: end if
14: x ← y {assign variable x with value of y}
```

1.1.2 Numerical results

The theoretical considerations presented in this work are supported by numerical results. For this purpose newly established as well as modified methods have been implemented and evaluated with the help of numerical experiments.

Some heuristics in this work evolved from theoretical considerations about different approaches and subsequent systematic testing of the corresponding implementations. Herein the heuristics have been developed with regard to performance for a small set of test problems and evaluated using a larger set. The main goal of this development was to provide algorithms that are suited for a wide range of nonlinear systems while providing good results with default settings. Still all algorithms should be adjustable to specific problems.

The methods presented in this work are often assessed based on their computational costs. Those costs comprise computation time as well as memory requirements. Within this work the latter are not regarded explicitly, instead the needed number of "boxes"(defined below) for the main branch-and-bound algorithm is evaluated. Note that the computational costs of a specific strategy always de-

pends on several parameters and that the effects caused by the parameters often interfere.

All numerical tests were executed with the help of SONIC, a rigorous solver and optimizer. The acronym stands for **S**olver and **O**ptimizer for **N**onlinear Problems based on **I**nterval **C**omputation. If not stated otherwise, all timings and results were obtained by serial computations on an Intel®Core™2 Quad CPU Q9650 @ 3.00GHz employing the interval library C-XSC 2.2.4.

All timings displayed in this work are given in seconds and rounded to one decimal place. If those timings have been used to compute further values, we used measurements in higher precision. Note further that the tests have been conducted in different stages of the development of SONIC. Nonetheless, the results we compare directly are yielded by versions differing only in the evaluated parameters or methods.

1.1.3 Conventions

Some noticeable values are highlighted, especially in tables. They are either emphasized as **good** or *not so good*. *Parameters*, *problems* and *files* used in SONIC are highlighted as well.

Even before Chapter 5 will introduce SONIC in detail, we give some background information on the program close to the respective theoretical discussion. This is either done in separate sections or within the text, where appropriate.

Small text passages only related to the implementation in SONIC are detached by shaded background color. They can be skipped by the reader without missing theoretical deliberations.

1.2 Intervals

There exist several approaches on how to define intervals. The Kaucher notation, for example, uses midpoint and radius to define an interval. Another important example are complex intervals.

We use the most common definition and handle *intervals* as connected and closed sets of real numbers defined by a lower and an upper endpoint. In addition, we allow our intervals to be unbounded sets. This becomes an advantage when it comes to enclosing parameters in real world problems. For them it is not always possible to determine finite guaranteed bounds. Thus, if we could only handle intervals that are bounded by real numbers, we would have to introduce artificial bounds for those values. Since this is not necessary when using unbounded inter-

vals, we use them in theory as well as in the implementation in SONIC. Hence, we are not compelled to introduce artificial bounds, but are free to work with the truly known range of all values.

For representing the desired kind of interval, the set of real numbers $\mathbb{R}$ is extended by the two elements $-\infty$ and $+\infty$ fulfilling

$$-\infty < a < +\infty, \quad \text{for all } a \in \mathbb{R}.$$

$\mathbb{R}$ is thus augmented to the set of *extended real numbers* ${}^*\mathbb{R}$ by defining

$${}^*\mathbb{R} := \mathbb{R} \cup \{-\infty, +\infty\}.$$

Instead of writing $+\infty$ we will just write ∞ later. It is also worthwhile to stress explicitly that some expressions in the extended real numbers are not defined (such as $-\infty + \infty$). We evade these expressions by suppressing the—not very helpful—intervals $[-\infty, -\infty]$ and $[+\infty, +\infty]$.

To denote intervals, we write $\boldsymbol{x} = [\underline{x}, \overline{x}]$ with bold face indicating interval values. Thereby $\underline{x}$ is called the *lower bound*, *lower endpoint* or *infimum* and $\overline{x}$ the *upper bound*, *upper endpoint* or *supremum* of the interval $\boldsymbol{x}$.

Given this notation, an arbitrary non-empty interval in the extended real number system can be written as

$$\boldsymbol{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x},\ \underline{x} \in {}^*\mathbb{R}\backslash\{\infty\},\ \overline{x} \in {}^*\mathbb{R}\backslash\{-\infty\}\}\,.$$

A non-empty interval $\boldsymbol{x}$ is called *bounded* if its limits $\underline{x}$ and $\overline{x}$ are real numbers, it is called *half-bounded* if exactly one of its limits is finite. Every non-empty interval that is not bounded is called *unbounded*. (Thus also half-bounded intervals are unbounded.)

Note that neither bounded nor unbounded intervals contain the infinite values $+\infty$ and $-\infty$. Intervals only comprise elements in $\mathbb{R}$, whether when their borders lie in $\mathbb{R}$ or ${}^*\mathbb{R}$. For reasons of consistency of the denotation, unbounded intervals are denoted by $[a, \infty]$—although the common mathematical notation of this interval would be $[a, \infty)$. For the empty interval, containing no real number, we use the symbol $\emptyset$ for the empty set.

The set of all non-empty intervals is denoted as ${}^*\mathbb{IR}$, the set of all bounded intervals as $\mathbb{IR}$. We can define the latter by

$$\mathbb{IR} := \{[\underline{x}, \overline{x}] \in {}^*\mathbb{IR} \mid \underline{x}, \overline{x} \in \mathbb{R}\}\,.$$

The sets and their power sets (denoted by $\mathcal{P}$) are obviously connected by the following relations

$$\mathbb{IR} \subsetneq {}^*\mathbb{IR}, \quad \mathbb{IR} \subsetneq \mathcal{P}(\mathbb{R}) \text{ and } {}^*\mathbb{IR} \subsetneq \mathcal{P}({}^*\mathbb{R}).$$

A bounded interval $\boldsymbol{x}$ is called a *point interval* or *thin* if $\underline{x}$ equals $\overline{x}$. In other words, a point interval consists of exactly one real number. (In other publications

this kind of interval is also called a *punctual* or *degenerate interval*.) Every real number $x \in \mathbb{R}$ can be embedded into ${}^*\mathbb{IR}$ (and $\mathbb{IR}$) by identifying it with the point interval $[x, x]$. This interval corresponding to the number x is also denoted by $[x]$. Whenever we use real numbers in interval analysis we consider them as point intervals. In practice, we even have to embed numbers and intervals into the set of intervals with machine-representable limits (cf. section 1.9). In this case, a single number that cannot be represented on the computer has to be represented by an interval $\boldsymbol{x}$ with nonzero width.

An interval that is neither a point interval nor empty is called a *proper interval* or *thick* and comprises infinitely many real numbers. We further call an interval *positive* (*nonnegative* or *negative*) if it contains only positive (nonnegative or negative) numbers.

Definition 1.1 (Functions of intervals)
For non-empty intervals $\boldsymbol{x} \in {}^*\mathbb{IR}$ the following functions are defined.

- Infimum $\quad \inf(\boldsymbol{x}) := \underline{x} = \inf\{x \mid x \in \boldsymbol{x}\}$
- Supremum $\quad \sup(\boldsymbol{x}) := \overline{x} = \sup\{x \mid x \in \boldsymbol{x}\}$
- Midpoint $\quad \operatorname{mid}(\boldsymbol{x}) := \begin{cases} 0 & \text{if } \underline{x} = -\infty \text{ and } \overline{x} = \infty \\ \frac{\underline{x}+\overline{x}}{2} & \text{else} \end{cases}$
- Width *(also called* diameter*)* $\quad \operatorname{width}(\boldsymbol{x}) := \overline{x} - \underline{x}$
- Radius $\quad \operatorname{rad}(\boldsymbol{x}) := \frac{1}{2} \cdot \operatorname{width}(\boldsymbol{x}) = \frac{1}{2}(\overline{x} - \underline{x})$
- Magnitude $\quad |\boldsymbol{x}| = \operatorname{mag}(\boldsymbol{x}) := \max\{|x| \mid x \in \boldsymbol{x}\}$
- Mignitude $\quad \operatorname{mig}(\boldsymbol{x}) := \min\{|x| \mid x \in \boldsymbol{x}\}$
- Interior $\quad \operatorname{int}(\boldsymbol{x}) := \{x \in \boldsymbol{x} \mid \underline{x} < x < \overline{x}\}$
- Topological boundary $\quad \partial(\boldsymbol{x}) := \{\underline{x}, \overline{x}\}$

Note that midpoint and width of $\boldsymbol{x}$ are always defined and the width is nonnegative because the definition of intervals specifies $\underline{x} \leq \overline{x}$, $\underline{x} \neq \infty$ and $\overline{x} \neq -\infty$. Further the magnitude of an interval argument should not be confused with the absolute value for real-valued arguments.

These definitions are applicable for arbitrary intervals in ${}^*\mathbb{IR}$ since bounded as well as unbounded intervals only contain elements in $\mathbb{R}$. Furthermore, as intervals are a only a special kind of sets, we use the usual notation for subsets, supersets and the set operations, as intersection and union, with their traditional meaning. We speak of disjoint sets if their (pairwise) intersection is empty.

Note that the union of two disjoint intervals $[a, b]$ and $[c, d]$ from ${}^*\mathbb{IR}$ is not necessarily an interval itself. It can also be a more general kind of set. (In case of $b < c$ the union of the intervals is $[a, b] \cup [c, d]$.) To be able to work in ${}^*\mathbb{IR}$ one defines an operator enclosing an arbitrary set of real numbers with the smallest possible interval. It is called the *box operator* $\square$. For $P \in \mathcal{P}(\mathbb{R})$ it is given as

$$\square : \mathcal{P}(\mathbb{R}) \to {}^*\mathbb{IR}$$
$$\square(P) := \begin{cases} [\inf(P), \sup(P)] & \text{if } P \neq \emptyset, \\ \emptyset & \text{else.} \end{cases}$$

Thus we get an interval—but have to accept overestimation of the set P. (For our example, $\square([a, b] \cup [c, d])$ yields the interval $[a, d]$. The overestimation thus comprises all numbers between b and c.) Having to enclose arbitrary sets by intervals is one of the reasons forcing us to deal with overestimation whenever calculating in ${}^*\mathbb{IR}$.

SONIC can store finite unions of intervals. As long as we do not make use of the information about gaps (cf. section 2.2.1), computations are conducted with the hull of this union. (So our theorems do not need to be reformulated for unions of intervals.)

1.3 Vectors of real numbers and intervals

We denote vectors in the extended real numbers ${}^*\mathbb{R}$ as

$$x = (x_1, \ldots, x_n)^T \in {}^*\mathbb{R}^n$$

and vectors of intervals as

$$\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T \in {}^*\mathbb{IR}^n$$

with subscripts indicating components. In general, interval values are set in bold face to separate them from real-valued ones. There is no typographical distinction between numbers and vectors. Following the common usage in mathematics we use lower case for both of them. Dimensions will be apparent by context.

Interval vectors in $\mathbb{IR}^n$ can be interpreted as (possibly hyperdimensional) cuboids and as boxes in the special case $\mathbb{IR}^3$. We generalize this term and call every arbitrary, possibly unbounded, interval vector $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ a *box*.

A box is called bounded if all its components are bounded and unbounded if at least one of its components is unbounded. It is called half-bounded if it contains at least one half-bounded component and consists only of bounded and

half-bounded components. Furthermore, if a box $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ is contained in another box $\boldsymbol{y} \in {}^*\mathbb{IR}^n$ in the sense of

$$\boldsymbol{x} \subseteq \boldsymbol{y} \quad \Leftrightarrow \quad \boldsymbol{x}_i \subseteq \boldsymbol{y}_i \text{ for all } i \in \{1, \ldots, n\},$$

we call $\boldsymbol{x}$ a *subbox* of $\boldsymbol{y}$ and $\boldsymbol{y}$ a *superbox* of $\boldsymbol{x}$.

In the special case of decomposing a box $\boldsymbol{x}$ into k subboxes $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}$ fulfilling

$$\boldsymbol{x} = \boldsymbol{x}^{(1)} \cup \ldots \cup \boldsymbol{x}^{(k)}$$

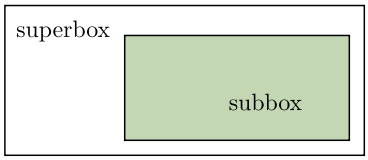
and

$$\text{int}\,(\boldsymbol{x}^{(i)}) \cap \text{int}\,(\boldsymbol{x}^{(j)}) = \emptyset \text{ for all } i, j \in \{1, \ldots, k\}$$

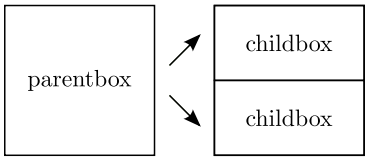
we speak of a *subdivision* of box $\boldsymbol{x}$ and say that $\boldsymbol{x}$ is *subdivided* into the set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}\}$. If only the first condition is fulfilled, we speak of a *covering* of box $\boldsymbol{x}$. In both cases we say that we *subdivide* the box $\boldsymbol{x}$.

When subdividing in the branch-and-bound algorithm we use denotations resembling terms of genealogy. We speak of a *parentbox* $\boldsymbol{x}$ for the box we subdivide and of *childboxes* for the resulting boxes $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}$. The childboxes form a covering or subdivision of the parentbox. An *ancestorbox* of a box $\boldsymbol{x}$ is a box that was subdivided into childboxes once or repeatedly with $\boldsymbol{x}$ being one of the evolving parts. Owing to these definitions, trivially every box is a subbox of all its parent- and ancestorboxes. The concepts of super- and subboxes as well as parent- and childboxes is depicted by Figure 1.1.

Note that a slight change of meaning will prove to be beneficial in Chapter 2. In this chapter the terms ancestorbox, parentbox and childbox are also used if boxes are not only subdivided but on the back of this also contracted.



(a) Superbox and subbox



(b) Parentbox and two childboxes

Figure 1.1: Relations of boxes

1.4 Matrices of real numbers and intervals

Matrices are denoted by capital letters. They are written as

$$A = (a_{i,j})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$$

for matrices A from ${}^*\mathbb{R}^{m \times n}$ and

$$\boldsymbol{A} = (\boldsymbol{a}_{i,j})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$$

for interval matrices $\boldsymbol{A}$ from ${}^*\mathbb{IR}^{m\times n}$.

As well as real numbers are embedded into ${}^*\mathbb{IR}$, vectors and matrices of real numbers are embedded into the set of interval vectors ${}^*\mathbb{IR}^n$ and matrices ${}^*\mathbb{IR}^{m\times n}$, respectively, by identifying every component with the corresponding point interval.

As usual, a square matrix in $\mathbb{R}^{n\times n}$ is denoted as *regular* or *nonsingular* if it is invertible. A square interval matrix in ${}^*\mathbb{IR}^{n\times n}$ is labeled *regular* or *nonsingular* if it contains only regular matrices in $\mathbb{R}^{n\times n}$. In any other case, both matrix types are called *singular*.

Of course, the nonsingularity of an interval matrix $\boldsymbol{A}$ can be checked in different ways. According to Beelitz [Bee06] an appropriate way is to show

$$||\boldsymbol{I} - \Delta\boldsymbol{C}\cdot\boldsymbol{A}||_\infty < 1$$

using the inverse midpoint preconditioner $C = (\operatorname{mid}(\boldsymbol{A}))^{-1}$. Further conditions for nonsingularity may be found in [RR95] or [Roh09].

1.5 Functions of interval vectors and matrices

For vectors and matrices of real numbers we use the operations of linear algebra in the usual sense. The box operator is applied componentwise, so for $P \in \mathcal{P}(\mathbb{R}^n)$ we get $\square(P) := (\square(P_1),\ldots,\square(P_n))^T$.

If possible, functions defined for intervals are transferred to work on interval vectors by applying the original functions to every component of the interval vector. By doing so we get the following definitions.

Definition 1.2 (Functions of interval vectors)
For interval vectors $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ we define the following functions.

$$\begin{aligned}
\inf(\boldsymbol{x}) &:= (\inf(\boldsymbol{x}_1),\ldots,\inf(\boldsymbol{x}_n))^T = (\underline{x}_1,\ldots,\underline{x}_n)^T \\
\sup(\boldsymbol{x}) &:= (\sup(\boldsymbol{x}_1),\ldots,\sup(\boldsymbol{x}_n))^T = (\overline{x}_1,\ldots,\overline{x}_n)^T \\
\operatorname{mid}(\boldsymbol{x}) &:= (\operatorname{mid}(\boldsymbol{x}_1),\ldots,\operatorname{mid}(\boldsymbol{x}_n))^T \\
\operatorname{width}(\boldsymbol{x}) &:= (\operatorname{width}(\boldsymbol{x}_1),\ldots,\operatorname{width}(\boldsymbol{x}_n))^T \\
\operatorname{rad}(\boldsymbol{x}) &:= (\operatorname{rad}(\boldsymbol{x}_1),\ldots,\operatorname{rad}(\boldsymbol{x}_n))^T \\
\operatorname{mag}(\boldsymbol{x}) = |\boldsymbol{x}| &:= (|\boldsymbol{x}_1|,\ldots,|\boldsymbol{x}_n|)^T \\
\operatorname{mig}(\boldsymbol{x}) &:= (\operatorname{mig}(\boldsymbol{x}_1),\ldots,\operatorname{mig}(\boldsymbol{x}_n))^T \\
\operatorname{int}(\boldsymbol{x}) &:= (\operatorname{int}(\boldsymbol{x}_1),\ldots,\operatorname{int}(\boldsymbol{x}_n))^T
\end{aligned}$$

In addition, the volume of a box $\boldsymbol{x}$ *is defined as*

$$\mathrm{vol}(\boldsymbol{x}) := \prod_{i=1}^{n} \mathrm{width}\,(\boldsymbol{x}_i)$$

and its topological boundary as

$$\partial\boldsymbol{x} := \bigcup_{i=1}^{n} \left((\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \underline{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)^T \cup (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \overline{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)^T\right).$$

As for interval vectors, we transfer the functions inf, sup, mid, width and $\square$ componentwise to interval matrices. Further, for each differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$, the *Jacobian matrix* comprising all first-order partial derivatives of f is denoted as Df. For $m = 1$ we use ∇f to denote the gradient of a function f. For twice differentiable functions the matrix of the second order partial derivatives is written as $D^2 f$. For $m = 1$, matrix $D^2 f$ is referred to as the *Hessian matrix* of the function f.

1.6 Interval arithmetic

As for real numbers, one has to define operations and elementary arithmetic functions for intervals in order to work with them. The proposed interval arithmetic is called *extended* (since it is working on intervals with endpoints in the extended real numbers). The extended arithmetic operations are mainly adopted from the study in [HvEW98].

Note that we confine ourselves to the discussion of real-valued functions $f : \mathbb{R}^n \to \mathbb{R}$. One can easily generalize the definitions to functions of higher dimensions by applying the given definitions to their components. Furthermore, we assume to calculate exactly in the following. In section 1.9 we will discuss some further aspects that have to be considered when implementing interval analysis on a computer.

Definition 1.3 (Range of a function)
May $f : \mathbb{R}^n \to \mathbb{R}$ *be a real-valued function, then the* range *of* f *over* $D \in \mathcal{P}(\mathbb{R}^n)$ *is defined as*

$$\begin{aligned} \mathrm{range}_f &: \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}) \\ \mathrm{range}_f(D) &= \{y \in \mathbb{R} \mid \textit{ it exists an } x \in D \textit{ with } f(x) = y\}. \end{aligned}$$

For the range over a box $\boldsymbol{x} \in \mathbb{IR}^n$ *we get the special case*

$$\mathrm{range}_f(\boldsymbol{x}) = \{y \in \mathbb{R} \mid \textit{ it exists an } x \in \boldsymbol{x} \textit{ with } f(x) = y\}.$$

Since also unbounded boxes only have elements in the real numbers, the last formula can be applied for unbounded boxes $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ *as well leading to the definition*

$$\begin{aligned} \text{range}_f &: {}^*\mathbb{IR}^n \to \mathcal{P}(\mathbb{R}) \\ \text{range}_f(\boldsymbol{x}) &= \{y \in \mathbb{R} \mid \textit{ it exists an } x \in \boldsymbol{x} \textit{ with } f(x) = y\} \end{aligned}$$

where $\text{range}_f(\boldsymbol{x})$ *is called the* range of f over the interval $\boldsymbol{x}$*.*

The range provides us with the set of all output values of function f for arguments x in $\boldsymbol{x}$ for which the function is defined. Note that the range of f over $\boldsymbol{x}$ is defined even if f is not defined for every real number x in the interval $\boldsymbol{x}$. For instance, the function f defined by $f(x) = 1/x$ is not defined for $x = 0$, but $\text{range}_f([0, 1]) = [1, \infty]$.

The range can be an arbitrary subset of the real numbers. Thus, if we want to work only with intervals and boxes, a further definition is needed.

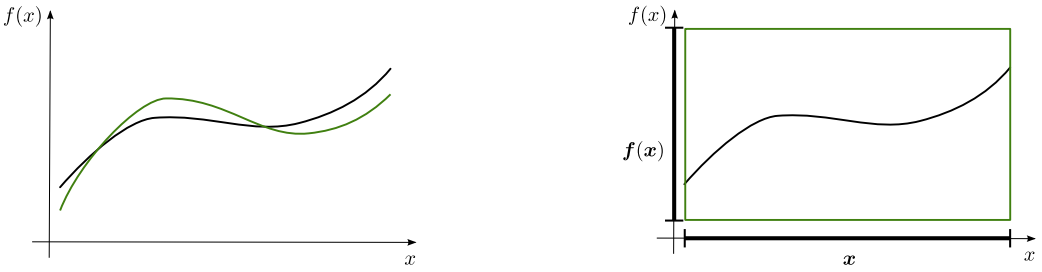
Definition 1.4 (Function enclosure)
Let $f : \mathbb{R}^n \to \mathbb{R}$ *be a real-valued function. Every function* $\boldsymbol{f} : {}^*\mathbb{IR}^n \to {}^*\mathbb{IR}$ *fulfilling*

$$\boldsymbol{f}(\boldsymbol{x}) \supseteq \text{range}_f(\boldsymbol{x}) \quad \textit{for all } \boldsymbol{x} \in {}^*\mathbb{IR}^n$$

is called a function enclosure *of* f*.*

In other publications a function enclosure is also called an *interval enclosure* or *inclusion function*. If it is clear that we refer to a function, we will also just speak of the *enclosure*. For clear notations the function enclosure will be denoted as $[f]$ instead of $\boldsymbol{f}$ whenever appropriate.

The crucial point to be derived from the definition is that every function enclosure is guaranteed to contain the range of the function. For reliable computations this is an advantage over using an approximation of the function (cf. also Figure 1.2).



(a) Evaluation by approximation

(b) Evaluation by function enclosure

Figure 1.2: Approaches to evaluate a function

In the past, different methods to compute function enclosures have been developed. These differ widely in regard to the quality of the results and the computational costs. Quality herein refers to the sharpness of a function enclosure.

Definition 1.5 (Sharpness of a function enclosure)
We say that a function enclosure $\boldsymbol{f}_1(\boldsymbol{x})$ *is* better *or* sharper *than another enclosure* $\boldsymbol{f}_2(\boldsymbol{x})$ *for the same function over a box* $\boldsymbol{x}$ *if* $\boldsymbol{f}_1(\boldsymbol{x}) \subset \boldsymbol{f}_2(\boldsymbol{x})$.

Of course, we are interested in minimizing the overestimation of the true range. Thus we want our function enclosures to be as sharp as possible. Some examples for function enclosures and a further measure for the quality of function enclosures can be found in section 1.8.

A first, very simple function enclosure is the function always returning the interval $[-\infty, \infty]$. Moreover, it is the one with the maximum width and suited for every function. However, this function enclosure is completely useless in practice.

On the other hand, if the range of f over each $\boldsymbol{x}$ is an interval itself, the sharpest possible function enclosure is given by

$$\boldsymbol{f}(\boldsymbol{x}) = \mathrm{range}_f(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \in {}^*\mathbb{IR}^n.$$

Unfortunately, for general functions, the range does not always yield single intervals. If the range is no interval, the best enclosure one can attain is

$$\boldsymbol{f}(\boldsymbol{x}) = \square(\mathrm{range}_f(\boldsymbol{x})) \quad \text{for all } \boldsymbol{x} \in {}^*\mathbb{IR}^n.$$

Consider, for example, the range of $f(x) = 1/x$ over the box $[-1, 1]$. It is given as $\mathrm{range}_f([-1, 1]) = [-\infty, -1] \cup [1, \infty]$. The smallest interval enclosing this range is $[-\infty, \infty]$.

Still, for some functions the range over a box is an interval and therefore itself the sharpest possible function enclosure. Due to the intermediate value theorem, this property is fulfilled for the important class of continuous functions. If the function is monotonic as well, one can even attain explicit formulae for the range. By virtue of these formulae we do not need to consider an infinite number of real numbers to determine the range, it is enough to compute with the endpoints of the interval arguments.

Note that explicit formulae are absolutely necessary when implementing interval analysis, since we cannot apply the theoretical formula for the range of a function when working with a computer but are bound to methods requiring only finitely many operations. For instance, the ranges of the arithmetic operations "+", "−" and "·" over intervals $[a, b]$ and $[c, d]$ can be computed using formulae (1.1), (1.2) and (1.3) because the functions are continuous, their ranges over intervals are intervals themselves and thereby the best possible enclosures. (We also use the shorter notation xy instead of $x \cdot y$ when a multiplication is to be applied.)

$$[a, b] + [c, d] = [a + c, b + d] \tag{1.1}$$
$$[a, b] - [c, d] = [a - d, b - c] \tag{1.2}$$
$$[a, b] \cdot [c, d] = [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}] \tag{1.3}$$

A proof of these formulae for unbounded intervals can be found in [HvEW98].

We can further give exact ranges for monotonic continuous functions, as for the exponential function over a bounded box $[a, b] \in \mathbb{IR}$

$$\exp([a, b]) = [\exp(a), \exp(b)].$$

Moreover, this is also possible for piecewise monotonic functions like sine or cosine.

If we want to extend division to intervals, we can make use of the following formula for the range given e.g. in [Bee06].

$$\frac{[a,b]}{[c,d]} = \begin{cases} [a,b] \cdot [1/d, 1/c] & \text{if} \quad 0 \notin [c,d] \\ \emptyset & \text{if} \quad c = d = 0 \\ [a/d, \infty] & \text{if} \quad a \geq 0, c = 0 \\ [-\infty, \infty] & \text{if} \quad 0 \in [a,b], c = 0 \\ [-\infty, b/d] & \text{if} \quad b \leq 0, c = 0 \\ [-\infty, a/c] \cup [a/d, \infty] & \text{if} \quad a > 0, c < 0 < d \\ [-\infty, \infty] & \text{if} \quad 0 \in [a,b], c < 0 < d \\ [-\infty, b/d] \cup [b/c, \infty] & \text{if} \quad b < 0, c < 0 < d \\ [-\infty, a/c] & \text{if} \quad a \geq 0, d = 0 \\ [-\infty, \infty] & \text{if} \quad 0 \in [a,b], d = 0 \\ [b/c, \infty] & \text{if} \quad b \leq 0, d = 0 \end{cases} \tag{1.4}$$

In [Wil04, page 30] Willems shows that the division for intervals defined as in formula (1.4) is not sufficient for solving a multiplication for a variable without losing solutions. Because this operation is essential for the methods described in section 2.3.1, we need another division operator.

Definition 1.6 (Relational division)
The relational division *for bounded intervals* $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IR}$

$$\boldsymbol{x} \oslash \boldsymbol{y} := \{z \in \mathbb{R} \mid \text{ there exist } x \in \boldsymbol{x}, y \in \boldsymbol{y} \text{ with } x = z \cdot y\}$$

was defined by Ratz [Rat96] and is computed via the formula

$$[a,b] \oslash [c,d] := \begin{cases} [a,b] \cdot [1/d, 1/c] & \textit{if} \quad 0 \notin [c,d] \\ [-\infty, \infty] & \textit{if} \quad 0 \in [a,b] \cap [c,d] \\ [b/c, \infty] & \textit{if} \quad b < 0, c < d = 0 \\ [-\infty, b/d] \cup [b/c, \infty] & \textit{if} \quad b < 0, c < 0 < d \\ [-\infty, b/d] & \textit{if} \quad b < 0, 0 = c < d \\ [-\infty, a/c] & \textit{if} \quad 0 < a, c < d = 0 \\ [-\infty, a/c] \cup [a/d, \infty] & \textit{if} \quad 0 < a, c < 0 < d \\ [a/d, \infty] & \textit{if} \quad 0 < a, 0 = c < d \\ \emptyset & \textit{if} \quad 0 \notin [a,b], c = d = 0. \end{cases} \tag{1.5}$$

The relational division has been extended to be used with unbounded intervals in [HJvE99].

Note that the result of the relational division operation is not always a single interval but may consist of the union of two disjoint intervals. Of course, this union can simply be enclosed in one single interval to achieve a function enclosure. By doing so, however, we lose the information about the "gap", which does not contain any solution. As mentioned previously, in some cases we exploit this information (see section 2.2.1).

As already shown in [HJvE99], one should always prefer the standard division (1.4) if we do not want to solve for a variable but to compute function enclosures because it provides sharper enclosures. To be precise the following lemma has been proved.

Lemma 1.1 ([HJvE99, Theorem 2])
The inclusion

$$\frac{\boldsymbol{x}}{\boldsymbol{y}} \subseteq \boldsymbol{x} \oslash \boldsymbol{y}$$

holds for all $\boldsymbol{x}, \boldsymbol{y} \in {}^*\mathbb{IR}$

To get to know the formulae for more functions see, e.g., [Neu90]. In the following, all arithmetic operations not applied to real numbers are to be understood as the corresponding interval operations. As long as nothing else is stated, we refer to the function f if the arguments are real numbers and to an enclosure $\boldsymbol{f}$ for interval arguments.

1.7 Properties of interval arithmetic

In the following, we want to highlight some of the effects distinguishing interval computations from real-valued ones.

1.7.1 Dependency problem

In interval arithmetic, and interval analysis in general, the intervals replacing variables that occur more than once in an arithmetic expressions are not "connected". Each appearance of a variable is handled independently of any other. As a consequence, in a computation, one might calculate with the upper bound for one occurrence of the variable, with the lower bound for another and with an inner point for a third occurrence. Hoefkens [Hoe01] described this property as follows.

> As far as interval arithmetic is concerned ... two intervals ... are two different entities with no relation between them, and the fact that they have the same endpoints is seen as a mere coincidence.

In the field of interval analysis this behavior is referred to as the *dependency problem*. It is one of the main obstacles in interval computations, although it is not a specific problems of intervals, but inherent to arbitrary sets.

Example 1.1
If we subtract a variable from itself using real numbers, we know the result to be zero. Applying interval arithmetic we can only conclude

$$\boldsymbol{x} - \boldsymbol{x} = \{x - y \mid x, y \in \boldsymbol{x}\}.$$

The resulting set surely contains zero, but maybe many more numbers too. For instance, given $\boldsymbol{x} = [1, 2]$*, the result of the subtraction* $\boldsymbol{x} - \boldsymbol{x}$ *is an interval of diameter* 2*:*

$$[1, 2] - [1, 2] = [-1, 1].$$

Equally, for the division of a variable by itself (given in formula (1.4)*) one gets*

$$\boldsymbol{x}/\boldsymbol{x} = \{x/y \mid x, y \in \boldsymbol{x}\}.$$

For our example interval $\boldsymbol{x} = [1, 2]$ *we see that this does not result in the point interval* $[1]$ *but in*

$$[1, 2]/[1, 2] = [0.5, 2].$$

1.7.2 Algebraic rules

Due to the dependency problem not all of the algebraic rules for real numbers can be conveyed to intervals. For instance, addition and subtraction, as well as multiplication and division, are no longer inverse functions. Besides, the result of a multiplication of an interval with itself does not have to be nonnegative. For example, for the interval $\boldsymbol{x} = [-1, 1]$ multiplication yields

$$\boldsymbol{x} \cdot \boldsymbol{x} = [-1, 1] \cdot [-1, 1] = [-1, 1]$$

which contains negative numbers. This is the reason for defining the square function

$$\boldsymbol{x}^2 := \begin{cases} [0, |\boldsymbol{x}|^2] & \text{if } 0 \in \boldsymbol{x} \\ [\underline{x}^2, \overline{x}^2] & \text{if } \underline{x} > 0 \\ [\overline{x}^2, \underline{x}^2] & \text{if } \overline{x} < 0 \end{cases} \tag{1.6}$$

which has nonnegative range for all intervals $\boldsymbol{x} \in {}^*\mathbb{IR}$. For vectors this function is applied componentwise. For the example interval $\boldsymbol{x} = [-1, 1]$ we now attain

$$\boldsymbol{x}^2 = [-1, 1]^2 = [0, 1]$$

which is not only nonnegative and a reduction of the overestimation of the enclosure, but the best possible result. Other power operators can be defined in a similar manner to avoid repeated multiplication and the resulting overestimation

due to the dependency problem. For any degree $k \in \mathbb{N}$ the power operator can be given as

$$\boldsymbol{x}^k := \begin{cases} [0, |\boldsymbol{x}|^k] & \text{for } 0 \in \boldsymbol{x} \text{ and even } k \\ [\underline{x}^k, \overline{x}^k] & \text{for } \underline{x} > 0 \text{ and even } k \\ [\overline{x}^k, \underline{x}^k] & \text{for } \overline{x} < 0 \text{ and even } k \\ [\underline{x}^k, \overline{x}^k] & \text{for odd } k. \end{cases} \tag{1.7}$$

Considering arithmetic properties, the commutative law and the associative law are fulfilled for intervals (see for example [Neu90]). Even though, one cannot apply the distributive law in general. Take for example the intervals $\boldsymbol{a} = [-1, 1]$, $\boldsymbol{b} = [-1, 0]$ and $\boldsymbol{c} = [0, 1]$. Since

$$\boldsymbol{a} \cdot (\boldsymbol{b} + \boldsymbol{c}) = [-1, 1] \cdot ([-1, 0] + [0, 1]) = [-1, 1] \cdot [-1, 1] = [-1, 1]$$

is not equal to

$$\boldsymbol{ab} + \boldsymbol{ac} = [-1, 1] \cdot [-1, 0] + [-1, 1] \cdot [0, 1] = [-1, 1] + [-1, 1] = [-2, 2]$$

distributivity cannot hold in general, although it holds for some special cases (see, e.g., [Neu90]). What we can rely on when working with intervals, is a weaker version of distributivity, called *subdistributivity*. This is given by the property

$$\boldsymbol{a} \cdot (\boldsymbol{b} \pm \boldsymbol{c}) \subseteq \boldsymbol{ab} \pm \boldsymbol{ac} \text{ for all intervals } \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in {}^*\mathbb{IR}.$$

1.7.3 Wrapping effect

Another source of overestimation is the so-called *wrapping effect*. Since we are only working with vectors of intervals, the only sets that can be enclosed exactly are intervals and boxes $\boldsymbol{x} \in {}^*\mathbb{IR}^n$. For them $\square(\boldsymbol{x}) = \boldsymbol{x}$ always holds true. For all other sets $P \in \mathcal{P}({}^*\mathbb{R}^n) \setminus {}^*\mathbb{IR}^n$ we have to "wrap" P by a box $\square(P)$. This box necessarily has to overestimate P. Figure 1.3 illustrates the wrapping effect with a two-dimensional example.

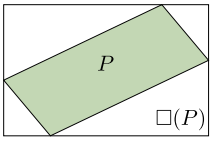


Figure 1.3: Wrapping effect for a two-dimensional set $P \in \mathcal{P}({}^*\mathbb{R}^2)$

However, there exist approaches to diminish the overestimation caused by the wrapping effect. We can subdivide sets into smaller parts and enclose the evolving parts with boxes. Other approaches that will not be followed here are variable transformations or the usage of other polytopes or ellipsoids to enclose sets. References to these approaches can, e.g., be found in [Rum10].

1.7.4 Cluster effect

A further interval-related problem occurring when solving systems is the cluster effect. Its name is derived from the effect it causes: a cluster of boxes which contain the same solution or for which the existence of a solution cannot be precluded.

The first situation—multiple boxes containing the same solution—can occur if a box is subdivided such that a solution lies on the common boundary of two emerging subboxes (we subdivide "at a solution"). This can happen in one or more dimensions. Even if working only with boxes in ${}^*\mathbb{IR}^n$ that are disjoint up to their boundaries, we may thus end up with 2^n boxes containing the same solution on their common boundaries.

A similar problem occurs when it is not possible to determine in which of the boxes evolving from subdivision a solution is seated. This can happen due to overestimation of function enclosures. In this case, we cannot discard boxes that do not contain a solution. The result is nearly the same: a cluster of boxes containing only one solution. The only difference is that not all boxes really contain the solution.

Both situations are referred to as *cluster effect*. Most strategies to cope with this effect try to prevent the formation of clusters. Some possibilities to avoid the formation of clusters are proposed when handling subdivision in section 2.2.

For clusters that already have evolved in a given algorithm, we propose a different approach. One could check for clusters and try to get rid of them. This can, for example, be achieved by enclosing a cluster into one or more new boxes and proceeding to work with these boxes in the algorithm. One simplified possible approach is sketched in Figure 1.4. The dot in these figures represents a solution. The union of the subboxes shown in the first part of the figure is enclosed by a superbox. In the second part, this superbox is subdivided into another set of boxes. In the third part of the figure, we assume to have precluded the existence of solutions in the outer subboxes. We can thus discard these boxes.

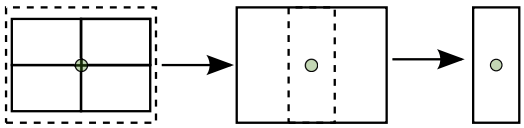


Figure 1.4: An approach to reduce the cluster effect

This handling is not included in SONIC until now, since it is assumed to influence the overall computation time negatively. Furthermore, the enclosure by a superbox may itself lead to multiple boxes covering the same solution. Approaches to reduce the cluster effect after the computation of the solution boxes is completed are discussed in section 4.3.5)

1.8 Function evaluation and enclosures

Until now we only discussed how to compute function enclosures for those basic functions for which we can compute the exact range by the endpoints of the interval arguments. For general functions another approach is needed. For them, it is usually not possible to compute a function enclosure by determining (and enclosing) the range, because we simply cannot compute the range of the function. However, interval analysis provides us with different methods for calculating enclosures that do not rely on knowledge of the function range. In this section some of these methods suited for general functions will be presented and pointers concerning the sharpness of the evolving function enclosures will be given.

Before presenting the techniques for calculations, we want to give the definition of a well-known measure for the quality of enclosures. (More possibilities to assess and compare the quality of function enclosures can, e.g., be found in [Neu90].)

Definition 1.7 (Order of function enclosures)
Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and $\boldsymbol{f} : {}^*\mathbb{IR}^n \to {}^*\mathbb{IR}$ an enclosure for f. Furthermore, let $\boldsymbol{x}$ be an arbitrary box in ${}^*\mathbb{IR}^n$ and $\mathrm{range}_f(\boldsymbol{x})$ the exact range over this box.

If there exist a constant number $\rho \in \mathbb{R}$, independent of $\boldsymbol{x}$, and a fixed positive integer $\alpha \in \mathbb{N}$, such that the inequality

$$||\mathrm{width}\,(\boldsymbol{f}(\boldsymbol{x}))||_\infty - ||\mathrm{width}\,(\square\mathrm{range}_f(\boldsymbol{x}))||_\infty \leq \rho \cdot ||\mathrm{width}\,(\boldsymbol{x})||_\infty^\alpha$$

holds for every $\boldsymbol{x}$ with sufficiently small $\mathrm{width}\,(\boldsymbol{x})$*, then $\boldsymbol{f}$ is called an* enclosure of order α *for the function f.*

Of course, this definition is useful only when considering bounded intervals. Which is satisfactory since we are just interested in the asymptotic behavior for small intervals. Note furthermore that, although it is true that higher order enclosures are more accurate for rather small boxes, it would be wrong to claim that they always provide better results. For larger boxes enclosures of lower order may still be favorable.

As said before, interval analysis provides us with different options to compute enclosures for the range of a function over some interval or interval vector. Be aware that due to overestimation these often yield an interval larger than the true range. Since function enclosures are not only important for single computations but have a huge influence on the performance of complex algorithms and solvers, a main goal in the field of interval analysis is to provide enclosures as small as possible. Besides the usage of a single promising function enclosure, we have some further options to reduce overestimation.

As a first option, we could compute function enclosures by different methods and intersect the results. Another way to sharpen function enclosures is to reformulate expressions (cf. section 1.8.7). A third possibility to reduce overestimation

is to compute a function enclosure over a box $\boldsymbol{x}$ with the help of a covering or subdivision $\boldsymbol{x} = \boldsymbol{x}^{(1)} \cup \ldots \cup \boldsymbol{x}^{(k)}$. The latter option benefits from the fact that overestimation usually grows with the size of the considered boxes. This property can be proved if a function is *inclusion monotonic*, which means nothing else than

$$\boldsymbol{f}(\boldsymbol{x}_{\text{sub}}) \subseteq \boldsymbol{f}(\boldsymbol{x}) \text{ for all } \boldsymbol{x}_{\text{sub}} \subseteq \boldsymbol{x}.$$

Then for the function enclosure over the covering (or subdivision) we know that

$$\bigcup_{i=1}^{k} \boldsymbol{f}(\boldsymbol{x}^{(i)}) \subseteq \boldsymbol{f}(\boldsymbol{x})$$

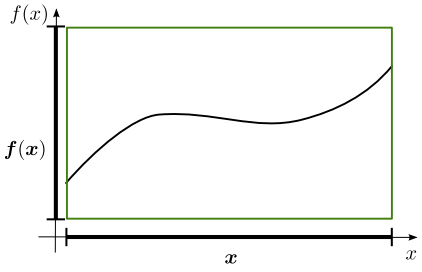
holds. But even if we do not consider an inclusion monotonic function enclosure, for subdivisions and coverings of $\boldsymbol{x}$ the inclusion

$$\text{range}_f(\boldsymbol{x}) = \bigcup_{i=1}^{k} \text{range}_f(\boldsymbol{x}^{(i)}) \subseteq \bigcup_{i=1}^{k} \boldsymbol{f}(\boldsymbol{x}^{(i)}) \tag{1.8}$$

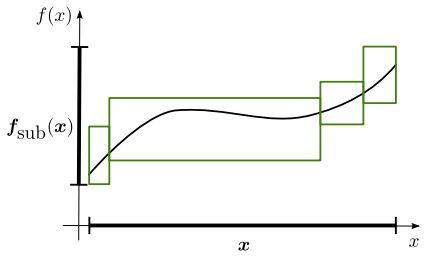
holds. We can thus compute the function enclosures for all parts $\boldsymbol{x}^{(i)}$ ($i \in \{1, \ldots, k\}$) and join them to attain a function enclosure $\boldsymbol{f}_{\text{sub}}$ over the original box

$$\boldsymbol{f}_{\text{sub}}(\boldsymbol{x}) := \boldsymbol{f}(\boldsymbol{x}^{(1)}) \cup \ldots \cup \boldsymbol{f}(\boldsymbol{x}^{(k)}).$$

The capacity of this method to reduce overestimation is illustrated by Figure 1.5. Note that in practice one always needs to find a trade-off between accuracy and the computation time needed to evaluate the enclosures on the subboxes. In most cases this has to be done using heuristics.



(a) Function enclosure over an interval



(b) Function enclosure on subdivision of the interval

Figure 1.5: Function enclosures

In the following, we briefly present some well-known methods to compute function enclosures.

1.8.1 Natural function enclosure

The *natural* or *naive function enclosure* is a very well-known and simple function enclosure. For attaining it, one only has to apply interval arithmetic for evaluating the function over a box. For that purpose, all variables in the function are replaced with intervals and the arithmetic operations and standard functions are interpreted as their interval counterparts.

Example 1.2
We evaluate the natural function enclosure $\boldsymbol{f}_N$ *for the function* f *defined by* $f(x) = x^2 - 2x + 1$ *on the interval* $\boldsymbol{x} = [0, 2]$.

$$\begin{aligned}\boldsymbol{f}_N(\boldsymbol{x}) &= \boldsymbol{x}^2 - 2\boldsymbol{x} + [1] = [0,2]^2 - 2 \cdot [0,2] + [1] \\ &= [0,4] - [0,4] + [1] = [-4,4] + [1] \\ &= [-3,5]\end{aligned}$$

The same function can be defined by $f(x) = (x-1)^2$. *Using this formulation we elude the dependency problem and get*

$$\boldsymbol{f}_N(\boldsymbol{x}) = (\boldsymbol{x} - 1)^2 = ([0,2] - [1])^2 = [-1,1]^2 = [0,1]$$

which is the exact range of f *over the interval* $\boldsymbol{x} = [0, 2]$.

The natural function enclosure is a function enclosure of order one and inclusion monotonic [RR84]. One advantage of the natural function enclosure is that it only causes little computational effort. Moreover, if every variable occurs only once in a function, the enclosure is exact. This still holds true if variables with multiple occurrences are enclosed by point intervals. (Powers x^k in this case have to be considered as multiple occurrences of $\boldsymbol{x}$, or we have to enclose them exactly as done by formula (1.7). Note again that point intervals may be enclosed by proper intervals in the set of machine-representable numbers, see section 1.9.)

When we have to deal with multiple occurrences of variables on the other hand, the natural function enclosure can cause vast overestimation. In this case usually one of the following more sophisticated methods provides a sharper enclosure.

1.8.2 Centered forms

If we consider a continuously differentiable function on a given box, we may apply the *centered function evaluation* for which we employ the mean value theorem.

Theorem 1.1 (Mean value theorem)
For a convex and open set $D \subseteq \mathbb{R}^n$, *two vectors* $x \in D$ *and* $c \in D$ *and a function* $f : \mathbb{R}^n \to \mathbb{R}$ *that is continuously differentiable on* D, *there exists a value* $\xi = c + \theta(x - c)$ *with* $\theta \in [0, 1]$ *such that*

$$f(x) = f(c) + \nabla f(\xi)(x - c).$$

Given this formulation, a continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, substituting $D = \boldsymbol{x} \in {}^*\mathbb{IR}^n$ and utilizing the definition of a function enclosure (Def. 1.4) as well as a point $c \in \boldsymbol{x}$ and an enclosure $[\nabla f]$ of the gradient ∇f we get

$$\boldsymbol{f}_C(\boldsymbol{x}) := f(c) + [\nabla f](\boldsymbol{x})(\boldsymbol{x} - c) \tag{1.9}$$

as an alternative function enclosure for f. The interval function $\boldsymbol{f}_C$ is called a *centered form* and is at least of quadratic order [RR84].

The vector c in this formula is called the *center* of the enclosure. Its selection can have enormous impact on the quality of the achieved function enclosure. If we consider the special case of evaluating at the midpoint, we set the center to the midpoint ($c = \operatorname{mid}(\boldsymbol{x})$) and get the so-called *mean value form*. This special centered form is illustrated by Figure 1.6.

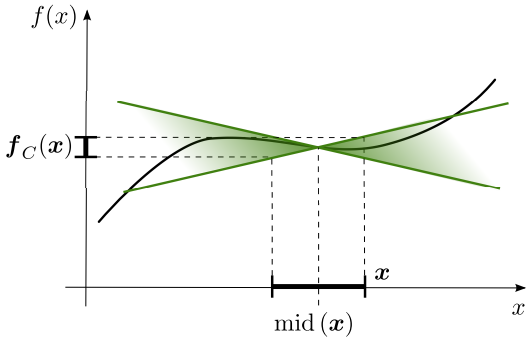


Figure 1.6: Function enclosure by the mean value

1.8.2.1 The center

We illustrate the influence of the choice for the center by an example.

Example 1.3
As in Example 1.2, we evaluate the centered function enclosure for the function defined by $f(x) = x^2 - 2x + 1$ *on the interval* $\boldsymbol{x} = [0, 2]$. *The center is chosen as the midpoint* $c = 1$. *The first derivative is computed with the help of the natural function enclosure as*

$$[\nabla f](\boldsymbol{x}) = 2\boldsymbol{x} - 2 = [2] \cdot [0, 2] - [2] = [0, 4] - [2] = [-2, 2].$$

Inserting the above result into formula (1.9) *we get*

$$\boldsymbol{f}_C(\boldsymbol{x}) = f(1) + [-2, 2]([0, 2] - [1]) = 0 + [-2, 2][-1, 1] = [-2, 2]$$

which is a sharper enclosure than the naive one, which only gave us $[-3, 5]$.

For showing the influence of the center, we repeat our computations for $c = 1/2$*. Since the first derivative stays the same this results in*

$$\begin{aligned}\boldsymbol{f}_C(\boldsymbol{x}) &= f\left(\frac{1}{2}\right) + [-2,2]\left([0,2] - \left[\frac{1}{2}\right]\right) = \left(\frac{1}{4} - 1 + 1\right) + [-2,2]\left[-\frac{1}{2}, \frac{3}{2}\right] \\ &= \frac{1}{4} + [-3,3].\end{aligned}$$

We can see clearly that in this case the first choice for the center delivers the sharper enclosure over the given interval.

A method for choosing centers sensibly was given by Baumann (see [Neu90, p. 58], [Bau88]). It tells us that choosing c as the midpoint of the given interval (or box) results in a width-optimal result. Further it states how to determine a center c_1 to attain the largest possible lower bound for the centered form and how to choose a center c_2 providing a smallest upper bound. One can now compute two function enclosures, one with center c_1, one with c_2. Then both function enclosures are intersected. For implementations, this procedure has the disadvantage of needing to compute two enclosures—resulting in higher computational efforts (although the derivatives only need to be computed once)—but it provides us with a sharper function enclosure.

1.8.3 Taylor forms

The centered form can also be seen as being derived from a Taylor expansion of order one. If a function can be differentiated more than once, one can apply higher order Taylor expansions as well to attain function enclosures. This is what is done for so-called *Taylor forms*. In principle, Taylor expansions are used to bound nonlinear interval functions by interval valued polynomials. The transfer to a function enclosure then is a simple adaptation of Definition 1.4. A Taylor form is called *Taylor form of order* k, depending on the order k of the Taylor expansion used to deduce it. (This is *not* the same as the order of convergence of the function enclosure.)

As an example, the enclosure based on the Taylor expansion of second order is given. For a twice continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, a box $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ and $c \in \boldsymbol{x}$ we get the enclosure

$$\boldsymbol{f}_T(\boldsymbol{x}) := f(c) + \nabla f(c)(\boldsymbol{x} - c) + \frac{1}{2}(\boldsymbol{x} - c)^T [D^2 f](\boldsymbol{x})(\boldsymbol{x} - c).$$

Example 1.4
As before, we consider the function f *defined by* $f(x) = x^2 - 2x + 1$*, the interval* $\boldsymbol{x} = [0,2]$ *with a center* $c = 1$*. The first derivative is again computed with the help of the natural function enclosure*

$$[\nabla f](c) = 2c - 2 = [2] \cdot [1] - [2] = [2] - [2] = [0].$$

The second derivative is simply enclosed by the point interval [2]*. We attain the second order Taylor form as*

$$\begin{aligned}\boldsymbol{f}_T(\boldsymbol{x}) &= f(1) + [0] \cdot ([0,2] - [1]) + \frac{1}{2}([0,2] - [1])[2]([0,2] - [1]) \\ &= 0 + [0][-1,1] + \frac{1}{2}[-1,1][2][-1,1] = [0] + \frac{1}{2}[-2,2] = [-1,1].\end{aligned}$$

Thus, for our example, the result for the Taylor form provides a sharper enclosure than that of the centered function enclosure.

For enclosures using Taylor forms of nth order, one has to consider the necessity of computing all derivatives up to nth order. Thus Taylor forms may require high computational efforts. On the other hand, one obtains function enclosures of at least quadratic order for all Taylor forms of at least second order [RR84]. (Note that in general it is difficult to attain a convergence order larger than 2 for Taylor forms of arbitrary functions [Kea96b].) A further thorough discussion of Taylor forms can be found in [Neu02].

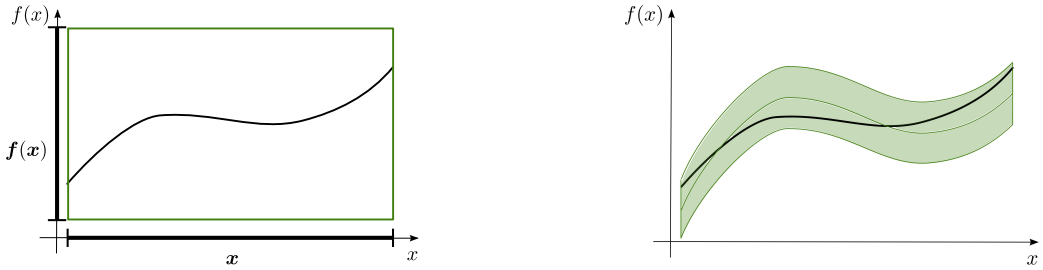
Kienitz tested the application of higher order Taylor forms in his diploma thesis [Kie03]. His observation is that, in general, they deliver sharper enclosures only for small boxes but require a lot more computation time. For this reason, only first- and second-order Taylor enclosures are used in SONIC.

1.8.4 Taylor models

As well as Taylor forms, *Taylor models* are derived from Taylor expansions. For a Taylor model all coefficients of the Taylor expansion except the constant term are computed as real numbers. The constant term is replaced by a *remainder interval* accumulating all computation errors that could be introduced by computing only with real coefficients instead of intervals. The simplest way to compute a function enclosure of Taylor model is to compute a function enclosure for the polynomial term and to add the remainder interval. For even sharper enclosures one can also use techniques such as *linear dominated bounder* [Mak98].

The fundamental difference between function enclosures relying on boxes and Taylor models is illustrated by Figure 1.7. Of course, Taylor models can also be used to compute function enclosures on subboxes.

Thorough definitions and studies can, e.g., be found in the work of the group around Berz and Makino (see for example [BM09] and [Ber]). They further implemented Taylor models in their software COSY Infinity [Cos] and consider so-called inverse Taylor models [Hoe01] [BH01] [HB02].



(a) Enclosure with a box

(b) Enclosure by a Taylor model

Figure 1.7: Function enclosures

1.8.5 Derivatives and slopes

As can be seen from the previous formulae, we need to compute enclosures not only for functions, but also for their derivatives. If a formula for the derivative is known or can be computed its function enclosure can be computed by the same methods as other functions. The natural function enclosure can be used for every function. For a twice differentiable function the first-order derivative can also be computed using the mean value form. For functions that are differentiable more than twice, we could even compute higher order derivatives and enclose them by applying Taylor forms of higher orders. (For a k times continuously differentiable function, the order of the derivative plus the order of the Taylor form applied to the derivative has to be smaller or equal k.)

Additionally, in the centered and Taylor form there is no need to use the derivative. One can also use a so-called slope.

Definition 1.8 (Slope)
For a function $f : \mathbb{R}^n \to \mathbb{R}$ an interval vector $\boldsymbol{s} \in {}^*\mathbb{IR}^n$ is called the slope *over a box $\boldsymbol{x}$ with center $c \in \boldsymbol{x}$ if for all $x \in \boldsymbol{x}$ there exists a vector $s \in \boldsymbol{s}$ such that*

$$f(x) - f(c) = s^T(x - c).$$

For multidimensional functions we can define slope matrices in the same way. Methods for computing slopes can, e.g., be found in [Kea96b].

A method to attain sharper enclosures of derivatives for multivariate functions is the *progressive derivation*. The ansatz dates back to Hansen and is also related to as "Hansen's slope technique" [Kea96b, p. 30] [Han92, p. 51] [JKDW01]. It computes derivatives by considering changes in the variables componentwise, not simultaneously.

In the following chapters we will further speak of the influence a variable has on a function and its enclosure. This influence is generally estimated by an enclosure of the Jacobian matrix $\boldsymbol{Df}$, since $\boldsymbol{Df}_{i,j}$ gives us a measure for the rate of change of function f_i with respect to a variable x_j within a given box.

1.8.6 Contractors

Definition 1.9 (Contractor)
For a function $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$ with solution set $\mathcal{S} = \{x \in D \mid f(x) = 0\}$, a function $\mu : {}^*\mathbb{IR}^n \to {}^*\mathbb{IR}^n \cup \{\emptyset\}$ is said to be a contractor *or* contraction method *of the system $f(x) = 0$ if it fulfills*

$$\mu(\boldsymbol{x}) \subseteq \boldsymbol{x} \quad \text{for all } \boldsymbol{x} \subseteq D$$

and

$$\mathcal{S} \cap \boldsymbol{x} = \mathcal{S} \cap \mu(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \subseteq D.$$

By using contractors we can shrink boxes while still covering the solution set $\mathcal{S}$. In the special case $\mu(\boldsymbol{x}) = \emptyset$ we even get the information that the box $\boldsymbol{x}$ contains no solution of the system. In this case we can discard the box completely from the branch-and-bound algorithm.

Since contractors can reduce the box number in the solution of interval problems considerably and thus accelerate the solving process, they are sometimes also referred to as "acceleration techniques".

Definition 1.10 (Contraction of a box)
For a function $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$ with solution set $\mathcal{S} = \{x \in D \mid f(x) = 0\}$ and a contractor $\mu : {}^*\mathbb{IR}^n \to {}^*\mathbb{IR}^n \cup \{\emptyset\}$, the box $\boldsymbol{x}$ is said to be contracted *by μ if it fulfills the relation*

$$\mu(\boldsymbol{x}) \subsetneq \boldsymbol{x}.$$

If μ is a contractor on $\boldsymbol{x}$ and

$$\text{width}\,(\boldsymbol{x})/\text{width}\,(\mu(\boldsymbol{x})) > \kappa$$

holds for some predefined value $\kappa \in \mathbb{R}$ with $\kappa > 1$ we say that the contractor μ is successful *or* successful enough *on $\boldsymbol{x}$. Accordingly a contractor is called successful if it is successful for all boxes $\boldsymbol{x}$ in a given domain. For boxes $\boldsymbol{x}$ the volume is considered instead of the width.*

Note the subtle difference of the demand for a proper subset in this definition.

1.8.6.1 Direct function evaluation

A simple and cheap contractor is the direct function evaluation given by formula (1.10).

$$\mu(\boldsymbol{x}) := \begin{cases} \emptyset & \text{for } 0 \notin \boldsymbol{f}(\boldsymbol{x}) \\ \boldsymbol{x} & \text{else} \end{cases} \tag{1.10}$$

For this contractor we consider an arbitrary function enclosure $\boldsymbol{f}$, compute a function enclosure $\boldsymbol{f}(\boldsymbol{x})$ and check whether it contains zero. If $0 \notin \boldsymbol{f}(\boldsymbol{x})$, the box $\boldsymbol{x}$ cannot contain any solution of the system ($\boldsymbol{x} \cap \mathcal{S} = \emptyset$) and can be discarded. If $0 \in \boldsymbol{f}(\boldsymbol{x})$, box $\boldsymbol{x}$ may or may not contain a solution and $\boldsymbol{x}$ is not changed by the contractor.

Note that the direct function evaluation cannot discard all boxes that do not contain a solution. It cannot discard a box when the function enclosure contains zero due to pure overestimation. But even if we can compute exact ranges instead of function enclosures, some boxes without solutions will withstand the contractor. This happens if the output of function $f : \mathbb{R}^n \to \mathbb{R}^m$ has more than one dimension ($m > 1$) and the ranges of function components of f in the dimensions i and j (in $\{1, \ldots, m\}$) both contain zero, but the roots do not coincide. In other words: f_i may be zero for a root $x^{(1)} \in \boldsymbol{x}$, but f_j is zero just for $x^{(2)} \in \boldsymbol{x}$ with $x^{(1)} \neq x^{(2)}$ (and $f_i(x^{(2)}) \neq 0$, $f_j(x^{(1)}) \neq 0$).

1.8.6.2 Linear constraints

For constructing further contractors, we first search all solutions for linear constraints such as

$$0 = \sum_{j=1}^{n} x_j \cdot y_j$$

for $x \in \boldsymbol{x}$, $y \in \boldsymbol{y}$. For any solution we can transform this constraint into the equation

$$x_i \cdot y_i = -\sum_{\substack{j=1 \\ j \neq i}}^{n} x_j \cdot y_j$$

and use enclosures of the variables and relational division to attain

$$x_i \in \left(-\sum_{\substack{j=1 \\ j \neq i}}^{n} \boldsymbol{x}_j \cdot \boldsymbol{y}_j \right) \oslash \boldsymbol{y}_i. \tag{1.11}$$

A contractor for $\boldsymbol{x}$ can thus be defined by

$$\mu(\boldsymbol{x}_i) := \boldsymbol{x}_i \cap \left(\left(-\sum_{\substack{j=1 \\ j \neq i}}^{n} \boldsymbol{x}_j \cdot \boldsymbol{y}_j \right) \oslash \boldsymbol{y}_i \right) \quad \text{for } i = 1, \ldots, n.$$

If we are interested in contracting $\boldsymbol{y}$, this can be done in the same way.

1.8.6.3 Linear systems

We further consider linear systems. A linear system of equations including interval values can be denoted as

$$\boldsymbol{A} \cdot \boldsymbol{x} = \boldsymbol{b}$$

with $\boldsymbol{A} \in {}^*\mathbb{IR}^{m \times n}$, $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ and $\boldsymbol{b} \in {}^*\mathbb{IR}^m$. We thus search for the solution set

$$\{x \in \boldsymbol{x} \mid Ax = b \text{ for some } A \in \boldsymbol{A} \text{ and some } b \in \boldsymbol{b}\}.$$

For this purpose, we can apply an interval version of the Gauß-Seidel method. In theory, we could resolve any constraint for any variable. However, we mostly assume to deal with square systems with A near to the identity matrix. Often this property can be attained by preconditioning.
Then for the *interval Gauß-Seidel method* the ith constraint is resolved as in equation (1.11) to contract the ith component of a given box $\boldsymbol{x}$. For non-square systems one can clearly contract the first $\min\{m, n\}$ variables in the same manner.

We make use of the most current values of all other components of $\boldsymbol{x}$. (Without this update of the variables we would end up with a Jacobi method.) In the following, we simply speak of the *Gauß-Seidel method* as it is clear that we have to apply the interval version for interval valued systems.

One step of the Gauß-Seidel method can be described by

$$\boldsymbol{x}_i' := \boldsymbol{x}_i \cap \left(\left(\boldsymbol{b}_i - \sum_{j=1}^{i-1} \boldsymbol{a}_{i,j}(\boldsymbol{x}_j') - \sum_{j=i+1}^{n} \boldsymbol{a}_{i,j}(\boldsymbol{x}_j) \right) \oslash \boldsymbol{a}_{i,i} \right)$$

for all components $i = 1, \ldots, \min\{m, n\}$. If we consider an underdetermined system ($m < n$), the remaining components stay unchanged ($\boldsymbol{x}_i' := \boldsymbol{x}_i$ for $i = m+1, \ldots, n$). To enhance the Gauß-Seidel method, these could also be contracted by suitable components.

Applying the same reasoning for an interval system in the later needed form

$$\boldsymbol{A} \cdot (\boldsymbol{x} - \widetilde{x}) = \boldsymbol{b}$$

with $\boldsymbol{A} \in {}^*\mathbb{IR}^{m \times n}$, $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ and $\boldsymbol{b} \in {}^*\mathbb{IR}^m$ we get a contractor

$$\begin{gathered} \mathcal{GS}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}, \widetilde{x}) : {}^*\mathbb{IR}^n \to {}^*\mathbb{IR}^n \\ \boldsymbol{x} \mapsto \boldsymbol{x}' \end{gathered}$$

by computing

$$\boldsymbol{x}_i' := \boldsymbol{x}_i \cap \left(\widetilde{x}_i + \left(\left(\boldsymbol{b}_i - \sum_{j=1}^{i-1} \boldsymbol{a}_{i,j}(\boldsymbol{x}_j' - \widetilde{x}_j) - \sum_{j=i+1}^{n} \boldsymbol{a}_{i,j}(\boldsymbol{x}_j - \widetilde{x}_j) \right) \oslash \boldsymbol{a}_{i,i} \right) \right)$$

Algorithm 1.2 INTERVAL GAUSS-SEIDEL METHOD $\mathcal{GS}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}, \widetilde{x})$

for $i = 1, \ldots, \min\{m, n\}$ **do**
  $\boldsymbol{x}'_i = \widetilde{x}_i + \square \left(\left(\boldsymbol{b}_i - \sum_{j=1}^{i-1} \boldsymbol{a}_{i,j}(\boldsymbol{x}'_j - \widetilde{x}_j) - \sum_{j=i+1}^{n} \boldsymbol{a}_{i,j}(\boldsymbol{x}_j - \widetilde{x}_j) \right) \oslash \boldsymbol{a}_{i,i} \right)$
  if $\boldsymbol{x}'_i \cap \boldsymbol{x}_i = \emptyset$ **then**
    exit {$\boldsymbol{x}$ cannot contain a solution}
  else
    $\boldsymbol{x}'_i = \boldsymbol{x}'_i \cap \boldsymbol{x}_i$
  end if
end for

for all components $i = 1, \ldots, \min\{m, n\}$. The algorithm for this formulation of the interval Gauß-Seidel method is given in Algorithm 1.2.

The Gauß-Seidel method becomes a promising contractor in connection with preconditioning. In this case we solve the preconditioned system

$$(C\boldsymbol{A}) \cdot (\boldsymbol{x} - \widetilde{x}) = C\boldsymbol{b} \tag{1.12}$$

with a real-valued preconditioning matrix $C \in \mathbb{R}^{m \times m}$.

Thus we attain the *preconditioned Gauß-Seidel method* as denoted in Algorithm 1.3. In our notation, C_k stands for the kth row of a preconditioner, so

$$C_k = (c_{k,1}, \ldots, c_{k,m})$$

and $\boldsymbol{A}_j$ is the jth column of the interval matrix $\boldsymbol{A} \in {}^*\mathbb{IR}^{m \times n}$

$$\boldsymbol{A}_j = (\boldsymbol{a}_{1,j}, \ldots, \boldsymbol{a}_{m,j})^T .$$

Algorithm 1.3 PRECONDITIONED INTERVAL GAUSS-SEIDEL METHOD $\mathcal{PGS}$ $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{x}, \widetilde{x})$

for $k = 1, \ldots, \min\{m, n\}$ **do**
  compute preconditioner row C_k
  $\boldsymbol{x}'_k = \widetilde{x}_k + \square \left\{ \left(C_k\boldsymbol{b} - \sum_{\substack{j=1 \\ j \neq k}}^{n} (C_k\boldsymbol{A}_j)(\boldsymbol{x}_j - \widetilde{x}_j) \right) \oslash (C_k\boldsymbol{A}_k) \right\}$
  if $\boldsymbol{x}'_k \cap \boldsymbol{x}_k = \emptyset$ **then**
    exit {$\boldsymbol{x}$ cannot contain a solution of system (1.12) }
  else
    $\boldsymbol{x}_k = \boldsymbol{x}'_k \cap \boldsymbol{x}_k$
  end if
end for

Preconditioners are discussed in detail in section 2.3.5. However, we want to say in advance that preconditioning preserves the property of the Gauß-Seidel method of being a contractor.

Note that there exist further methods to solve interval linear systems, as the Krawzcyk method or Gaussian elimination. The discussion of contractors for nonlinear constraints is postponed to Chapter 2.

1.8.7 Expression optimization

Many functions can be defined by different expressions. As we have already seen in Example 1.2, different expressions may lead to different function enclosures.

To avoid overestimation due to the dependency problem we have the option to rearrange the expression defining a function. For this purpose, expressions are modified and terms simplified before any variables are replaced by interval values and the computation is started. Naturally, this has to be done without changing the represented functions. Generally, due to the dependency problem, fewer occurrences of a variable in an expression have to be preferred. Likewise, it is often helpful to use fewer operations.

We can formulate some rules for simplifying terms with the aim of reducing overestimation. Examples for simplification rules are straightforward conversions applying algebraic rules. These are rules like $0 + \boldsymbol{x} = \boldsymbol{x}$, $-(-\boldsymbol{x}) = \boldsymbol{x}$, $\boldsymbol{x} - \boldsymbol{x} = [0]$, $\exp(\log(\boldsymbol{x})) = \boldsymbol{x}$, $\boldsymbol{x} + \boldsymbol{x} = 2\boldsymbol{x}$ and $\boldsymbol{x} \cdot \boldsymbol{x} = \boldsymbol{x}^2$ using the square function given in equation (1.6).

Note that expression optimization has to be applied to algebraic expressions and cannot be used after replacing variables with intervals. Furthermore, the usage of predefined rules for reformulating expressions can only be fruitful if unfavorable constellations are easy to spot. For more complicated expressions, it often is impossible to minimize the number of occurrences of variables *and* the number of operations at the same time. In this case one often does not know for sure, which formulation of a term provides the sharper enclosure. Even more, which enclosure is sharper may also depend on the considered interval (see Example 1.5).

For getting a small enclosure for the function range, it may help to compute enclosures for different expressions defining the function and to intersect them. Of course, this results in higher computational costs due to the necessity to calculate more than one function enclosure.

Example 1.5
We consider the function f given by the expression $x - x^2$ and calculate the natural function enclosure on two intervals $\boldsymbol{x}'$ and $\boldsymbol{x}''$.

$$\begin{aligned}
\boldsymbol{x}' = [0,1]: \quad & \boldsymbol{x}' - \boldsymbol{x}'^2 = [0,1] - [0,1]^2 \quad = [0,1] - [0,1] \;\; = [-1,1] \\
\boldsymbol{x}'' = [-1,1]: \quad & \boldsymbol{x}'' - \boldsymbol{x}''^2 = [-1,1] - [-1,1]^2 = [-1,1] - [0,1] = [-2,1]
\end{aligned}$$

The same function can be defined using the expression $x \cdot (1 - x)$. This results in other enclosures.

$$\begin{aligned}
\boldsymbol{x}' = [0,1]: \quad & \boldsymbol{x}' \cdot ([1] - \boldsymbol{x}') = [0,1] \cdot ([1] - [0,1]) \quad = [0,1] \cdot [0,1] = [0,1] \\
\boldsymbol{x}'' = [-1,1]: \quad & \boldsymbol{x}'' \cdot ([1] - \boldsymbol{x}'') = [-1,1] \cdot ([1] - [-1,1]) = [-1,1] \cdot [0,2] = [-2,2]
\end{aligned}$$

We see that for $\boldsymbol{x}'$ enclosing the second expression results in the sharper enclosure while for $\boldsymbol{x}''$ the first expression does. Hence we cannot decide which of the two given formulations should be preferred in general.

Actually the expression $\frac{1}{4} - (\frac{1}{2} - x)^2$ can get us the exact range of function f, since the function is continuous and variable x only occurs once.

$$\begin{aligned}
\boldsymbol{x}' = [0,1]: \quad & \frac{1}{4} - \left(\frac{1}{2} - \boldsymbol{x}'\right)^2 = \frac{1}{4} - \left(\frac{1}{2} - [0,1]\right)^2 = \frac{1}{4} - \left(\left[-\frac{1}{2}, \frac{1}{2}\right]\right)^2 \\
& = \frac{1}{4} - \left[0, \frac{1}{4}\right] = \left[0, \frac{1}{4}\right] \\
\boldsymbol{x}'' = [-1,1]: \quad & \frac{1}{4} - \left(\frac{1}{2} - \boldsymbol{x}''\right)^2 = \frac{1}{4} - \left(\frac{1}{2} - [-1,1]\right)^2 = \frac{1}{4} - \left(\left[-\frac{1}{2}, \frac{3}{2}\right]\right)^2 \\
& = \frac{1}{4} - \left[0, \frac{9}{4}\right] = \left[-2, \frac{1}{4}\right]
\end{aligned}$$

Comparing the results of the two expressions, the overestimating effect caused by the dependency problem becomes apparent again.

The set of rules for expression optimizations used in SONIC can be found in the Rules-File. A big advantage of these rules is that we do not have to burden the user with formulating the problem in an optimal way. Even more important, all formulae arising in the computation, like derivatives, are simplified automatically. Thus we can, e.g., reduce overestimation of function enclosures working relying on enclosures of the derivatives.

◇

In later chapters we will use the following definitions for terms and operators. Herein we restrict our considerations to unary ($\Phi = \Phi(\tau)$) and binary operators ($\Phi = \Phi(\tau_1, \tau_2)$).

Definition 1.11
An operator or function Φ is called elementary *if it depends only on one or two operands. Thus it is either an unary or binary operator. A constraint is called* elementary *if it comprises only one elementary operator.*

Also in SONIC we only work with unary and binary operators.

Definition 1.12 (Term)
Let V_τ be a set of variables and C_τ a set of constants. Let Φ define an elementary operator on these variables.

Then the set of terms *over V_τ is defined as the minimal set such that*

- *every element of V_τ and C_τ is a term,*
- *if τ_1 is a term and Φ is an unary operator then also (τ_1) and $\Phi(\tau_1)$ are terms and*
- *if τ_1 and τ_2 are terms and Φ is a binary operator then also $\Phi(\tau_1, \tau_2)$ is a term.*

For the expressions occurring in our problems, V_τ is the set of variables, C_τ the set of constants of the system.

1.9 Interval arithmetic and computers

As already addressed in short in the introduction, we cannot represent arbitrary real numbers when using computers. This is a fundamental issue, arising from the inevitably limited storage space of computers. The internal representation of numbers in the storage is given by a predefined number of significant digits. Consequently, computers can only work with a finite subset of the infinite set of real numbers. This subset $\mathbb{R}_M$ of so-called *machine numbers* can never contain any numbers requiring an infinite number of digits. Neither representable on a computer are almost all finite real numbers (thus $\mathbb{R}_M \subsetneq \mathbb{R}$). Which finite numbers *are* included in $\mathbb{R}_M$ depends on the used computer number format.

Furthermore, computers store numbers in binary representation. Common formats are presented in every introduction to computer science. Mostly, floating point numbers are used, representing numbers by significant digits and exponents. As an important ramification, all numbers that cannot be represented exactly have to be rounded to machine numbers. We denote the rounding to the nearest machine number by

$$\mathrm{fl} : \mathbb{R} \to \mathbb{R}_M.$$

If we assume to use standard double precision numbers, usually about 16 significant digits can be stored. It is easy to see that we cannot store any number with, e.g., 100 nonzero digits exactly by this kind of machine number.

Example 1.6
We again consider the example 0.1*. Its binary representation is given by*

$$\begin{aligned} 0.1_{10} &= 0.0001100110011001100110011001100110011..._2 \\ &= 0.0\overline{0011}_2 \end{aligned}$$

and has an infinite number of significant digits. Because only a finite number of the significant digits is storable, 0.1 *cannot be represented exactly on a computer—independent of the number of stored digits.*

If we cannot store all initially given numbers accurately, we also cannot assume computations on the rounded representatives to yield exact results. At best they yield approximations. In the worst case, or for difficult computations, these approximations may deviate severely from the true result. This is due to the tendency of (ever so minor) errors to spread and grow during computations.

However, even if we were able to represent all initial values, errors could still be introduced within the computation because intermediate results cannot be represented and have to be rounded to machine numbers. For example, if we use numbers strongly differing in magnitude such as in the formula

$$(2^{30} + 1) - 2^{30},$$

we get the wrong result zero, whenever $2^{30} \in \mathbb{R}_M$ but the intermediate result $2^{30} + 1$ is no element of the machine numbers $\mathbb{R}_M$ and has to be rounded to 2^{30}. We thus have to deal with inevitable rounding and truncation errors whenever using computers. To give another formulation by Hoefkens [Hoe01, Introduction]:

> Most of these errors are caused by the transition from the perfect world of numerical analysis to the limited world of finite state machines.

Interval analysis can alleviate this problem and provide guaranteed enclosures of the precise results. (However, this enclosure may suffer from overestimation.) This is accomplished by enclosing every number and every interval into an interval in $\mathbb{R}_M$, mostly by outward rounding.

Outward rounding is defined by *downward rounding* of the lower bound, meaning rounding to the next smaller or equal number in $\mathbb{R}_M$, and analogously *upward rounding* of the upper bound to the next greater or equal number in $\mathbb{R}_M$. Following the notation proposed in [KNN+02], we denote the outward rounding of an interval $\boldsymbol{x}$ as $\text{flint}(\boldsymbol{x})$. Within this work we assume $\mathbb{R}_M$ to be extended by symbols for $-\infty$ and ∞ (which is requested by the IEEE 754 standard and realized in SONIC). Thus outward rounding is possible for all intervals in ${}^*\mathbb{IR}$. For vectors outward rounding is applied componentwise.

To preclude rounding errors and attain verified solutions, outward rounding has to be applied for numbers, point intervals, and every interval in the computation

process. Be aware that herein a set consisting of a single number not in $\mathbb{R}_M$ is enclosed by an interval with diameter larger than zero. Besides, outward rounding can cause a function enclosure to overestimate the function range—even if the exact range could be computed with interval analysis in ${}^*\mathbb{IR}$.

Note that mathematical rigor of interval analysis on the machine numbers is ensured by the theory of interval analysis and appropriate handling of rounding when transferring theory to practice. To regulate how machine numbers are handled, the IEEE 754 standard was enacted in 1985 (and updated in 2008 [iee08]). It provides definitions for different rounding modes as rounding to the nearest number, upward and downward rounding. An IEEE 754 format also has to comprise symbols for the infinities $-\infty$ and ∞. Since rounding errors are inevitable and for the sake of convenient reading, they are not mentioned in this work unless necessary. Correct rounding is assumed implicitly in all of the algorithms within this work.

Of course, the reliability of numerical results relies on several components. The program itself, as well as the underlying soft- and hardware, has to work correctly. If these presumptions are met, interval analysis can rise to its strength and provides mathematically rigorous results.

Chapter 2

Nonlinear systems of equations

If there is a problem you can't solve,
then there is an easier problem you can solve: find it.

George Pólya

The upcoming chapter is dedicated to the definition of the systems we want to study as well as to the most important techniques for solving these systems reliably. Again, the descriptions are given matching the functionality in SONIC. Thus they follow [Bee06] and [Wil04] in wide parts and our selection of presented methods is inevitably biased.

The classical formulation of a nonlinear system of equations is based on a function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ and formulated as

$$f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix} = 0 \tag{2.1}$$

with $x \in D$. (The right-hand side is to be understood as a column vector of m zeros.) The solution set $\mathcal{S}$ of this problem can be denoted as

$$\mathcal{S} = \{x \in D \mid f(x) = 0\}.$$

Before we transfer this real-valued problem to the interval formulation of a nonlinear systems of equations we are interested in, we want to define a precision requirement for boxes.

Definition 2.1 (Small enough)
For a given precision vector $\psi \in \mathbb{R}^n_+$ *and* threshold vector $\Psi \in \mathbb{R}^n_+$*, a box* $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ *is called* small enough *if it fulfills at least one of the precision requirements*

- $\operatorname{width}(\boldsymbol{x}_i) \leq \psi_i$ *for all* $i \in \{1, \ldots, n\}$

or

- *there exists an index* i *with* $\operatorname{mig}(\boldsymbol{x}_i) \geq \Psi_i$.

Thus the width of a small enough box is less or equal ψ_i in every direction or one of its components exceeds the threshold given by the ψ_i. The latter condition is not commonly found when interval systems are considered. The author enhanced the usual precision requirements by introducing the threshold vector Ψ for the purpose of handling unbounded boxes. The threshold vector provides a stopping criterion for handling half-bounded boxes with components of large mignitude that cannot be discarded by other means. For details see section 2.2.2.

We now proceed to reformulate system (2.1) into an interval problem.

Definition 2.2 (Interval nonlinear system of equations)
An interval nonlinear system of equations is given by

- *a function* $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$,
- *a* startbox $\boldsymbol{x}^{(0)} \in {}^*\mathbb{IR}^n$,
- *the system* (2.1) *with solution set* $\mathcal{S} = \{x \in \boldsymbol{x}^{(0)} \mid f(x) = 0\}$,
- *a precision vector* $\psi \in \mathbb{R}^n_+$ *and*
- *a* threshold vector $\Psi \in \mathbb{R}^n_+$.

We call an interval nonlinear system of equations square *(*non-square*) if* $n = m$ *(*$n \neq m$*).*

The solution of an interval nonlinear system of equations is defined as a set of small enough solution boxes $\boldsymbol{x}^{(i)} \in {}^*\mathbb{IR}^n$ *fulfilling the conditions*

- $i \in \{1, \ldots, s\}, \quad s \in \{0\} \cup \mathbb{N}$ *and* $\mathcal{S} \subseteq \bigcup_{i=1}^{s} \boldsymbol{x}^{(i)}$.

(Thus the solution list may also be the empty set.)

Note that an unbounded startbox $\boldsymbol{x}^{(0)}$ is allowed in the context of this work. Thus we can even handle problems for which not all variables are bounded or not all boundaries are known. Consequently we do not need to restrict variables artificially.

Every set of solution boxes is guaranteed to "cover" all solutions of system (2.1) in the startbox $\boldsymbol{x}^{(0)}$. Thus no element of $\mathcal{S}$ is contained in $\boldsymbol{x}^{(0)} \setminus (\boldsymbol{x}^{(1)} \cup \ldots \cup \boldsymbol{x}^{(s)})$. However, not every solution box of system (2.1) is required to actually contain a solution of system (2.1). How to compute a set of solution boxes is the main topic of this chapter. (Usually there exist various sets of solution boxes for one and the same interval nonlinear system of equations. We are interested in determining just one of them.)

By choosing a small vector ψ for the required precision of the solution boxes, we request narrow solution boxes. However, in implementations a smaller precision vector typically causes higher computational costs. Further, if system (2.1) does not have isolated solutions but has a manifold of solutions, a choice of ψ with entries of small absolute value results in an enormous number of solution boxes.

The definition of the solution set has two important ramifications for algorithms to solve interval nonlinear systems of equations. Firstly, if we end up with an empty list of solution boxes, then there exists no solution to system (2.1) in the entire startbox $\boldsymbol{x}^{(0)}$. Secondly, if we attain a non-empty list of solution boxes, we still do not know in which way the number of solution boxes is related to the number of solutions of system (2.1) and in which boxes the real-valued solution may be seated. To be able to make such statements we need further methods supplied by interval analysis. Those methods allow the verification of the existence or even uniqueness of a solution within a box and are treated in Chapter 4.

We could also demand disjoint interiors of the solution boxes. However, this additional requirement would prevent the deliberate inflation of boxes within the computation or for verification methods (as done for exclusion regions [SN04, Bee06] or for epsilon-inflation). We decided to waive this additional requirement.

Note further, that we can augment nonlinear systems of equations by inequalities. Those then have to be checked by interval methods, too.

Unless stated otherwise, all problems considered in this work are interval nonlinear systems of equations. For ease of notation they are also simply referred to as "problems" or "nonlinear systems".

2.1 Basic approach: branch-and-bound

The basic strategy to compute a set of solution boxes from a startbox is a branch-and-bound algorithm. Its concept includes the application of methods that are meant to either contract a given box or to determine that it does not contain any solution of the system and to discard it. If a box is not discarded, it is subdivided into two (or more) smaller subboxes. In the same way, the approach is applied to the evolving subboxes. The branch-and-bound algorithm stops as soon as all boxes have been discarded or have reached the precision demand for solution boxes.

Algorithm 2.1 BRANCH-AND-BOUND (box $\boldsymbol{x}^{(0)}$, working list $L_w := \{\boldsymbol{x}^{(0)}\}$, solution list $L_s := \emptyset$)

```
1: while L_w ≠ ∅ do
2:    pick a box x from the working list L_w
3:    if box x is small enough then
4:       move x from L_w to L_s
5:       continue {ends current pass of the while-loop}
6:    end if
7:    if box x does not contain a solution then
8:       discard x from L_w
9:       continue {ends current pass of the while-loop}
10:   end if
11:   apply contractors (Alg. 2.8)
      {try to contract box x while maintaining all comprised solutions}
12:   subdivide x (Alg. 2.6)
13:   in L_w: replace x by all its subboxes
14: end while
```

Algorithm 2.1 sketches the basic branch-and-bound algorithm. In a first step the *working list* L_w is initialized with the startbox. Afterwards, the algorithm repeats to analyze individual boxes from the working list. Contractors determine whether the current box can be contracted or even discarded from the working list. This contracting or discarding of boxes is called the "bounding step" of the algorithm. (Literally we cut branches of the branch-and-bound tree as depicted in Figure 2.1.) If a box meets the precision requirements for a solution box, it is moved from the working list to a *solution list* L_s. If the box is not yet small enough, it subdivided in the "branching step" and replaced by the evolving subboxes. This procedure is repeated until the working list is empty.

Because we only discard boxes that are proved to contain no solution of system (2.1), the solution set $\mathcal{S}$ is covered by the boxes in solution and working list at all times, meaning

$$\mathcal{S} \subseteq \bigcup_{\boldsymbol{x} \in L_w \cup L_s} \boldsymbol{x}$$

holds throughout the branch-and-bound algorithm. At the end of the algorithm, the working list is empty and the solution list comprises all solutions in its boxes.

Algorithm 2.1 gives a formulation of the branch-and-bound algorithm with loops. If we would reformulate the branch-and-bound algorithm into a recursive form, the concept of a recursion level would arise intuitively. Since the formulation with loops is suited for parallelization (cf. section 5.4), we stay with that formulation and just transfer the concept of recursion levels for our purposes. The recursion level of a box then equals its "depth" in the tree evolving from the branch-and-bound algorithm. Note that in a single branching step the box may

be subdivided iteratively (see section 2.2).

Definition 2.3 (Recursion level)
The recursion level *of the startbox* $\boldsymbol{x}^{(0)}$ *is set to* 1 *(*$\mathrm{rl}(\boldsymbol{x}^{(0)}) = 1$*). For a box* $\boldsymbol{x}$ *of recursion level* $\mathrm{rl}(\boldsymbol{x}) \in \mathbb{N}$*, the recursion level of all subboxes* $\boldsymbol{x}_{\mathrm{sub}}$ *obtained by a single branching step in the branch-and-bound algorithm is given by* $\mathrm{rl}(\boldsymbol{x}_{\mathrm{sub}}) = \mathrm{rl}(\boldsymbol{x}) + 1$.

We speak of an upper *recursion level, for recursion levels with small values of* r *and of a* deeper *recursion level if* r *is large.*

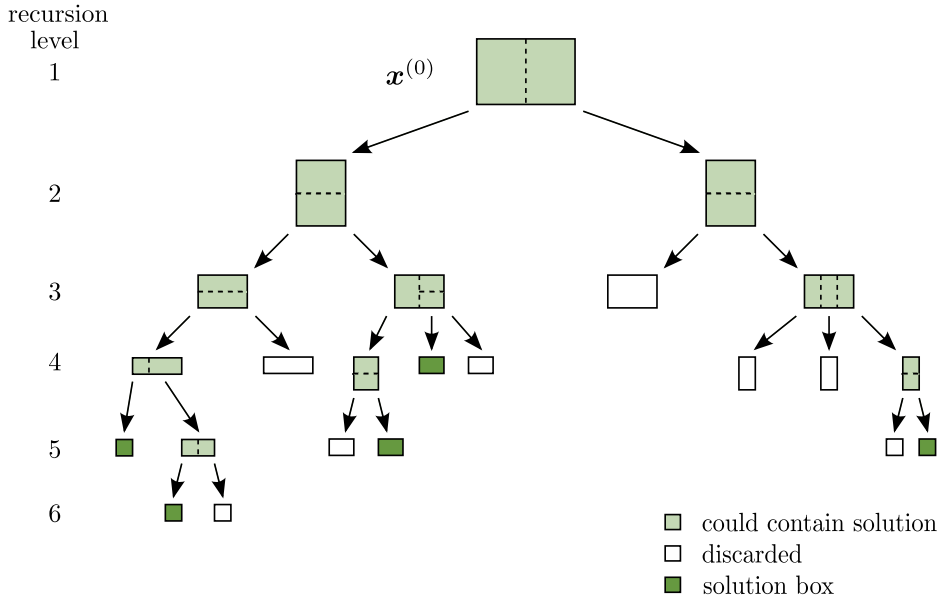


Figure 2.1: Scheme of the branch-and-bound tree for a two-dimensional example. The tree begins in the startbox $\boldsymbol{x}^{(0)}$ and evolves due to the subdivision of boxes. The dashed lines indicate where a box is to be subdivided. The boxes for which the existence of a solution is excluded are depicted in white. The solution list consists of all non-white leaves of the tree.

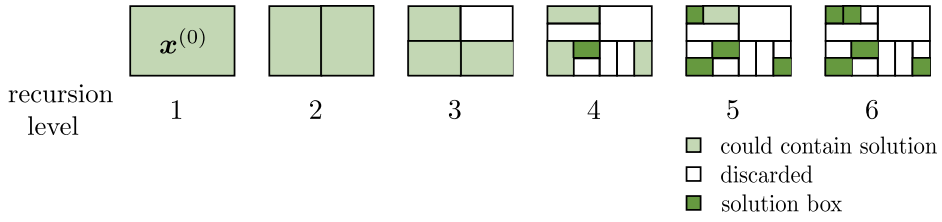


Figure 2.2: Scheme of a branch-and-bound algorithm for a two-dimensional example. Shown is the sequence of computation steps.

In Figure 2.1 and 2.2 a possible execution process for the branch-and-bound

algorithm is illustrated. These figures are simplified depictions and showed only the discarded boxes, not the contracted ones. The status of the boxes in each recursion level is shown in the state as in line 14 of Algorithm 2.1. We find upper recursion levels at the top of the branch-and-bound tree and the deeper recursion levels at its bottom.

For solving nonlinear systems of equations, the branch-and-bound algorithm provides two major benefits. By subdividing boxes, in general, sharper enclosure for the function of the nonlinear system can be obtained. By contracting and discarding boxes that do not contain a zero, the exponentially increasing number of boxes is reduced. If a box can be discarded, there is no need for further investigation of its subboxes. Both steps—contracting and discarding boxes—save computational costs that otherwise would be needed in the following recursion levels.

We want to append some deliberations concerning the correctness of the branch-and-bound algorithm as given in Algorithm 2.1. Firstly, the algorithm has to terminate, so the working list L_w has to run empty at some point. This happens when, in every step, we can discard a box completely or replace it by one or more smaller subboxes until all boxes reached one of the criteria for small enough boxes. (Special consideration would be needed if we would use methods inflating boxes during the computation as the so-called exclusion regions [SN04].) Furthermore, every contractor has to terminate.

For the correctness of the algorithm we have to preserve all solutions contained in the startbox $\boldsymbol{x}^{(0)}$. Under special conditions for the branching (subdivision) and bounding (the application of contractors) step we want to argue that all solutions in the startbox are contained in the union of all elements in the lists L_w and L_s, i.e.

$$\boldsymbol{x}^{(0)} \cap S = (L_w \cup L_s) \cap S \tag{2.2}$$

at the end of each statement of Algorithm 2.1. This formula is clearly fulfilled right at the beginning of the branch-and-bound algorithm because then

$$L_w = \{\boldsymbol{x}^{(0)}\} \text{ and } L_s = \emptyset$$

hold and it is trivial to conclude

$$\boldsymbol{x}^{(0)} \cap S = L_w \cap S = (L_w \cup L_s) \cap S.$$

It is further simple to see that nothing changes if a box is either small enough and moved from L_w to L_s or discarded because it cannot contain a solution of the system.

To preserve property (2.2) in the branching step, we need to ensure that the evolving subboxes $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}$ form a covering for the dissected box $\boldsymbol{x}$ fulfilling $\boldsymbol{x} = \boldsymbol{x}^{(1)} \cup \ldots \cup \boldsymbol{x}^{(k)}$, $k \in \mathbb{N}$. Then

$$\boldsymbol{x} \cap S = (\boldsymbol{x}^{(1)} \cup \ldots \cup \boldsymbol{x}^{(k)}) \cap S \text{ for all } \boldsymbol{x} \in L_w.$$

In the bounding step we have to take care to apply just contractors. For those the prerequisite

$$S \cap \boldsymbol{x} = S \cap \mu(\boldsymbol{x})$$

is fulfilled for arbitrary boxes $\boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}$. Thus, replacing a box $\boldsymbol{x}$ in L_w by $\mu(\boldsymbol{x})$ does not interfere with property (2.2). Within a branch-and-bound algorithm we have to take care to fulfill the mentioned conditions to achieve correctness of the algorithm.

2.2 Branching / Subdivision

In our algorithm "branching" stands for the subdivision of boxes. The union of these subboxes of a box $\boldsymbol{x}$ covers the box $\boldsymbol{x}$ itself and thus contains all solutions included in $\boldsymbol{x}$. Furthermore, subboxes are chosen to be disjoint up to their relative boundaries. Subdivision of boxes can be realized by different methods, some of which will be discussed in the following section.

For most subdivision strategies, we subdivide a box into two subboxes along a *subdivision direction* d. The point in which we subdivide is called the *subdivision point* $p \in \boldsymbol{x}_d$. In a simple bisection $\boldsymbol{x}$ is parted into the two subboxes

$$\begin{aligned} \boldsymbol{x}^{(1)} &= (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, [\underline{x}_d, p], \boldsymbol{x}_{d-1}, \ldots, \boldsymbol{x}_n)^T \\ \boldsymbol{x}^{(2)} &= (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, [p, \overline{x}_d], \boldsymbol{x}_{d-1}, \ldots, \boldsymbol{x}_n)^T \end{aligned} \tag{2.3}$$

with union $\boldsymbol{x}^{(1)} \cup \boldsymbol{x}^{(2)} = \boldsymbol{x}$. (When saying "we subdivide $\boldsymbol{x}$ at some subdivision point" we always mean a subdivision in the subdivision direction.)

The presumably most simple way to subdivide a box is to subdivide it into two subboxes of equal size in the component of maximum width. Other methods use more sophisticated approaches to determine subdivision direction and subdivision point or subdivide in more than two subboxes. To ensure that we always achieve subboxes smaller than the subdivided box, we do not allow to subdivide in point intervals or to choose the subdivision point p equal to an endpoint of the box in the subdivision direction.

As said before, the main motivation for subdivision is to get smaller boxes and to attain sharper function enclosures according to relation (1.8). A sharper enclosure in turn is more likely to give information about the existence of roots in the box. Thus a sensible choice of the subdivision strategy is crucial for the efficiency of the overall branch-and-bound algorithm. (For a numerical impression, see section 2.2.9.)

In the following, parameter SubMinNew (SubMaxNew) denotes the minimum (maximum) number of subboxes a branching step should yield. Both parameters are used to control the subdivision strategies.

2.2.1 Subdivision using gaps

As stated before, the result of a relational division (Definition 1.6) is a union of disjoint boxes if the interval in the denominator contains zero in its interior. For two intervals $\boldsymbol{y}$ and $\boldsymbol{z}$ in ${}^*\mathbb{IR}$ with $\overline{y} < \underline{z}$ we refer to the (exceptionally) open interval $(\overline{y}, \underline{z})$ as the *gap* between $\boldsymbol{y}$ and $\boldsymbol{z}$. If we store both intervals $\boldsymbol{y}$ and $\boldsymbol{z}$, we can exploit the implicit information about the gaps for subdivision.

The following method we call "subdivision using gaps" thus does not simply subdivide boxes, but uses the additional information about the location of roots originating from relational division to "cut out" pieces without any computational cost. Thus it is not a pure subdivision method but rather a subdivision combined with a contractor. The method is discussed at length in [Wil04] and [Bee06]. Similar approaches have been proposed in [HJvE99] and [Han92]. We want to recapitulate it in short for a better overview of the methods used in the branching step.

When applying subdivision using gaps, the biggest gaps (relative to the width) are chosen to separate a box. If subdivision is applied in direction d where

$$\boldsymbol{x}_d = \square(\boldsymbol{y} \cup \boldsymbol{z}), \quad y, z \in {}^*\mathbb{IR}^n$$

then box $\boldsymbol{x}$ is split into the subboxes

$$\boldsymbol{x}^{(1)} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, \boldsymbol{y}, \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$$

and

$$\boldsymbol{x}^{(2)} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, \boldsymbol{z}, \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T.$$

An analogous procedure is chosen if $\boldsymbol{x}_i$ is a union of more than two disjoint boxes. However, not all gaps have to be made use of, see Algorithm 2.2.

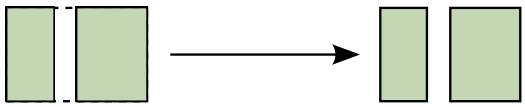


Figure 2.3: Subdivision using gaps

Algorithm 2.2 SUBDIVISION USING GAPS (box $\boldsymbol{x}$, SubMaxNew)

1: compute relative size of gaps in the components $\boldsymbol{x}_i$
2: **while** box $\boldsymbol{x}$ is a union of more than SubMaxNew parts **do**
3: find direction i of smallest gap
4: in direction i: enclose all parts into one
5: **end while**
6: remove $\boldsymbol{x}$ from L_{sub}
7: add parts of $\boldsymbol{x}$ to L_{sub}

In SONIC, finite unions of boxes are stored in a special data type (`XDivision`). This data type allows us to store information about gaps resulting from relational division. As long as we do not want to exploit the additional information about the gaps, we compute with the convex hull of the set of boxes as if dealing with a usual box. When subdividing however, we make use of the information about gaps.

The minimal width a gap must have to be used for subdivision can be chosen in the Controls-File with the variable *MinRelGapsizeForSubdivisionUsingGaps* The number of resulting subboxes can be limited by choosing the value of *SubMaxNew*.

To subdivide a given box if no information about gaps can be exploited, a promising subdivision direction and a subdivision point have to be determined. Different strategies for this purpose are presented in the next sections.

2.2.2 Handling of unbounded variables

Unbounded variables cannot be handled sensibly by the strategies in the following sections. Therefore they have to be treated separately. We adjust the strategy given in [Bee06] to attain a subdivision scheme that can handle boxes with multiple unbounded components.

For a box $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ containing one or more component equal $[-\infty, \infty]$, the subdivision direction is determined as the smallest $d \in \{1, \ldots, n\}$ with $\boldsymbol{x}_d = [-\infty, \infty]$. The subdivision point is chosen as some fixed number $\delta_1 \in \mathbb{R}$. If $\boldsymbol{x}$ has only bounded or half-bounded components $\boldsymbol{x}_i = [-\infty, a]$ or $\boldsymbol{x}_i = [a, \infty]$ with $a \in \mathbb{R}$, we subdivide the box in one of the half-bounded components. This component is dissected into a bounded interval and another half-bounded one. The subdivision direction d is chosen from the index set of all half-bounded components of $\boldsymbol{x}_i$ by considering the function

$$v : {}^*\mathbb{IR} \setminus \{[-\infty, \infty]\} \to \mathbb{R}$$
$$v(\boldsymbol{y}) := \begin{cases} \overline{y} & \text{for } \underline{y} = -\infty \\ -\underline{y} & \text{for } \overline{y} = \infty \end{cases}$$

and picking the direction d maximizing this function.

$$v(\boldsymbol{x}_d) = \max_{\substack{i \in \{1,\ldots,n\} \\ \text{width}\,(\boldsymbol{x}_i) = \infty}} v(\boldsymbol{x}_i)$$

As for the subdivision point, our strategy is to begin to "split off" bounded intervals of small width. Small intervals because we want to avoid having to

subdivide the evolving bounded intervals. If the box containing the left-over half-bounded interval in direction d cannot be discarded in the bounding step of the branch-and-bound algorithm, in the next branching steps we split off bounded intervals of increasing width in direction d. Thus the mignitude (in direction d) of the leftover half-bounded box grows. To attain this goal, we use a predefined parameter $\delta_2 \in \mathbb{R}_+$ and the subdivisions

$$[-\infty, a] = [-\infty, \min\{-\delta_2, 2a\}] \cup [\min\{-\delta_2, 2a\}, a]$$

and

$$[a, \infty] = [a, \max\{\delta_2, 2a\}] \cup [\max\{\delta_2, 2a\}, \infty]$$

for any $a \in \mathbb{R}$. The complete algorithm for determining a subdivision direction and subdivision point for an unbounded box is given in Algorithm 2.3.

Algorithm 2.3 SUBDIVISION OF AN UNBOUNDED BOX (box $\boldsymbol{x}$, parameters δ_1, δ_2)

if there exist unbounded components of $\boldsymbol{x}$ **then**
  if there exists $i \in \{1, \ldots, n\}$ with $\boldsymbol{x}_i = [-\infty, \infty]$ **then**
    determine subdivision direction $d \leftarrow \min_{\boldsymbol{x}_i = [-\infty, \infty]} i$
    subdivision point p is set to δ_1
  else {only half-bounded and bounded components}
    subdivision direction d is determined as $v(\boldsymbol{x}_d) = \max_{j \in \{1, \ldots, n\}} v(\boldsymbol{x}_j)$
    if $\inf(\boldsymbol{x}_d) = -\infty$ **then**
      determine subdivision point p as $\min\{-\delta_2, 2 \cdot \sup(\boldsymbol{x}_d)\}$
    else
      determine subdivision point p as $\max\{\delta_2, 2 \cdot \inf(\boldsymbol{x}_d)\}$
    end if
  end if
end if

To prevent that the branch-and-bound algorithm does not terminate due to half-bounded boxes that are subdivided over and over without being discarded in the bounding step, the author introduced the threshold vector Ψ within Definition 2.1. It allows the branch-and-bound algorithm to stop when the mignitude of a component $\boldsymbol{x}_i$ exceeds Ψ_i.

As for the choice of the parameters, experience showed that many test problems have a root in zero. Subdividing with $\delta_1 = 0$ may thus cause a cluster effect. Therefore the author favors to shift the subdivision point slightly and to set δ_1 to 0.1. Furthermore, the minimum width for one of the subboxes resulting from the subdivision of unbounded boxes can be shown to be $\delta_2/2$ in direction d. Thus for small δ_2 many branching steps may be needed to find small enough boxes. This is especially true when, in one component, searching for a zero with large absolute value in a half-bounded box with small mignitude in that very component (as evolving when we start with components $[-\infty, \infty]$). If calculating with a large

δ_2, on the other hand, the evolving box has a large diameter in the subdivision direction and many branching steps may be needed, if a root with small absolute value is to be found. (This problem still occurs when starting with a box containing half-bounded intervals of large mignitude.) The author proposes the value $\delta_2 = 1$ for general problems.

The default value for Ψ in SONIC is set to 10^{100}. We tested to replace the factor 2 in $\min\{-\delta_2, 2a\}$ and $\max\{\delta_2, 2a\}$ by 16.) The results for changing the factor 2 to 16 were ambiguous, therefore no "optimal" factor can be given. We further conducted some tests with different values for Ψ (10^{10}, 10^{20} and 10^{50}) for a problem modified to need to be stopped due to the new criterion based on Ψ. Herein the change of Ψ had no significant effect on the computational costs.

The handling of unbounded and half-bounded boxes in SONIC was changed to the described strategy. The parameters have been changed as proposed for the practical use of the solver.

Another way to handle unbounded variables are variable transformations as mentioned in [BFLW05].

2.2.3 Determining a direction for subdivision

From now on subdivision strategies for bounded boxes are considered. In a first step we want to determine a direction $d \in \{1, \ldots, n\}$ for subdivision.

In each of the following strategies to select a subdivision direction d, the direction is additionally demanded to be chosen such that $\text{width}\,(x_d) > \psi_d$. This additional demand prevents subdivision in directions, in which the box already fulfills the given precision requirements. (However, subdividing in directions fulfilling $\text{width}\,(x_d) < \psi_d$ may still improve function enclosures.)

MaxDiam The most basic strategy for choosing a subdivision direction is to take the direction with the largest width of the box. We thus determine d by

$$\text{width}\,(\boldsymbol{x}_d) = \max_i \text{width}\,(\boldsymbol{x}_i).$$

This strategy is called MaxDiam. It causes less computational costs than the following strategies to elect a subdivision direction. A further advantage of the MaxDiam strategy is that it always terminates (even without the additional demand of not subdividing components fulfilling the precision requirements).

MaxSmear The MaxSmear strategy originates in the works of Kearfott and Novoa [KN90]. It uses the Jacobian matrix as a clue to judge the influence of

the $\boldsymbol{x}_i$ to a function enclosure. The subdivision direction is chosen as the index of the component with the highest influence on the function value. To establish the strategy one considers the function enclosures calculated by a centered form as given in equation (1.9). Because we examine a function with more-dimensional range, we have to substitute the gradient by an enclosure $\boldsymbol{Df}$ of the Jacobian matrix. The width of the enclosure $\boldsymbol{f}_i(\boldsymbol{x})$ is

$$\operatorname{width}\left(\boldsymbol{f}_i(\boldsymbol{x})\right) = \sum_{j=1}^{n} |\boldsymbol{Df}_{i,j}| \cdot \operatorname{width}\left(\boldsymbol{x}_j\right).$$

For the MaxSmear strategy, the subdivision direction d is chosen to fulfill

$$|\boldsymbol{Df}_{i_d,d}| \cdot \operatorname{width}\left(\boldsymbol{x}_d\right) = \max_{i,j} \left(|\boldsymbol{Df}_{i,j}| \cdot \operatorname{width}\left(\boldsymbol{x}_j\right)\right)$$

for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$ and a suiting function component i_d.

MaxSmearDiam The MaxSmearDiam strategy mentioned by Hansen [Han92] resembles the MaxSmear strategy. However, instead of the magnitude, the width is considered to estimate the influence of the $\boldsymbol{x}_j$ to the function enclosure for f over $\boldsymbol{x}$. The subdivision direction d is determined by

$$\operatorname{width}\left(\boldsymbol{Df}_{i_d,d}\right) \cdot \operatorname{width}\left(\boldsymbol{x}_d\right) = \max_{i,j} \left(\operatorname{width}\left(\boldsymbol{Df}_{i,j}\right) \cdot \operatorname{width}\left(\boldsymbol{x}_j\right)\right).$$

MaxSumMagnitude The MaxSumMagnitude strategy is mentioned in [Han92] and discussed in detail in [SA02]. It determines d such that $\boldsymbol{x}_d$ is the component of $\boldsymbol{x}$ with maximum contribution to all $\boldsymbol{f}_i$ and can be modified by a parameter $\omega_1 \in \mathbb{R}_+$ according to the rule

$$\sum_{i=1}^{m} |\boldsymbol{Df}_{i,d}| \cdot \operatorname{width}\left(\boldsymbol{x}_d\right)^{\omega_1} = \max_{j} \sum_{i=1}^{m} |\boldsymbol{Df}_{i,j}| \cdot \operatorname{width}\left(\boldsymbol{x}_j\right)^{\omega_1}.$$

In SONIC, 1.0 is the default value for ω_1.

ZeroNearBound In the ZeroNearBound strategy we search a function component f_{i_d} for which a root is supposed to be seated near the boundary of the considered box. A root near to a boundary is assumed if zero is near to one of the boundaries of the function enclosure $\boldsymbol{f}_{i_d}(\boldsymbol{x})$. As subdivision direction the variable d with $\boldsymbol{x}_d$ most influencing f_{i_d} is chosen. The goal of this strategy is to subdivide

such that one of the evolving subboxes can be discarded in the next bounding steps. We determine i_d as the variable i minimizing the "symmetry measure"

$$\sigma(\boldsymbol{f}_i(\boldsymbol{x})) := \frac{\min\{-\inf(\boldsymbol{f}_i(\boldsymbol{x})), \sup(\boldsymbol{f}_i(\boldsymbol{x}))\}}{\max\{-\inf(\boldsymbol{f}_i(\boldsymbol{x})), \sup(\boldsymbol{f}_i(\boldsymbol{x}))\}}$$

and the subdivision direction d as the solution of

$$|\boldsymbol{Df}_{i_d,d}| \cdot \text{width}\,(\boldsymbol{x}_d)^{\omega_2} = \max\left(|\boldsymbol{Df}_{i_d,j}| \cdot \text{width}\,(\boldsymbol{x}_j)^{\omega_2}\right)$$

for some parameter $\omega_2 \in \mathbb{R}_+$.

In SONIC, 1.0 is used as default value for ω_2.

Hybrid strategy by Beelitz In his thesis, Beelitz [Bee06] argues that every subdivision strategy works fine for some nonlinear problems—but none does work equally well for all.

Because of this, Beelitz developed a special hybrid strategy (based on a strategy used in GlobSol) to automatically choose a promising direction. This strategy really is a heuristic determining which of the above basic strategies to elect a subdivision direction is appropriate for a given box. The strategy was chosen with the intention to provide a robust and strong strategy for most nonlinear systems of equations. For that purpose different measures of the box are evaluated. A remark by Beelitz is that boxes with strongly varying width in different directions tend to have a negative influence on the performance of the overall solver. He introduced a threshold to determine whether the width of some $\boldsymbol{x}_d$ is "too large" in comparison to the others. In this case the subdivision direction d is determined by the MaxDiam strategy and the component is subdivided at its midpoint. ZeroNearBound is only used if the enclosure for $\boldsymbol{f}_{i_d}(\boldsymbol{x})$ is not "too small" and one boundary of the function enclosure is almost zero. A summary of the strategy is given in Algorithm 2.4. More details and a comparison of the basic strategies called by the hybrid strategy can be found in the work by Beelitz [Bee06].

The default parameters for this algorithm are currently set as $b_1 = 20$, $b_2 = 0.01$, $b_3 = 0.5$ and $b_4 = 1.0$.

Other strategies A further tempting strategy is to simply subdivide a box periodically in one direction after the other (*round-robin*). This procedure is suggested, e.g., in [SA02] and used by some interval solvers. Our own tests showed that this strategy is not to be favored in practice. Indeed, compared to the hybrid

Algorithm 2.4 HYBRID STRATEGY (box $\boldsymbol{x}$, Jacobian matrix $\boldsymbol{Df}$ $(\boldsymbol{x})$, parameters b_1, b_2, b_3 and b_4)

1: $w_{\max} = \max_{i=1,\ldots,n} \text{width}\,(\boldsymbol{x}_i)$
2: $w_{\text{mean}} = \frac{1}{n} \sum_{i=1}^{n} \text{width}\,(\boldsymbol{x}_d)$
3: **if** $w_{\max}/w_{\text{mean}} \geq b_1$ **then**
4: determine d by MaxDiam
5: **else**
6: compute i_d as in ZeroNearBound
7: **if** width $\left(\boldsymbol{f}_{i_d}(\boldsymbol{x})\right) > b_2$ **and** $\sigma(\boldsymbol{f}_{i_d}(\boldsymbol{x})) < b_3$ **then**
8: determine d by ZeroNearBound
9: **else**
10: compute i_d and d by MaxSmearDiam
11: **if** $\text{mag}(\boldsymbol{Df}_{i_d,d}) < b_4 \cdot \text{width}\,(\boldsymbol{Df}_{i_d,d})$ **then**
12: determine d by MaxSmearDiam
13: **else**
14: determine d by MaxSmear
15: **end if**
16: **end if**
17: **end if**

subdivision strategy, we observed increases in computation time and box number by factors as high as 70.

$\diamond$

Up to this point, all presented strategies were dedicated to the best choice for the subdivision direction. Nothing was said about the subdivision point. In the following, methods to determine one or more points to subdivide a box are discussed.

Mind that, when implementing the proposed strategies, one has to take additional care considering rounding issues. Every subdivision point is rounded to a number in $\mathbb{R}_M$. Thus one must ensure that the rounded subdivision point is still an element of $\boldsymbol{x}_d$. For each subdivision point p we thus compute

$$p' = \max\left(\underline{x}_d, \min\left(\overline{x}_d, \text{fl}(p)\right)\right)$$

and actually subdivide at p'.

2.2.4 Bisection

The most common way to subdivide boxes is a *bisection*, i.e. a subdivision into two subboxes. For applying bisection a subdivision direction d is determined by one of the methods described in section 2.2.3. If not stated otherwise, the subdivision point is chosen as the midpoint of $\boldsymbol{x}_d$. Consequently, subdivision is conducted as given by formula (2.3) with subdivision point

$$p = \text{mid}\,(\boldsymbol{x}_d).$$

By this choice, the box is subdivided into two subboxes of half volume.

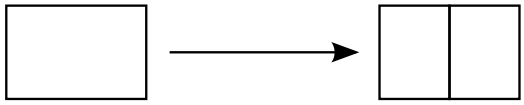


Figure 2.4: Bisection in the midpoint of one component

Bisection in the midpoint is the standard subdivision in SONIC as well as many other interval solvers.

2.2.5 Multisection

We speak of applying a *multisection* when subdividing a box into more than two subboxes along one direction. Of course, in this case we need to determine more than one subdivision point.

For testing purposes, we implemented multisection into three subboxes with equal width. The subdivision direction d was chosen by the hybrid strategy and subdivision of a box $\boldsymbol{x}$ was conducted to yield the subboxes

$$\boldsymbol{x}^{(1)} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, [\underline{x}_d, \underline{x}_d + 1/3 \cdot \text{width}\,(x_d)], \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$$
$$\boldsymbol{x}^{(2)} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, [\underline{x}_d + 1/3 \cdot \text{width}\,(x_d), \underline{x}_d + 2/3 \cdot \text{width}\,(x_d)], \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$$

and

$$\boldsymbol{x}^{(3)} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, [\underline{x}_d + 2/3 \cdot \text{width}\,(x_d), \overline{x}_d], \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$$

with union $\boldsymbol{x}^{(1)} \cup \boldsymbol{x}^{(2)} \cup \boldsymbol{x}^{(3)} = \boldsymbol{x}$.

For some of the tested problems the computation time decreases rapidly by applying the described multisection. In other cases computation time increases strongly. The author ascribes the positive behavior to situations where a cluster effect is prevented or more subboxes are discarded in upper recursion levels than for a simple bisection. An increase in computation time can be explained by

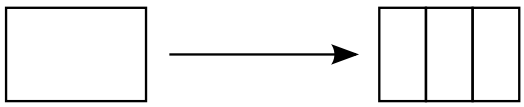


Figure 2.5: Multisection into three subboxes

subboxes that cannot be discarded immediately. In this case, a higher number of boxes has to be considered and thus more computation time is spent. This becomes important considering that the box number increases exponentially in the worst case. Using multisection into i parts instead of bisection, the maximum box number in a given recursion level r rises from 2^r to i^r.

Because we do not know about any universal strategy on when to apply multisection, it is not implemented in the current version of SONIC.

2.2.6 Non-equidistant subdivision

Standard subdivision yields subboxes of equal size. Another possibility the author wants to discuss at this point is a *non-equidistant* subdivision (Figure 2.6).

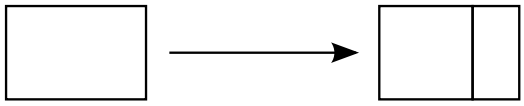


Figure 2.6: Bisection in a shifted subdivision point

In most cases, equidistant subdivision is applied because we do not have any information to determine in which part of a box a solution may lie. Therefore we also do not know where to subdivide a box to attain sharper function enclosures or to be able to discard a box. Using a non-equidistant subdivision can nevertheless have advantages. It can, for example, reduce the cluster effect.

A common cause of the cluster effect are symmetric intervals ($\boldsymbol{x}_i = [-a, a]$ with $a \in \mathbb{R}_+$) in combination with a root in zero. If we always split in the midpoint, the solution will be seated on the boundary of the evolving subboxes and thus be contained in both subboxes. If the indicated situation occurs in c dimensions, we get up to 2^c boxes that have to be handled in the working list. And because all of them contain a solution, we end up with the same amount of solution boxes for a single root of the system. One disadvantage in this situation is that multiple solution boxes contain the same solution. A second downside is that we cannot discard any of the subboxes and all of them need further consideration in the branch-and-bound algorithm.

Often the first problem is circumvented by increasing the startbox slightly so it becomes asymmetric (for example by changing $[-a, a]$ to $[-a, a + \delta]$ for some small $\delta \in \mathbb{R}_+$). In our tests, increasing the startbox worked extremely well for the problem *Brent7*. Only about a fifth of boxes and computation time were needed.

However, inflation of the startbox changes the solved problem. Thus solution boxes may be computed that do not lie in the originally given startbox. By employing a non-equidistant subdivision, we evade the cluster effect without changing the given problem. (Of course, this can only hold true if the new subdivision point does not fall on a solution itself.)

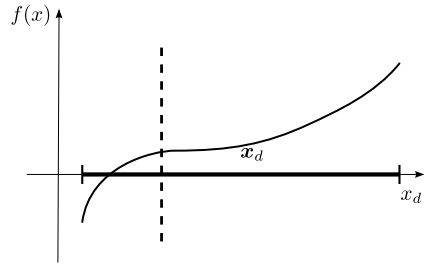
An arbitrary subdivision point p in $\boldsymbol{x}_d \in {}^*\mathbb{IR}$ can be determined by the formula

$$p = \underline{x}_d + \eta \cdot \text{width}\,(\boldsymbol{x}_d)$$

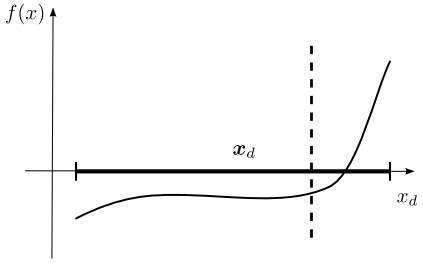
using a real-valued parameter $\eta \in (0, 1)$ to shift the subdivision point from the midpoint. For $\eta = 0.5$ (and provided exact computations) the subdivision point p coincides with the midpoint of the interval x_d. To avoid an increase of the total box number, the subdivision point should be shifted just slightly from the midpoint.

Another possibility we want to propose for a reasonable usage of the non-equidistant subdivision is to apply it in combination with the strategy ZeroNearBound. Remind that this strategy determines a subdivision direction d in which it is probable that a root of function component f_{i_d} is seated near to one boundary of the interval $\boldsymbol{x}_d$. We could now use a shifted subdivision to enlarge the subbox that is probable not to contain a root but to be discarded in the next bounding step. In order to subdivide appropriately, we need some further information, near to which one of the endpoints the root may lie. This information influences, whether we should shift the subdivision point to the upper or lower boundary of $\boldsymbol{x}_d$. Near to which endpoint the root is seated may be assessed by a function evaluation of f_{i_d} over the two subboxes $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, \underline{x_d}, \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$ and $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d-1}, \overline{x_d}, \boldsymbol{x}_{d+1}, \ldots, \boldsymbol{x}_n)^T$. The subdivision point can then be chosen near to that endpoint. Our tests for this strategy showed ambiguous results, but contained encouraging time reduction for individual problems. For other problems it seemed that, the assessed information about the function was not enough to determine the best subdivision point. This can be ascribed to the different possibilities for the function behavior on an interval, see the sketches in Figure 2.7. How the shift η should be determined exactly thus needs some further investigations.

Note further that, if we evaluate the function component f_{i_d} over a box $\boldsymbol{y}$ with $\boldsymbol{y}_d$ being a proper subintervals of $\boldsymbol{x}_d$ and its enclosure would not contain zero, we could yet "shave off" that part of box $\boldsymbol{x}$. Our tests, however, showed that for all our test problems $\boldsymbol{f}_{i_d}$ contained zero even when evaluated for $\boldsymbol{y}_d = \underline{x}$ or $\boldsymbol{y}_d = \overline{x}$.



(a) Subdivision point should be shifted towards the lower boundary



(b) Subdivision point should be shifted towards the upper boundary

Figure 2.7: Two functions with roots near to the boundaries of interval $\boldsymbol{x}_d$

The option of a shifted midpoint was also introduced into SONIC, where parameter η can be found as *ShiftSubPoint*.

2.2.7 Iterated subdivision

A further option for subdividing a box is to subdivide it repeatedly before executing the bounding step of the branch-and-bound algorithm. In effect, one can apply more than one subdivision per box and recursion level in a single branching step.

The subdivision to be applied in each iteration step can be chosen among the strategies presented above. Only if a box with unbounded components is to be subdivided, the only option is to evoke subdivision for unbounded boxes. Subdivision is iterated until the desired number of subboxes is attained. Boxes fulfilling the precision requirements for solution boxes are moved directly to the solution list. Algorithm 2.5 illustrates the approach.

Note that intermediate boxes are not considered in the bounding step of the branch-and-bound algorithm and do not contribute to the overall box number.

◇

If a subdivision into four or more boxes is aspired, one has to ponder, whether to use a multisection or an iterated subdivision. Generally speaking, the iterated subdivision is the more sophisticated strategy because for every subdivision a new subdivision direction is chosen. Even if it increases the computational costs to compute subdivision directions we recommend to use iterated subdivision if SubMinNew > 4.

Consult Table 2.1 for average results for multisection into three boxes and iterated subdivision with SubMinNew $= 3$. What the table does not show is that

Algorithm 2.5 ITERATED SUBDIVISION (list of boxes L_{sub}, SubMinNew)

```
1: while L_sub is not empty and contains fewer than SubMinNew subboxes do
2:   pick first box y from list L_sub
3:   if y is small enough then
4:     move y to solution list L_s (box is removed from L_sub)
5:   else
6:     if y is unbounded then
7:       apply subdivision of unbounded boxes (Alg. 2.3)
8:     else
9:       determine subdivision direction d
10:      determine subdivision point(s) for subdivision in direction d
11:      subdivide y
12:      remove y from list L_sub
13:      attach parts of y to list L_sub
14:    end if
15:  end if
16: end while
```

multisection provided better computation time for *Brent7* (factor 0.24) and *7er-System* (factor 0.56). These results even improve the computation time compared to an iterated subdivision with a minimum number of three new boxes (yielding time factors 0.74 and 1.16). The most probable explanation for this result is that the derived subboxes have simply been more advantageous in the overall algorithm. A further reason for the observed behavior could be that only one subdivision direction is needed for multisection whereas multiple directions have to be determined for iterated subdivision.

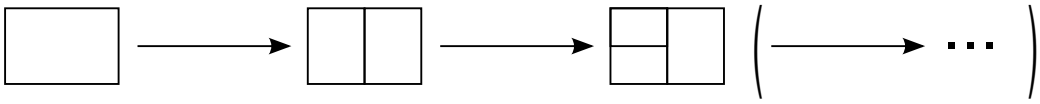


Figure 2.8: Iterated subdivision

2.2.8 Subdivision strategy in SONIC

The parameters SubMinNew and SubMaxNew are also used in SONIC. They control the implemented subdivision strategy. In this strategy, subdivision using gaps is applied followed by iterated subdivision. In the default settings, each step of the iterated subdivision of bounded boxes is a bisection in the midpoint. The subdivision direction is determined by the hybrid strategy. The overall subdivision strategy yields at least SubMinNew subboxes and at most SubMinNew subboxes. The default values are SubMinNew = 2 and SubMaxNew = 16. If not stated otherwise, all further tests in this work are conducted with the standard subdivision

strategy.

In the branch-and-bound algorithm given in Algorithm 2.1 we evoke Algorithm 2.6 for subdivision.

Algorithm 2.6 SUBDIVIDE (box $\boldsymbol{x}$, ψ, Ψ, SubMinNew, SubMaxNew)

```
initialize the list L_sub of subboxes with x
{small enough boxes will be removed from L_sub immediately}
if L_sub is not empty and contains fewer than SubMinNew subboxes then
  if subdivision using gaps should be used then
    apply subdivision using gaps (Alg. 2.2)
  end if
end if
if L_sub is not empty and contains fewer than SubMinNew subboxes then
  apply iterated subdivision to L_sub (Alg. 2.5) {includes subdivision of unbounded boxes (Alg. 2.3)}
end if
```

2.2.9 Numerical studies

We conducted some numerical studies to give an impression of the huge influence the subdivision strategy have on the computation time and box number needed by the branch-and-bound algorithm.

To this end, we give the average ratio of box number and computation time over test set $\mathcal{T}_B$ (cf. appendix). The numbers $\text{boxes}_{\text{init}}$ and $\text{time}_{\text{init}}$ refer to the box number and computation time needed if running the standard subdivision as presented in the last section. The values $\text{boxes}_{\text{mod}}$ and time_{mod} refer to the box number and computation time for a modified subdivision strategy.

The average box and time ratio are computed by

$$\text{boxes}_{\text{ratio}} := \frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{boxes}_{\text{mod}}}{\text{boxes}_{\text{init}}}$$

and

$$\text{time}_{\text{ratio}} := \frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{time}_{\text{mod}}}{\text{time}_{\text{init}}}.$$

In Table 2.1 we list the results for some variations of the standard subdivision strategy in SONIC, especially of parameter SubMinNew and the shift η for non-equidistant subdivision. (In this case, we employ the box-intern hierarchy as described in section 3.3.) Beyond this we show our measurements for a trisection, a multisection into three equally sized subboxes. Two versions of this multisection have been implemented and tested. For the first test, we subdivided every box into

three equidistant subboxes. For a second test, trisection was only applied once per component of the startbox $\boldsymbol{x}^{(0)}$. Afterwards, the standard subdivision strategy was deployed. This strategy was chosen in an attempt to reduce the cluster effect while at the same time restraining the (potentially) exponential increase in box number caused by multisection. (Remember that in the worst case trisection subboxes would result in 3^r boxes in recursion level r whereas bisection would only cause 2^r boxes to be considered.)

setting		$\text{boxes}_{\text{ratio}}$	$\text{time}_{\text{ratio}}$
SubMinNew	2	1.00	1.00
SubMinNew	3	1.01	0.80
SubMinNew	4	1.02	0.71
SubMinNew	5	1.16	0.70
SubMinNew	8	1.45	0.65
SubMinNew	16	2.18	0.69
η	0.1	8.34	8.57
η	0.2	2.27	2.40
η	0.3	1.33	1.35
η	0.4	3.82	4.59
η	0.45	6.69	6.75
η	0.49	-	-
η	0.5	1.00	1.00
η	0.51	0.86	0.86
η	0.55	1.07	1.10
η	0.6	1.99	2.11
η	0.7	1.33	1.30
η	0.8	2.03	1.93
η	0.9	5.86	5.41
trisection			
for all boxes		1.14	0.87
once per direction		0.88	0.87

Table 2.1: Comparison of different subdivision strategies in box number and computation time

Table 2.1 shows that setting SubMinNew to a value greater than 2 can reduce the average computation time while the box number rises due to the higher number of subboxes per branching step. Reasonable explanation for the decrease of computation time would be that the smaller boxes can be discarded faster in the following bounding steps or that more gaps are exploited in subdivision and thus larger parts of a box are discarded within subdivision. However, the behavior for

the problems in the test set differs strongly. Thus no universal advice is given concerning the choice of parameter SubMinNew.

The individual test problems also behave differently with respect to the choice of the shift η. Especially the values for *Trigexp1* include significant outliers. It turned out that this problem is extremely sensitive with respect to subdivision (or more precisely with respect to the subboxes considered in the branch-and-bound algorithm). For $\eta = 0.49$, 0.4 and 0.6 the box number and computation time increase massively, causing an increased value for the average over the test set. (For $\eta = 0.49$ the computation time and box number of *Trigexp1* are omitted since the computation for this problem could not be finished within the given time limit of 36 hours. Further tests for this problem showed, that these values for η resulted in intervals with endpoints near to the single root of the problem. These could then not be discarded. We observed that the number of recursion level exceeded 250 for $\eta = 0.49$ and at least 4 boxes in each level.)

When ignoring outliers, the average computation time and box number behave as assumed: they increase when the subdivision point is shifted towards the boundaries of the subdivided interval. This is hardly surprising, as the box $\boldsymbol{x}$ is then subdivided into one comparatively small box and one box of only slightly smaller volume than $\boldsymbol{x}$, meaning the box size is not reduced adequately in the branch-and-bound algorithm. Hence more recursion levels have to be run and more subboxes are computed—causing an increase in computation time.

The results for single test problems show that shifted subdivision can be beneficial for the computation. For example, setting $\eta = 0.49$ the computational costs for problem *Brent7* dropped to one fourth compared to the equidistant subdivision. However, as for SubMinNew, no general strategy could be found to calibrate parameter η a priori.

Multisection too only shows improvement for specific problems, but not on average. For the test applying subdivision into three subboxes, the computation time of problem *Brent7* is reduced by a factor of 0.24 and for the problem *7erSystem* at least by a factor 0.56. For all other problems, the negative effects of a higher number of subboxes seem to be predominant resulting in increased computation time.

$\diamond$

Note again, that an appropriate subdivision is necessary for the termination of the branch-and-bound algorithm. We can only guarantee termination if every box is subdivided into one or more smaller subboxes.

2.3 Bounding / Contracting

This chapter introduces some contractors, symbolical as well as numerical ones, that help to reduce the number of subdivisions in the branch-and-bound algorithm—either by reducing boxes in volume or by discarding whole boxes. The invocation of any contractor in one of the following algorithms implies that its prerequisites have been checked and are fulfilled.

2.3.1 Constraint propagation

Constraint propagation (CP) is a symbolic contractor. Its basics seem to have been discovered more than once, in different areas of science. One often quoted origin are the works by van Hentenryck. An advantage of CP is that it does not require the considered functions to be continuous, differentiable or total. Furthermore, it is applicable to square and non-square systems.

The principle of constraint propagation is to attain sharper enclosures by rearranging constraints. The algorithm begins by resolving one constraint for one of the contained variables. Subsequently, a new enclosure for the variable is computed based on the new representation.

The procedure is repeated for changing combinations of variables and constraints. Thus information about the variables and function enclosures is *propagated* through the constraints. CP is terminated, as soon as the variables cannot be contracted any longer or are not contracted by more than a certain threshold value.

When rearranging constraints to yield enclosures of variables and function values, we have to deal with two basic problems. Firstly, we have to be able to resolve the constraints for specific variables. Secondly, the enclosure is required to be an interval.

Based on the reformulated constraint, it may be impossible to enclose the variable by an interval or a finite union of intervals. Take for example the simple constraint $\boldsymbol{x}_2 = \sin(\boldsymbol{x}_1)$. Resolving for x_1 yields $\boldsymbol{x}_1 = \sin^{-1}(\boldsymbol{x}_2)$ which may consist of an *infinite* union of intervals. Indeed, this problem has a simple solution. We just need to intersect the result of a resolved constraint with the previously known domain of the variables. Thus we attain a further contractor. For our example function resolving results in the new value $\boldsymbol{x}_1{}' = \boldsymbol{x}_1 \cap \sin^{-1}(\boldsymbol{x}_2)$. As a second example consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by

$$f(x) = {x_1}^2 + x_2.$$

If an enclosure $\boldsymbol{f}(\boldsymbol{x})$ for f is known, the enclosure for variable x_2 can be refined by calculating

$$\boldsymbol{x}_2{}' = \boldsymbol{x}_2 \cap \left(\boldsymbol{f}(\boldsymbol{x}) - {\boldsymbol{x}_1}^2\right).$$

The enclosures of $\boldsymbol{x}_2$ and $\boldsymbol{f}(\boldsymbol{x})$ can be contracted in the same manner using the updated domain of $\boldsymbol{x}_1$.

The issue of complicated expressions in the constraints is addressed by introducing additional variables to store intermediate results. Intermediate variables are introduced for every elementary function and every operator in the system. Yet only one variable is introduced for expressions that occur more than once. All intermediate variables are initialized with the interval $[-\infty, \infty]$. We end up with an *extended system* consisting of *elementary* constraints. We will consider the described extended system in some more detail in Chapter 3 under the name *fullsplit*. For a technical solution, a computational graph is constructed, representing the constraint system including the intermediate variables. The elementary constraints can now be resolved for the variables they contain.

Example 2.1
For our example function

$$f(x) = {x_1}^2 + x_2$$

an intermediate variable $x_3 = {x_1}^2$ *can be introduced and initialized with* $[-\infty, \infty]$. *We attain the new system*

$$\begin{aligned} x_3 &= {x_1}^2 \\ f(x) &= x_3 + x_2. \end{aligned}$$

In a first step we may compute a new enclosure for x_3 *by*

$$\boldsymbol{x_3}' = \boldsymbol{x}_3 \cap {\boldsymbol{x}_1}^2 = [-\infty, \infty] \cap {\boldsymbol{x}_1}^2,$$

or optionally by calculating

$$\boldsymbol{x_3}' = \boldsymbol{x}_3 \cap (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}_2)\,.$$

The first formula represents a *forward propagation step*, the second a *backward propagation step*. In general, the forward step propagates information about the variables to the function values, the backward step proceeds the other way around.

Note that for the running time of the algorithm the sequence in which the constraints and variables are considered is important. Our scheduling strategy to choose the next step in constraint propagation is given in [Wil04]. The next constraint for contraction is determined based on a priority assigned to the constraints. This priority depends on the relative changes of variables and the estimated costs to resolve constraints. The variable to consider next in constraint propagation is chosen to be "near" (in the computational graph) to the best contraction observed.

A more detailed presentation of CP can be found in most books concerning interval analysis (for example in [Kea96b] under the name of "substitution-iteration"). A nice, formal presentation can be found in [Wil04] in which Willems also allows to work with more general kinds of sets such as unions of intervals.

Furthermore, [Wil04] and [Bee06] discuss the issue of constraint propagation when there exist cycles in the computational graph.

In SONIC the computational graph is subdivided into acyclic subgraphs. They are considered as elementary constraints by constraint propagation. We can then not only resolve these constraints, but also apply *Taylor refinements* for some of them. We refer to this option as the internal usage of Taylor refinement in constraint propagation.

A method based on constraint propagation that offers a more effective sequence to consider the constraints is called *forward-backward propagation* and is, e.g., discussed in [JKDW01].

2.3.2 Taylor refinement

Taylor refinements can be applied to square and non-square systems. Like CP their purpose is to achieve sharper enclosures for variables by resolving constraints. Herein Taylor refinements make use of Taylor expansion and rearrangement of the constraints.

For the Taylor refinement of first order a once differentiable function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ is considered. A linearization is attained by the Taylor expansion of first order given by

$$f(\boldsymbol{x}) \subseteq f(c) + \sum_{i=1}^{n} \boldsymbol{s}_i \cdot (\boldsymbol{x}_i - c_i)$$

with some center $c \in \boldsymbol{x}$ and a slope vector $\boldsymbol{s}$. Now one can search for the solutions of the linear constraint

$$0 = f(c) + \sum_{i=1}^{n} \boldsymbol{s}_i \cdot (\boldsymbol{x}_i - c_i).$$

These solutions have to comprise all solutions of the (possibly nonlinear) constraint $f(x) = 0$ with $x \in \boldsymbol{x}$. A contractor can be computed by Algorithm 2.7. The algorithm is given with slopes, although also derivatives could be used instead.

Algorithm 2.7 Taylor refinement of first order (box $\boldsymbol{x}$)

1: **for** $i = 1, \ldots, n$ **do**

2: $$\boldsymbol{x}_i' \leftarrow \boldsymbol{x}_i \cap \left\{ c_i - \left(f(c) + \sum_{\substack{j=1 \\ j \neq i}}^{n} \boldsymbol{s}_j \cdot (\boldsymbol{x}_j - c_j) \right) \oslash \boldsymbol{s}_i \right\}$$

3: **end for**

Willems showed in [Wil04] that the computed enclosure does not get sharper if the $\boldsymbol{x}_j$ with $i < j$ in Algorithm 2.7 are replaced by the already computed $\boldsymbol{x}_i'$— unless the slope $\boldsymbol{s}$ is updated as well. (He also emphasizes correctly that we have

to use $[f(c)]$ instead of an evaluation of $f(c)$ to evade rounding errors. This is one of the many subtle sources of error when transferring interval analysis to machine numbers.)

For functions that are twice continuously differentiable on the box $\boldsymbol{x}$ we can further utilize the Taylor refinement of second order. The principle to be applied is the same, except that a Taylor expansion of order two is employed. However, some reformulations are helpful, see for example [BBLSA04, SA02] for details.

In SONIC we usually use only the Taylor refinement of first order, since the second-order refinement is computationally expensive.

2.3.3 Contraction by Taylor models

Also Taylor models can be used to create a contractor. Basically this method is just an evaluation of the function enclosure $\boldsymbol{f}_{TM}$ computed by Taylor models. As for the direct function evaluation test, the box is discarded if the function enclosure does not contain zero.

$$\mu_{TM}(\boldsymbol{x}) := \begin{cases} \emptyset & \text{for } 0 \notin \boldsymbol{f}_{TM}(\boldsymbol{x}) \\ \boldsymbol{x} & \text{else} \end{cases}$$

The disadvantage of Taylor models is their time-consumption. In general they cannot compensate this downside with sharper function enclosures.

Our first approach to the usage of Taylor models in SONIC was to integrate an existing C++ implementation (modul in RiOT [RiO] by Eble, described in [Ebl07]). However, this implementation can only handle eight variables in the Taylor models and does not include inverse Taylor models.

For these reasons we opted for a custom-built implementation of Taylor models in SONIC. A preliminary implementation was realized by Lars Balzer for his masters thesis [Bal13]. He also gave some test results for his Taylor models. His implementation uses the naive function enclosure to evaluate the polynomial part of the Taylor model could further be improved by techniques such as linear dominated bounder [Mak98]. It is further not yet optimized for performance and ,though it seems to work correctly for most functions, Balzer announced that the solution of one test problem is lost when applying the Taylor models. We thus did not include this method in our standard version of the solver.

Although, we conducted some further tests with those Taylor models to assess the yet available functionality and the potential for better enclosures. For those tests, the contractor based on Taylor models was included in our scheme for the application of contractors. We were especially interested in comparing the

strength of μ_{TM} with that of the second expensive contractor, the interval Newton method (described in the next section). Taylor models have thus been applied directly before the interval Newton method so this method could be skipped if the Taylor models could discard a box (an application scheme for the contractors follows in section 2.3.7, Algorithm 2.8). For our tests we considered spherical t-designs (see appendix) for $t = 3$ and different precision vectors ψ as it was done in [BLUW09]. The sobering results showed that the box number could not be decreased by the additional usage of Taylor models and that computation time increased dramatically.

Still, augmenting the first implementation we tested may improve these results. A first possibility to do so is to use techniques providing sharper enclosures of the function enclosure for Taylor models. Further, to the impression of the author, the utilization of inverse Taylor models [Hoe01] seems promising for the construction of contractors because they can benefit from their utilization of dependencies in a system.

2.3.4 Interval Newton method

The interval Newton method is a modification of the well known *Newton method* or *Newton-Gauß-Seidel method* for systems of equations with interval coefficients. The technique applies an *interval Newton operator* to contract intervals and boxes. Especially the preconditioned interval Newton method is an effective contractor.

To construct the interval Newton operator, we consider a box $\boldsymbol{x} \in {}^*\mathbb{IR}^n$ and a linearization of a function $f : \mathbb{R}^n \to \mathbb{R}^m$. Further needed is an interval matrix $\boldsymbol{S}$ containing slopes or derivatives for function f on $\boldsymbol{x}$ with respect to a predictor or initial guess point $c \in \boldsymbol{x}$. For setting c point near to a solution, a real-valued Newton method can be employed. We then can linearize to get

$$\boldsymbol{f}(\boldsymbol{x}) \subseteq f(c) + \boldsymbol{S}(\boldsymbol{x} - c)$$

as for Taylor refinements (and as before one has to calculate with the enclosure $[f(c)]$ in an implementation). For each solution x^* of $f(x) = 0$ with $x^* \in \boldsymbol{x}$ one can reformulate the above relation into

$$0 \in f(c) + \boldsymbol{S}(x^* - c)$$

and derive that x must be a solution of the corresponding *linear* system

$$-f(c) = \boldsymbol{S}(x^* - c) \tag{2.4}$$

as well. Thus every contractor of the linear system also has to be a contractor for the nonlinear system.

For solving the linear system (2.4), the Gauß-Seidel-approach $\mathcal{GS}$ presented in Algorithm 1.2 or the preconditioned version $\mathcal{PGS}$ given in Algorithm 1.3 can be utilized.

The evolving operator for the nonlinear system is called the *interval Newton operator*. It can be computed using slopes

$$\begin{aligned}\mathcal{N}(f, \boldsymbol{x}) &: \mathbb{IR}^n \to \mathbb{IR}^n \\ (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T &\mapsto \mathcal{GS}(\boldsymbol{S}, -f(c), \boldsymbol{x}, c).\end{aligned}$$

or enclosures of the derivatives

$$\begin{aligned}\mathcal{N}(f, \boldsymbol{x}) &: \mathbb{IR}^n \to \mathbb{IR}^n \\ (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T &\mapsto \mathcal{GS}(\boldsymbol{Df}(\boldsymbol{x}), -f(c), \boldsymbol{x}, c).\end{aligned}$$

A further advantage of the interval Newton method is that it does not only contract boxes, but can even verify the existence or uniqueness of solutions in given boxes. For verifying solutions we use a modified interval Newton method with operator $\mathcal{N}_{\text{mod}}(f, \boldsymbol{x})$. In principle, $\mathcal{N}_{\text{mod}}(f, \boldsymbol{x})$ is equal to the interval Newton operator when omitting the intersection with the original box $\boldsymbol{x}$ in the Gauß-Seidel method (Alg. 1.2, line 6), thus

$$\mathcal{N}(f, \boldsymbol{x}) \subseteq \mathcal{N}_{\text{mod}}(f, \boldsymbol{x}) \cap \boldsymbol{x}$$

holds and we can formulate the following theorem.

Theorem 2.1 (Properties of the modified interval Newton operator [Kea96b, SA02])
Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be continuous over a bounded box $\boldsymbol{x} \in \mathbb{IR}^n$ and $c \in \boldsymbol{x}$ and $\boldsymbol{S}$ be chosen as for the interval Newton method. Then

$$\mathcal{N}_{\text{mod}}(f, \boldsymbol{x}) \subseteq \boldsymbol{x}$$

implies that

(i) the slope matrix $\boldsymbol{S}$ is regular and

(ii) $\mathcal{N}_{\text{mod}}(f, \boldsymbol{x})$ (and therefore $\boldsymbol{x}$) contains a root of function f.

If $Df(\boldsymbol{x}) \subseteq \boldsymbol{S}$ holds additionally, the root in $\boldsymbol{x}$ is unique. If

$$\mathcal{N}_{\text{mod}}(f, \boldsymbol{x}) \cap \boldsymbol{x} = \emptyset$$

no root of f exists in $\boldsymbol{x}$.

In combination with the preconditioners described in section 2.3.5 the interval Newton method becomes crucial for the efficiency of most nonlinear interval solvers. Chapter 3 will revisit the interval Newton method. It introduces extended systems and presents strategies for applying the Newton method on the different systems. Further a row-wise application of the interval Newton method will be discussed.

2.3.5 Preconditioning

Preconditioners are used to modify functions. A preconditioner is a real-valued matrix which is multiplied from the left to a linear system of equations. It can be computed without using interval analysis.

For a given system of dimension $m \times n$

$$f(x) = A \cdot (x - \widetilde{x}) - b \tag{2.5}$$

the preconditioner C is element of $\mathbb{R}^{m \times m}$ and the preconditioned system can be formulated as

$$g(\boldsymbol{x}) = C \cdot f = (C \cdot A) \cdot (x - \widetilde{x}) - C \cdot b.$$

Lemma 2.1
For every system of the form (2.5) *and any preconditioning matrix* C*, the roots of the preconditioned function* $g = C \cdot f$ *comprise the roots of function* f*.*

Proof. Let x^* be a root of f. Then $A(x^* - \widetilde{x}) - b = 0$. Hence x^* is also a root of $C \cdot (A(x^* - \widetilde{x}) - b) = C \cdot A \cdot (x^* - \widetilde{x}) - C \cdot b$ and therefore a root of g. □

Lemma 2.2
For a system of the form (2.5) *and a regular preconditioning matrix* $C \in \mathbb{R}^{m \times m}$*, the given function* f *has the same roots as the preconditioned function* g*.*

Proof. The last lemma showed that each root of f is also a root of g. What is left to show is that also every root of g is also a root of f. We consider the equation

$$0 = g(x) = (C \cdot A) \cdot (x - \widetilde{x}) - C \cdot b = C \cdot (A \cdot (x - \widetilde{x}) - b).$$

Since C is regular we can conclude that

$$0 = A(x^* - \widetilde{x}) - b$$

and thus x^* is root of f. □

Lemma 2.3
Every contractor of the preconditioned function g *is also a valid contractor for function* f*.*

Proof. We need to consider both functions over the startbox $\boldsymbol{x}^{(0)}$. We already know that the set $\mathcal{S}_g$ of roots of g comprises $\mathcal{S}_f$, the set containing all roots of f. By definition, every contractor for g fulfills the conditions

$$\mu(\boldsymbol{x}) \subseteq \boldsymbol{x} \quad \text{for all } \boldsymbol{x} \subseteq \boldsymbol{x}^{(0)} \tag{2.6}$$

and

$$\mathcal{S}_g \cap \boldsymbol{x} = \mathcal{S}_g \cap \mu(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}.$$

To show that μ also is an contractor for f we simply have to show

$$\mathcal{S}_f \cap \boldsymbol{x} = \mathcal{S}_f \cap \mu(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}.$$

The inclusion $\mathcal{S}_f \cap \mu(\boldsymbol{x}) \subseteq \mathcal{S}_f \cap \boldsymbol{x}$ holds due to (2.6). For proving the reversed inclusion we consider an arbitrary box $\boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}$ and an arbitrary solution $x^* \in \mathcal{S}_f \cap \boldsymbol{x}$. Then clearly $x^* \in \mathcal{S}_f$ holds and $\mathcal{S}_f \cap \boldsymbol{x} \subseteq \mathcal{S}_g \cap \boldsymbol{x} = \mathcal{S}_g \cap \mu(\boldsymbol{x}) \subseteq \mu(\boldsymbol{x})$ yields $x^* \in \mu(\boldsymbol{x})$. Combining these results we get $x^* \in \mathcal{S}_f \cap \mu(\boldsymbol{x})$ and thus

$$\mathcal{S}_f \cap \boldsymbol{x} \subseteq \mathcal{S}_f \cap \mu(\boldsymbol{x}) \quad \text{for all } \boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}$$

(and thus the desired equality $\mathcal{S}_f \cap \boldsymbol{x} = \mathcal{S}_f \cap \mu(\boldsymbol{x})$). □

Thus, one can apply every matrix with suiting dimension as preconditioner and still have a contractor for f.

2.3.5.1 Row-wise preconditioners

One step of the preconditioned interval Gauß-Seidel method (Alg. 1.3) for the kth component of the linear system was formulated as

$$\boldsymbol{x}'_k = \widetilde{x}_k - \Bigg\{ -C_k\boldsymbol{b} + \sum_{j=1}^{k-1}(C_k\boldsymbol{A}_j)(\boldsymbol{x}'_j - \widetilde{x}_j) + \sum_{j=k+1}^{n}(C_k\boldsymbol{A}_j)(\boldsymbol{x}_j - \widetilde{x}_j) \Bigg\} \oslash (C_k\boldsymbol{A}_k) \tag{2.7}$$

As before, C_k denotes the kth row of a preconditioner C and $\boldsymbol{A}_j$ names the jth column of the interval matrix $\boldsymbol{A} \in {}^*\mathbb{IR}^{m\times n}$.

Note that for the computation of $\boldsymbol{x}'_k$ only the kth row of the preconditioner is needed. Thus, if we do not compute all $\boldsymbol{x}'_k$, we may save computational effort by just computing the needed preconditioner rows C_k. This is possible for preconditioners that can be computed row-wise, called *row-wise preconditioners*. The preconditioner for which we compute a certain preconditioner row for the interval Gauß-Seidel method can then even be chosen individually for each variable.

Helpful preconditioners can be constructed considering the influence the variables of a system have on the function enclosures. Preconditioners aim to transform the system such that every variable i is most influential in the preconditioned function component g_i. This provides benefits, e.g., for the interval Gauß-Seidel method.

2.3.5.2 A simple preconditioner

A class of very simple preconditioners are "pivot preconditioners". They only contain one entry different from zero per row C_i, all other entries are zero. For each component of a function f_i this entry marks the one variable with the largest influence on the function enclosure for f_i. One special pivot preconditioner is described by Stadtherr in [LS03]. Its nonzero entries are equal to 1.

However, pivot preconditioners have the disadvantage that each component of the preconditioned function g is only related to one component of function f. This can be changed by allowing more than one nonzero entry per row.

2.3.5.3 The inverse midpoint preconditioner

A widely known and successfully applied preconditioner (given a differentiable function) is the inverse midpoint preconditioner. Unlike pivot preconditioners, it replaces a single function component by a linear combination of several function components.

The inverse midpoint preconditioner of an interval system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is computed as

$$C_{\text{InvMid}} := (\text{mid}\,(\boldsymbol{A}))^{-1}.$$

Note that the existence of the inverse matrix is required. In general this is only possible for square matrices $\boldsymbol{A}$ and thus only for square systems. There exists a related approach that allows to compute row-wise preconditioners for non-square systems as well (see [Bee06, p. 38]).

For nonlinear systems one often calculates with $\boldsymbol{A} = \boldsymbol{Df}(\boldsymbol{x})$ which occurs in linearizations. In this case $\boldsymbol{Df}(\boldsymbol{x})$ is a measure for the influence the variables have on the function enclosures. For rigorous calculations one has to keep in mind that only an approximation of the inverse midpoint preconditioner is computed.

2.3.5.4 Optimal preconditioners

It was shown that for the interval Gauß-Seidel method there exist even better preconditioners than the inverse midpoint preconditioner [Kea90]. Even more, formulae have been developed to compute preconditioners which provide optimality with respect to several criteria for $\boldsymbol{x}'_i$ as computed by the Gauß-Seidel method. (Note that this optimality properties cannot necessarily be transferred to the result of the intersection of $\boldsymbol{x}'_i \cap \boldsymbol{x}_i$.)

Like the inverse midpoint preconditioner, these preconditioners typically have more than one nonzero entry per row C_k. Thus the application of the preconditioner results in the replacement of one function component by a linear combination of function components.

For the further discussion the numerator and denominator of the relational division in equation (2.7) are shortened by

$$\boldsymbol{n}_k(C_k) := -C_k\boldsymbol{b} + \sum_{j=1}^{k-1}(C_k\boldsymbol{A}_j)(\boldsymbol{x}'_j - \widetilde{x}_j) + \sum_{j=k+1}^{n}(C_k\boldsymbol{A}_j)(\boldsymbol{x}_j - \widetilde{x}_j)$$

and $\boldsymbol{d}_k(C_k) := C_k\boldsymbol{A}_k$.

Thus the formula for $\boldsymbol{x}'_k$ is simplified to

$$\begin{aligned}\boldsymbol{x}'_k &= \widetilde{x}_k - \boldsymbol{n}_k(C_k) \oslash \boldsymbol{d}_k(C_k)\\ &= \widetilde{x}_k - [\inf \boldsymbol{n}_k(C_k), \sup \boldsymbol{n}_k(C_k)] \oslash [\inf \boldsymbol{d}_k(C_k), \sup \boldsymbol{d}_k(C_k)]\end{aligned}$$

In [Nov93] Novoa presents and categorizes row-wise applicable preconditioners which are defined to be optimal in a predefined sense. His first distinction is made with respect to $\boldsymbol{d}_k$.

Definition 2.4
The preconditioner row C_k is called a C-preconditioner *if*

$$0 \notin \boldsymbol{d}_k(C_k).$$

It is furthermore called normal *if*

$$\underline{d_k}(C_k) = 1.$$

The "C" in the name is based on the contracting property of C_k. (Because $\boldsymbol{d}_k(C_k)$ does not contain zero, $\boldsymbol{x}'_k$ is a single, connected interval and $\boldsymbol{x}'_k \cap \boldsymbol{x}_k \subseteq \boldsymbol{x}_k$.)

Novoa further introduced a second class of preconditioners for which the conditions $0 \in \boldsymbol{d}_k(C_k)$ and $0 \notin \boldsymbol{n}_k(C_k)$ hold and the relational division of $\boldsymbol{n}_k(C_k)$ by $\boldsymbol{d}_k(C_k)$ thus yields two disjoint intervals. Since the result is "split" into two parts, the preconditioners in this class are called S-*preconditioners* (sometimes also E-preconditioners). S-preconditioners can be useful to separate multiple solutions in a box. However, this work concentrates on C-preconditioners.

Note that the case $0 \in \boldsymbol{d}_k(C_k)$ and $0 \in \boldsymbol{n}_k(C_k)$ is neither covered by C- nor S-preconditioners because this would yield $\boldsymbol{d}_k(C_k) \oslash \boldsymbol{n}_k(C_k) = [-\infty, \infty]$ and no contraction could be attained in the interval Gauß-Seidel method.

The existence of a C-preconditioner can be proved considering column $\boldsymbol{A}_k$ of the interval matrix $\boldsymbol{A}$.

Theorem 2.2 (Existence of a C-preconditioner [Kea96b])
A C*-preconditioner exists if and only if at least one element of $\boldsymbol{A}_k$ does not contain zero.*

Furthermore, preconditioner rows can be defined as equivalent.

Definition 2.5
The preconditioner rows $C_k^{(1)}$ *and* $C_k^{(2)}$ *are called* equivalent, *if there exists a real number* $\rho \in \mathbb{R}\backslash\{0\}$ *such that*

$$C_k^{(1)} = \rho \cdot C_k^{(2)}$$

holds.

Lemma 2.4
For every C*-preconditioner row* $C_k^{(1)}$ *there exists an equivalent normal* C*-preconditioner row* $C_k^{(2)}$.

Proof. Since $C_k^{(1)}$ is a C-preconditioner, $\boldsymbol{d}_k(C_k^{(1)})$ does not contain zero and a normal preconditioner $C_k^{(2)}$ can be constructed by

$$C_k^{(2)} := \begin{cases} C_k^{(2)} := \frac{1}{\overline{d_k}(C_k^{(1)})} C_k^{(1)} & \text{if } \overline{d_k} < 0 \\ C_k^{(2)} := \frac{1}{\underline{d_k}(C_k^{(1)})} C_k^{(1)} & \text{if } \underline{d_k} > 0 \end{cases}.$$

$C_k^{(2)}$ is obviously equivalent to $C_k^{(1)}$ and normal. □

Lemma 2.5
The interval Gauß-Seidel method computes the same enclosures for equivalent preconditioner rows.

Proof. Assume we have given two equivalent preconditioner rows $C_k^{(1)}$ and $C_k^{(2)}$ with $C_k^{(2)} = \rho \cdot C_k^{(1)}$ and $\rho \in \mathbb{R}\backslash\{0\}$. Then we can reformulate

$$\begin{aligned} \boldsymbol{n}_k(C_k^{(2)}) \oslash \boldsymbol{d}_k(C_k^{(2)}) &= \boldsymbol{n}_k(\rho \cdot C_k^{(1)}) \oslash \boldsymbol{d}_k(\rho \cdot C_k^{(1)}) \\ &= \left(\rho \cdot \boldsymbol{n}_k(C_k^{(1)})\right) \oslash \left(\rho \cdot \boldsymbol{d}_k(C_k^{(1)})\right) \\ &= \boldsymbol{n}_k(C_k^{(1)}) \oslash \boldsymbol{d}_k(C_k^{(1)}). \end{aligned}$$

Thus the enclosures of the interval Gauß-Seidel method have to coincide for arbitrary k. □

We proceed by introducing the criteria for optimality of a C-preconditioner following the works of Novoa and Kearfott [Nov93, Kea96b]. Due to Lemma 2.4 the following considerations can be restricted to normal C-preconditioners.

Definition 2.6
Let $\boldsymbol{x}'_k$ be the box yielded by the interval Gauß-Seidel method. A C*-preconditioner row C_k is called a*

- C^{W}-preconditioner*, if it minimizes* width $(\boldsymbol{x}'_k)$
- C^{L}-preconditioner*, if it maximizes* $\inf(\boldsymbol{x}'_k)$
- C^{R}-preconditioner*, if it minimizes* $\sup(\boldsymbol{x}'_k)$
  and
- C^{M}-preconditioner*, if it minimizes the magnitude* $|\boldsymbol{x}'_k - \widetilde{x}_k|$

over all C*-preconditioners.*

Considering the spectrum of categories, only one will be interesting for us: width optimal C^{W}-preconditioner. In [KS96] Kearfott speaks about the application of the different preconditioners. He suggests to use C^{W}-preconditioners for finding solutions of nonlinear systems and C^{M}-preconditioners for verifying solutions. (S-preconditioners are suggested as appropriate to prove that no solution exists in a given box.)

Each of the optimal preconditioners is the solution of a nonlinear optimization problem. Novoa showed that one does not even need to solve this nonlinear problem, but that the optimization problem can be reformulated as a linear problem. The formulation of the linear optimization problem for the C^{W}-preconditioner (and its derivation) can be found both in [Kea96b] and [Bee06].

Note that the solution of this optimization problem may differ from the C^{W}-preconditioner. As in [Kea96b], we speak of the C^{W}*-LP-preconditioner* for the computed solution of the linear optimization problem for the C^{W}-preconditioner. For $0 \in \boldsymbol{n}_k(C_k)$ the numerical results likewise only give an approximation of the C^{W}-LP-preconditioner, e.g., because only a fixed number of iterations are conducted in a simplex method to solve the optimization problem.

Further the optimization problem requires the condition $0 \in \boldsymbol{n}_k(C_k)$ to be fulfilled to yield an optimal preconditioner. This condition cannot be checked in advance. But even if $0 \notin \boldsymbol{n}_k(C_k)$ for the resulting C^{W}-LP-preconditioner, it still provides significant contraction for variable $\boldsymbol{x}_k$ in the sense that

$$\text{width}\,(\boldsymbol{x}'_k \cap \boldsymbol{x}_k) \leq \frac{1}{2} \cdot \text{width}\,(\boldsymbol{x}_k)$$

holds true for every C-preconditioner with $0 \notin \boldsymbol{n}_k(C_k)$ [Kea90].

2.3.5.5 The C^{W}-LP-preconditioner

Within the development of SONIC different approaches were made to compute the C^{W}-LP-preconditioner. For the time being, the default solving method is a C++ version of an GlobSol simplex algorithm.

Another way to compute a C^W-LP-preconditioner is to construct the optimization problem and to hand it over to an arbitrary optimizer for real-valued optimization problems. There are multiple programs available for this task. Some of them make use of the structure of the optimization problem or of sparse data structures. The usage of sparse data structures gains additional importance in combination with systems that contain many variables—as the extended systems presented in Chapter 3. These systems often contain a large number of variables and constraints, but each constraint depends only on a handful of variables. Thus the Jacobian matrix contains few entries per row and can be saved as a sparse matrix.

The extern optimizers for solving linear problems that can be started within SONIC are Galahad and GLPK. However, using them implies downsides. Either the solvers do not solve all problems and abort with errors or they are not thread-safe and can thus not be used in parallel computations (cf. section 5.4).

Some more optimizers (namely CLP, lp_solve and qsopt) have recently been investigated by Balzer in [Bal10]. In parts these optimizers also have the problem of stopping due to errors. Moreover, relating to the computation time needed in SONIC, they still are inferior to the previously used routine. Thus the GlobSol version stays the default linear optimizer in SONIC.

Despite the usage of other external optimizers, the author sees room for further improvements with respect to the computation of the C^W-LP-preconditioner in SONIC. A goal for upcoming work is to reformulate the computation based on GlobSol to sparse data structures. Also the usage of fast routines for vector and matrix operations may provide significant benefit for this implementation.

A reformulation of the current implementation, which we tested following a suggestion by B. Lang, was to change the computation sequence of rows and columns for matrix operations. The purpose of that reformulation is to make better use of the cache. However, our test problems are usually too small to see a difference in computation time. (For the extended systems presented in the next chapter the C^W-LP-preconditioner is not applied for large extended systems in the default setting due to step size $\sigma_{C^W}(\mathcal{E})$.) Yet, the reducing influence on the computation time becomes obvious if we force our solver to compute the C^W-LP-preconditioner for all variables in large systems. But we also sas that for some problems the computation on a single box can consume more time than our default approach needs to consider several thousand boxes and to solve the whole problem.

Another point worth further consideration is an appropriate choice of the parameter δ and the iteration number in the simplex algorithm used for the computation of the C^W-LP-preconditioner. Again a trade-off has to be found between more iterations, providing better preconditioners, and fewer iterations saving computation time.

2.3.6 Application scheme for contractors

For general problems, one cannot decide a priori which contractors provide the sharper enclosures. This decision is yet impossible if considering just the two most important contractors, constraint propagation and the interval Newton method. Although there seems to exist a general consensus in the interval community that constraint propagation is more appropriate for boxes with "larger" width and the Newton method should be preferred for "small" intervals, nobody can give a universal a priori decision when to switch the applied contractor.

In the context of this work some tests were conducted, showing that a simple switching from constraint propagation to the Newton method does not provide uniform improvement. For that purpose the two contractors were applied depending on the width of the considered boxes. If the width of a box has been larger than a prescribed threshold 10^ν in every direction, only CP was applied. As soon as width $(\boldsymbol{x}_i)$ became smaller than 10^ν for some $i \in \{1, \ldots, n\}$, the Newton method was the only contractor applied for the box. The predefined integer $\nu \in \mathbb{N}$ was varied from -7 up to 1.

We found that there exists no value for ν suited equally well for all problems, but that the computation time for individual problems was smallest for different values of ν. As examples we address *Trigonometric* and *min-04-07*. While for *Trigonometric* $\nu = 1$ yielded the best computation time for our simple test, $\nu = -1$ was a good choice for *min-04-07*.

2.3.7 Application of contractors in SONIC

In SONIC a general strategy to apply the contractors is pursued. It is summarized in Algorithm 2.8. (Naturally, the requirements of the contractors, as continuity or differentiability, are checked implicitly before the methods are applied.) Note that CP is applied repeatedly following the other contractors. This is done to propagate contractions in single variables through the whole constraint system.

Experience has taught us that good average timings can be achieved by disabling the Taylor refinement of second order within constraint propagation as well as separate Taylor refinements of first and second order. However, activating more contractors can reduce the average box number. For individual problems they also reduce the computation time considerably. For example, using a Taylor refinement of first order within CP reduces the computation time needed for problem *Trigonometric* by nearly 60% (and the box number by 50%).

$\diamond$

For easier denomination of the relations between boxes in the branch-and-bound algorithm some definitions of Chapter 1 are expanded.

Algorithm 2.8 Application scheme for contractors (box $\boldsymbol{x}$)

1: if enabled: apply CP {can use Taylor refinement of first and second order internally}
2: if box $\boldsymbol{x}$ cannot contain a solution: discard $\boldsymbol{x}$
3: **if** Taylor refinement of first order is enabled **then**
4: apply Taylor refinement of first order (Alg. 2.7)
5: if enabled: apply CP
6: **end if**
7: if box cannot contain a solution: discard
8: **if** Taylor refinement of second order is enabled **then**
9: apply Taylor refinement of second order
10: if enabled: apply CP
11: **end if**
12: if box $\boldsymbol{x}$ cannot contain a solution: discard $\boldsymbol{x}$
13: **if** interval Newton method is enabled **then**
14: apply hybrid Newton method (Alg. 3.2)
15: if enabled: apply CP
16: **end if**
17: if box $\boldsymbol{x}$ cannot contain a solution: discard $\boldsymbol{x}$

We furthermore speak of $\boldsymbol{x}$ as a childbox of $\boldsymbol{y}$ if it is derived from $\boldsymbol{y}$ by one branching step and the application of contractors ($\boldsymbol{y} \supseteq \boldsymbol{x}$ and $\mathrm{rl}(\boldsymbol{y}) + 1 = \mathrm{rl}(\boldsymbol{x})$). In analogous fashion, the box $\boldsymbol{y}$ will be referred to as the parentbox of $\boldsymbol{x}$ if it was the last superbox of $\boldsymbol{x}$ regarded in the branch-and-bound algorithm. In the same way we speak of $\boldsymbol{y}$ as an ancestorbox if $\boldsymbol{y}$ is a superbox of $\boldsymbol{x}$ and the recursion level of $\boldsymbol{x}$ is larger than that of $\boldsymbol{y}$ ($\boldsymbol{y} \supseteq \boldsymbol{x}$ and $\mathrm{rl}(\boldsymbol{y}) > \mathrm{rl}(\boldsymbol{x})$).

2.4 Operating sequence for the branch-and-bound algorithm

All choices in the branching as well as the bounding step of the branch-and-bound algorithm result in different overall computation time and box number. Figuratively speaking, the configuration of the methods determines which branch-and-bound tree is build. However, until now, we have not discussed, how to process through the boxes in a given branch-and-bound tree.

This sequence depends on how boxes are inserted into and picked and deleted from the working list L_w. If, for example, we always take the first box from the list and attach each new box to its end, we traverse the branch-and-bound tree breadth-first.

If we are only interested in whether the system has any solution, huge amounts of computation time can be saved by stopping the computation as soon as a single

solution box is found and verified to indeed contain a solution of the system. (The option to do so in SONIC is set by parameter *TerminationOnFirstHit*. Methods for the verification of solutions are discussed in Chapter 4.) For finding the solution boxes faster, one can adjust the order of elements in the working list. This can be done by assigning a rank to each box and processing the boxes in the working list according to their rank. The rank is picked to relate to the probability for a box to contain a solution [Bee06]. We start to search for solutions in the most promising box.

Furthermore, the data structure for L_w can be handled both as list and as a heap.

In SONIC boxes can be stored in a heap or list. Three box ordering strategies can be chosen. A short presentation and comparison of these strategies can be found in [Bee06, p. 67].

◇

Note again that in the following we are mainly interested in reducing the average computation time for the overall computation of the branch-and-bound algorithm. A reduction of the box number is just an auxiliary goal to lower the computation time. If one is interested in a small average box number, one would simply apply all available contractors in the bounding step of the branch-and-bound algorithm and choose an appropriate subdivision strategy.

Chapter 3

Extended systems

I cannot say whether things will get better if we change; what I can say is they must change if they are to get better.

Georg C. Lichtenberg

The main idea to be presented in this chapter is that contractors cannot only be applied to the originally given system of equations. They can be even more helpful if applied to *extended systems*. Those are modified nonlinear systems derived from the original system by adding intermediate variables for subterms. Extended systems have simpler, "extended" constraints, but still represent the same system. We will focus on strategies for employing these systems efficiently.

The utilization of different *extended systems* is one of the features making SONIC a powerful solver for nonlinear systems of equations. It further is an advantage over other solvers that do not apply contractors on these systems.

The systems implemented in SONIC have been described by Willems in [Wil04] under the name of *split levels* or *splits*. He used this denomination because the new variables allow the "splitting" of constraints into more simple relations. (The name "split" remains in use for parameters in the implementation for reasons of consistency with earlier publications.)

3.1 Basics

The following section is dedicated to extended systems and the contractors used on them. All of the following extended systems have been named, defined formally and described in detail in [Wil04].

3.1.1 Introduction of extended systems

As functions can be represented by different expressions, a given system of equations of functions $f_1, \ldots, f_m$ in variables $x_1, \ldots, x_n$ can be formulated with the help of different sets of constraints. An extended system is a representation of a given system in which intermediate variables are introduced together with constraints defining them. In general, the first additional variable is called x_{n+1}. It is identified with a term τ and each occurrence of this term is replaced by x_{n+1}. The following variables $x_{n+2}, x_{n+3}, \ldots$ are introduced in the same manner.

Every extended systems contains at least the variables of the originally given system. Depending on the parts of a constraint system that are represented by intermediate variables, different extended systems are constructed.

The box we consider is *not* augmented by intermediate variables. Consequently every volume we speak about is computed for $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$.

Definition 3.1
For each system of equations $\mathcal{E}$, the set $\mathrm{Var}(\mathcal{E})$ *is defined as the set of all variables in $\mathcal{E}$. The* number of variables *in is denoted by* $|\mathrm{Var}(\mathcal{E})|$.

An extended system $\mathcal{E}^{(1)}$ is called finer *than another system $\mathcal{E}^{(2)}$ if* $|\mathrm{Var}(\mathcal{E}^{(1)})| > |\mathrm{Var}(\mathcal{E}^{(2)})|$. *System $\mathcal{E}^{(2)}$ is then called* coarser *than $\mathcal{E}^{(1)}$.*

Note that this definition does not imply that a finer system $\mathcal{E}^{(1)}$ contains all the variables of a coarser system $\mathcal{E}^{(2)}$ (so even if $|\mathrm{Var}(\mathcal{E}^{(1)})| > |\mathrm{Var}(\mathcal{E}^{(2)})|$ holds, the inclusion $\mathrm{Var}(\mathcal{E}^{(1)}) \supset \mathrm{Var}(\mathcal{E}^{(2)})$ does not have to be true).

Definition 3.2 (Hierarchy of extended systems)
A hierarchy of extended systems *is a list containing different extended systems in coarse to fine order. It is denoted by $\mathcal{H}_{\mathcal{E}}$. The coarsest system of the hierarchy is denoted as $\mathcal{E}_{\mathrm{coarsest}}$, the finest as $\mathcal{E}_{\mathrm{finest}}$.*

When the extended systems are built for a special problem and two extended systems contain the same variables, one of them is discarded from the hierarchy.

After all intermediate variables are introduced, a renumbering of the variables is enforced. Thus we attain a numbering in which the constraints for intermediate variables only depend on variables with smaller index. These dependencies can be used when applying contractors.

Original system

The given system, more precisely the system without any additional variables, is called the *original system* and denoted by $\mathcal{E}_{\text{orig}}$. We also speak of the *original variables* for the elements of $\text{Var}(\mathcal{E}_{\text{orig}})$.

In SONIC the original system may still differ from the system in the problem file due to expression optimization. Due to implementation issues, SONIC is bound to elementary operators. All systems are constructed once, prior to any computations on the systems.

Fullsplit system

The *fullsplit system* $\mathcal{E}_{\text{full}}$ was described before by Kearfott [Kea96b, Kea91] and is also applied in other solvers such as GlobSol [Kea09]. To obtain the fullsplit system, we start with the original system and introduce intermediate variables until only elementary constraints are left.

Any subset of $\text{Var}(\mathcal{E}_{\text{full}})$ containing at least the variables of the original system $\text{Var}(\mathcal{E}_{\text{orig}})$ defines an extended system. The following three systems have been developed by Willems [Wil04] for the usage in SONIC. In the same work, Willems analyzed additional extended systems that turned out as not promising and are not addressed within this work.

CST system

The name of the *CST system* $\mathcal{E}_{\text{cst}}$ is an abbreviation of **C**ommon **S**ub**T**erms. As the name suggests, intermediate variables are added for the "largest" subterms with multiple occurrences in the original system. This choice of additional variables has the advantage of reducing the number of identical entries in the Jacobian matrix.

How common subterms are found and handled is discussed in [Wil04]. In principle, subterms are processed from smaller to larger ones. For unary operators just the operation and the operand need to be compared. If two subterms τ_1 and τ_2 occur more than once and are connected by binary operators, one checks whether there can be found an even larger common subterm. For that purpose, the connecting operator is considered. If the operator is the same (Φ) and connects τ_1 and τ_2 in the same order, then $\Phi(\tau_1, \tau_2)$ is itself a subterm with multiple occurrences. If Φ represents an addition or multiplication, it is commutative and $\Phi(\tau_1, \tau_2)$ and $\Phi(\tau_2, \tau_1)$ are considered as the same term. There are also other approaches to use multiple occurrences in constraints, see for example [ANT12].

CSTN system

A further extended system is called the *CSTN system* $\mathcal{E}_{\text{cstn}}$ (short for **C**ommon **S**ub**T**erms and **N**eighbors). Like the CST system, it comprises the "largest" subterms with multiple occurrences. In addition, $\mathcal{E}_{\text{cstn}}$ contains intermediate variables for their "siblings" and "parents" in the fullsplit system as defined by Willems [Wil04]. This means that, if a common subterm (or multiply used original variable) τ_1 is found and it is connected to another term τ by $\tau = \Phi(\tau_1)$ in the fullsplit system, then not only

$$x_{n+1} = \tau_1$$

is introduced as for the CST system, but also the additional constraint

$$x_{n+2} = \Phi(x_{n+1}).$$

If $\tau = \Phi(\tau_1, \tau_2)$ the added constraints are

$$\begin{aligned} x_{n+1} &= \tau_1 \\ x_{n+2} &= \tau_2 \quad \text{and} \\ x_{n+3} &= \Phi(x_{n+1}, x_{n+2}). \end{aligned}$$

This is done for all common subterms.

Linear system

One advantage of extended systems is their "reduced nonlinearity". To promote this property one can define intermediate variables just for those subterms in the original system that are neither atomic nor linear. Although, the system still contains constraints that "concentrate" nonlinear operations, Willems called this system the *linear extended system* or *linear system* $\mathcal{E}_{\text{lin}}$.

Example 3.1
We show the extended systems and computational graphs for the original system $\mathcal{E}_{\text{orig}}$

$$\begin{aligned} 0 &= f_1 = x_1^2 - \exp(x_2) \\ 0 &= f_2 = x_1^2 + x_2 \cdot x_3. \end{aligned} \tag{3.1}$$

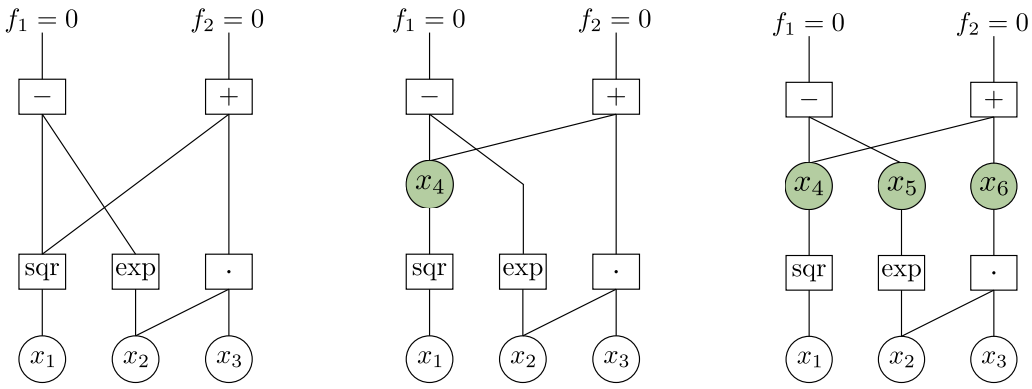
In the CST system $\mathcal{E}_{\text{cst}}$ *an intermediate variable is introduced for the common subterm* x_1^2 *yielding the system*

$$\begin{aligned} 0 &= x_4 - \exp(x_2) \\ 0 &= x_4 + x_2 \cdot x_3 \\ x_4 &= x_1^2. \end{aligned}$$

For the fullsplit system $\mathcal{E}_{\text{full}}$ *further variables are introduced for* $\exp(x_2)$ *and* $x_2 \cdot x_3$.

$$\begin{aligned} 0 &= x_4 - x_5 \\ 0 &= x_4 + x_6 \\ x_4 &= x_1^2 \\ x_5 &= \exp(x_2) \\ x_6 &= x_2 \cdot x_3 \end{aligned}$$

Figure 3.1 shows the computational graphs for the three extended systems. An example including $\mathcal{E}_{\text{cstn}}$ *and* $\mathcal{E}_{\text{lin}}$ *extended systems can, e.g., be found in [JL12].*



(a) $\mathcal{E}_{\text{orig}}$ (b) $\mathcal{E}_{\text{cst}}$ (c) $\mathcal{E}_{\text{full}}$

Figure 3.1: Computational graph for three extended systems of system (3.1) (Intermediate variables are depicted with colored background.)

For the hierarchy of extended systems $\mathcal{H}_{\mathcal{E}}$ it is usually sensible to choose $\mathcal{E}_{\text{coarsest}}$ as the original system and $\mathcal{E}_{\text{finest}}$ as the fullsplit system. Our default hierarchy of extended systems is

$$\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}}).$$

3.1.2 Contractors on extended systems

The contractors presented in section 2.3 can also be applied on extended systems. For shorter notations the author will often speak of the "choice and application of an extended system" instead of the "choice and application of contractors for an extended system". Accordingly other technical terms like the success of contractors are transferred to extended systems (still, mathematically they refer to the contractors on the system in question).

3.1.2.1 Constraint propagation

Constraint propagation is always applied on the fullsplit system. It thus updates all variables, including the intermediate variables in the extended systems. Constraint propagation can be interpreted as "connecting" the different extended systems, because it is applied after the contractors are applied on the separate extended systems,.

3.1.2.2 Extended Newton method

Since the interval Newton method is only formulated for finding zeros of a given function, the extended systems have to be modified before the interval Newton operator can be applied. Assume we have given a function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ and an extended system for this function

$$\mathcal{E} := \begin{cases} 0 & = \Phi_1 \\ \vdots & \\ 0 & = \Phi_m \\ x_{n+1} & = \Phi_{m+1} \\ \vdots & \\ x_N & = \Phi_M \end{cases}$$

comprising $N \in \mathbb{N}$ variables and $M \in \mathbb{N}$ elementary functions $\Phi_1, \ldots, \Phi_M$ (with $N - n = M - m$). This system is modified into a system $\mathcal{E}'$ and defines the modified function $f_{\mathcal{E}'}$ by

$$\mathcal{E}' := \begin{cases} 0 & = \Phi_1 \\ \vdots & \\ 0 & = \Phi_m \\ 0 & = x_{n+1} - \Phi_{m+1} \\ \vdots & \\ 0 & = x_N - \Phi_M \end{cases} \quad \text{and} \quad f_{\mathcal{E}'} := \begin{cases} \Phi_1 \\ \vdots \\ \Phi_m \\ x_{n+1} - \Phi_{m+1} \\ \vdots \\ x_N - \Phi_M \end{cases} .$$

Obviously, the zeros of function $f_{\mathcal{E}'}$ in a box $\boldsymbol{x}$ are the same as for function f and equal to the solutions of the systems $\mathcal{E}$ and $\mathcal{E}'$ in box $\boldsymbol{x}$. The zeros of $f_{\mathcal{E}'}$ can now be determined by applying the Newton method to it (row-wise for non-square systems).

Note that all variables in the extended system are contracted by the interval Newton operator, not only the variables in the original system we are mainly interested in. Furthermore, the variables of the extended systems have to be initialized before running the Newton method. This is done by a run of CP. To

propagate contractions in the variables, CP is used again after the Newton method was applied to function $f_{\mathcal{E}'}$. The application of the interval Newton operator in combination with CP, as stated in Algorithm 3.1, is referred to as the interval Newton method on an extended system or simply as the *extended Newton method*. The expression "application of the extended Newton method" always refers to the application of the Newton method to the modified system $f_{\mathcal{E}'} = 0$.

Algorithm 3.1 EXTENDED NEWTON METHOD (box $\boldsymbol{x}$, system $\mathcal{E}$)

1: update all variables by CP on box $\boldsymbol{x}$
2: apply Newton method to function $f_{\mathcal{E}'}$ on box $\boldsymbol{x}$
3: update all variables by CP on box $\boldsymbol{x}$

The complexity of each step of the extended Newton method is given as $\mathcal{O}(|\mathrm{Var}(\mathcal{E})|^3)$ [Wil04, p. 100]. Thus using it on extended systems $\mathcal{E}$ with large $|\mathrm{Var}(\mathcal{E})|$ can be way more expensive than applying the Newton method on $\mathcal{E}_{\mathrm{orig}}$. Nevertheless, there exist good reasons to apply the Newton method on extended systems. Primarily, the Newton method is assessed to be more powerful on finer extended systems because these systems are less complex in structure and "more linear" than the original system. As a ramification, the linearization in the Newton method should be more efficient. In addition, the Jacobian matrix of an extended system can be sparse and the derivatives can be computed exactly more often [Kea91] [Bee06, p. 49] [Wil04, p. 101]. Numerical validation for the usefulness of extended systems follow within this chapter. For computations one can choose between the different systems and their respective advantages concerning contractors.

3.1.2.3 Hybrid Newton method

The hybrid Newton method was presented by Beelitz [Bee06]. It works row-wise and controls, for which rows or constraints the extended Newton method is applied and which preconditioners are used. (Since we solve the *ith* constraint for the *ith* variable within the extended Newton method we will speak as well of "contracting variable i" as of the constraint i we are working on.) The general workflow of the hybrid Newton method is given in Algorithm 3.2.

As the extended systems, the preconditioners to be applied are ordered in a hierarchy $\mathcal{H}_{\mathrm{precond}}$. They are ordered by strength (as well as computational costs), the most promising preconditioner is the first in the hierarchy. Currently the hierarchy contains 4 elements: the C^{W}-LP-preconditioner ($C^{(1)}$), the inverse midpoint preconditioner ($C^{(2)}$), the simple pivot preconditioner ($C^{(3)}$) and the identity matrix ($C^{(4)}$) representing a computation without preconditioning. All preconditioners are computed and applied row-wise.

There exist problems for which another sequence of preconditioners or the selection of fewer preconditioners in the hierarchy showed to be advantageous.

For example, the computation time for problem *Reactor* could be reduced to one third when not applying the C^W-LP-preconditioner. On the other hand, the computation time for *DirectKinematics* rises by a factor 4 with this option. The hierarchy of the preconditioners was elected (in earlier works) to provide good average results and is based on experience and numerical tests.

A special remark needs to be made regarding the variables considered in the extended systems. Willems introduced two *step sizes* for each system. The step sizes are used to skip variables in large systems. The internal numbering of variables needed for applying a step size is done in the fullsplit system. However, the variables in the original system are always considered since our main interest is to contract them.

The first step size σ_{var} is used to choose the variables to be considered in the extended Newton method. The second step size σ_{C^W} is used to regulate the number of variables for which the computationally expensive C^W-LP-preconditioner (see section 2.3.5.4) is computed. The step sizes are defined as

$$\sigma_{var}(\mathcal{E}) := \begin{cases} 1 & \text{if } |\mathrm{Var}(\mathcal{E})| \leq 50 \\ \max\{|\mathrm{Var}(\mathcal{E}_{\mathrm{orig}})|, \lfloor 0.2 \cdot |\mathrm{Var}(\mathcal{E})| \rfloor\} & \text{else} \end{cases}$$

and

$$\sigma_{C^W}(\mathcal{E}) := \begin{cases} 1 & \text{if } |\mathrm{Var}(\mathcal{E})| \leq 10 \text{ or } \mathcal{E} = \mathcal{E}_{\mathrm{orig}} \\ 2 & \text{if } |\mathrm{Var}(\mathcal{E})| \leq 20 \\ 5 & \text{if } |\mathrm{Var}(\mathcal{E})| \leq 50 \\ |\mathrm{Var}(\mathcal{E})| + 1 & \text{else} \end{cases}$$

for every extended system $\mathcal{E}$. (Thus the C^W-LP-preconditioner is per default not used for each system with more than 50 variables) The aim of the step sizes σ_{var} and σ_{C^W} is to attain comparable computation time for systems strongly differing in variable number. Tests showed the that the step sizes reduce the overall computation time dramatically, despite lessening the contraction per box (if not only the original system is considered). In that sense, the step sizes provide a trade-off between the contraction provided and the computation time consumed by calculations on extended systems. The step sizes are an inherent part of the strategies for utilization of extended systems to be presented in this chapter (namely AS and BIH).

Algorithm 3.2 HYBRID NEWTON METHOD (box $\boldsymbol{x}$, system $\mathcal{E}$, $\mathcal{H}_{\text{precond}}$)

compute $\sigma_{var}(\mathcal{E})$ and $\sigma_{C^W}(\mathcal{E})$
repeat
  $i \leftarrow |\text{Var}(\mathcal{E})|$
  while $i > 0$ **do**
    if $i = |\text{Var}(\mathcal{E})|$ **or** volume of $\boldsymbol{x}$ was reduced by a predefined amount **then**
      compute derivatives anew
      apply CP
    end if
    $l = 0$
    repeat
      $l \leftarrow l + 1$
      apply extended Newton method with preconditioner row $C_k^{(l)}$ on system $\mathcal{E}$
    until $l \geq |\mathcal{H}_{\text{precond}}|$ **or** box small enough
    if $i \leq |\text{Var}(\mathcal{E}_{\text{orig}})|$ **then**
      $i \leftarrow i - 1$ {all variables in $\mathcal{E}_{\text{orig}}$ are considered}
    else
      if $l = 1$ **then** {C^{W}-LP-preconditioner }
        $i \leftarrow i - \sigma_{C^W}(\mathcal{E})$
      else
        $i \leftarrow i - \sigma_{var}(\mathcal{E})$
      end if
    end if
  end while
until box is small enough or contracted enough or applied maximum number of times

3.1.3 Extended systems in SONIC

Extended systems are not constructed explicitly in SONIC. They are implicitly derived from the internal representation of the fullsplit system. Which systems are constructed and used in SONIC can be chosen in the file *config.cpp*. If two extended systems are identical for a given problem, only one of them is used. As soon as all systems have been constructed, the extended systems are sorted by variable number to build a hierarchy $\mathcal{H}_\mathcal{E}$ of extended systems. (If two systems $\mathcal{E}^{(1)}$ and $\mathcal{E}^{(2)}$ are constructed with $|\mathrm{Var}(\mathcal{E}^{(1)})| = |\mathrm{Var}(\mathcal{E}^{(2)})|$) but $\mathrm{Var}(\mathcal{E}^{(1)}) \neq \mathrm{Var}(\mathcal{E}^{(2)})$) their order in $\mathcal{H}_\mathcal{E}$ follows the sequence in which the systems are built.) These are the two reasons why we may end up with different $\mathcal{H}_\mathcal{E}$ for individual problems even if we start to construct the same extended systems. If we do not consider a specific problem, especially the number of variables in the linear system is hard to relate to the CST or CSTN system. In general, we can only state the relations

$$|\mathrm{Var}(\mathcal{E}_{\mathrm{orig}})| \leq |\mathrm{Var}(\mathcal{E}_{\mathrm{cst}})| \leq |\mathrm{Var}(\mathcal{E}_{\mathrm{cstn}})| \leq |\mathrm{Var}(\mathcal{E}_{\mathrm{full}})|$$

and

$$|\mathrm{Var}(\mathcal{E}_{\mathrm{orig}})| \leq |\mathrm{Var}(\mathcal{E}_{\mathrm{lin}})| \leq |\mathrm{Var}(\mathcal{E}_{\mathrm{full}})|.$$

If we would not construct a system $\mathcal{E}^{(2)}$ based on the original system but on the previous extended system $\mathcal{E}^{(1)}$, we could also attain the relation $\mathrm{Var}(\mathcal{E}^{(2)}) \supseteq \mathrm{Var}(\mathcal{E}^{(1)})$.

$\diamond$

Having defined extended systems and contractors to work on them further questions arise. Which extended systems should be constructed? And which contractors should be run on each of them to optimize the overall computation time of the branch-and-bound algorithm? Does the choice of appropriate extended systems and contractors depend on the given problem or is it universal for all nonlinear systems of equations?

For these considerations we have to remind that coarse and fine systems each have their advantages and disadvantages. In general, coarser systems cause less computational costs due to their lower variable number. When applying the contractors to finer systems we can assume to attain a stronger contraction than on a coarser system. To balance these benefits out, the step sizes $\sigma_{var}(\mathcal{E})$ and $\sigma_{C^W}(\mathcal{E})$ are applied.

It is easy to guess that running only computational cheap contractors on the coarsest system will result in the least amount of time needed per box. However, this strategy often leads to an enormous overall box number. On the other side, running all contractors on all available extended systems tends to minimize the overall box number while increasing the computation time needed for a single box tremendously.

Unfortunately, one cannot say that either of the two strategies is better than the other. For example, for some of the harder problems (for example *min-04-07*) it is better to go for mere box throughput by disabling all contractors but CP [BLUW09]. By contrast, when applying all extended systems for problem *Trigonometric*, 12,241 boxes and 149.8 seconds are needed for the computation. Turning off all contractors on the other hand results in a vast increase of both these numbers: now 46,919,939 boxes are computed and 27,945.1 seconds are needed to consider them.

In fact, even if $\mathcal{H}_{\mathcal{E}}$ is fixed, we are not be able to determine which contractors to apply in general. This has already been discussed in section 2.3.6, where contractors for the original system were examined.

In the following, we discuss strategies to choose the extended systems to be applied. These strategies can represent the branching-step of the branch-and-bound algorithm. All strategies for the usage of extended systems have been developed to suit a wide range of problems and aim at finding a compromise between the extremes of using none or all systems.

3.2 Separate analysis phase

A first possibility to choose "promising" extended systems and contractors is the following. For each problem the branch-and-bound algorithm is started for an "analysis phase". In this phase, a computation is started just to store information about successful extended systems and contractors. Afterwards, the computation is started anew. In the second, "computation phase" the successful systems and contractors are preferred.

However, this approach has not been implemented since the author sees several disadvantages. Firstly, we cannot analyze all boxes in the first phase if the entire approach should be competitive. Since we cannot compute all boxes, we should use a depth-first recursion through the branch-and-bound algorithm in the analysis phase to get results in as many recursion levels as possible. But still only boxes in some branches of the branch-and-bound tree are analyzed. Thus predictions may only be valid for the boxes considered in the first phase.

A second problem is that it is likely that the choice for extended systems and contractors is modified when changing from the analysis to the computation phase. Hence different boxes are computed in the branch-and-bound algorithm. For these boxes the sampled information could not be suited at all. Thus the analysis phase cannot only cause an overhead in computation time but even lead to an unfavorable strategy for the computation step.

3.3 A box-intern hierarchy controlling extended systems (BIH)

Willems suggested an approach to apply the extended systems as they appear in $\mathcal{H}_{\mathcal{E}}$. His approach combines the benefits of the different extended systems in a sensible way. The coarsest systems, with anticipated low computation time, are applied first.

The hierarchical approach is applied independently for every box considered in the bounding step of the branch-and-bound algorithm. Thus we will call it the *box-intern hierarchy* or BIH. Below we describe the strategy in short. A depiction of the algorithm is given in Figure 3.2. A representation in pseudo code can be found in Algorithm 3.3. For further background information see [Wil04] (esp. p. 114 ff.).

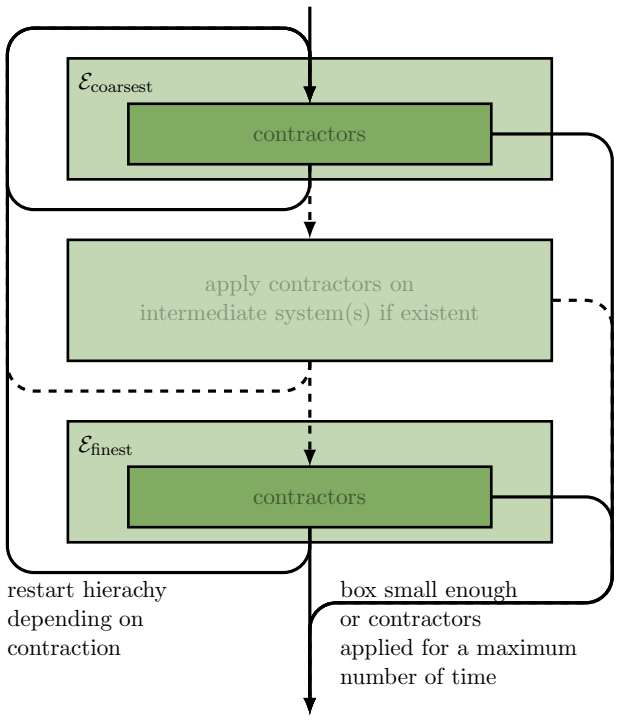


Figure 3.2: BIH for using extended systems

The strategy of the BIH mainly is a traversal of the hierarchy of extended systems. By applying $\mathcal{E}_{\text{coarsest}}$ first, a box is contracted while investing little computation time. If a box could not be contracted sufficiently, the next finer system is applied. Analogously, one proceeds to apply contractors on increasingly finer systems until a sufficient contraction has been attained or until $\mathcal{E}_{\text{finest}}$ has been applied.

Algorithm 3.3 BIH FOR USING EXTENDED SYSTEMS (box $\boldsymbol{x}$, $\mathcal{H}_{\mathcal{E}}$)

1: $\mathcal{E} \leftarrow \mathcal{E}_{\text{coarsest}}$
2: **repeat**
3: determine the number of point intervals Thin_{old} in $\boldsymbol{x}$
4: determine the relative volume of the box $\text{RelVol}_{\text{old}} \leftarrow \text{RelVol}(\boldsymbol{x})$
5: run contractors on current system $\mathcal{E}$ (Alg. 2.8)
6: determine the number of thin variables Thin_{new}
7: determine the relative volume of the box: $\text{RelVol}_{\text{new}} \leftarrow \text{RelVol}(\boldsymbol{x})$
{Which system to consider next?}
8: **if** $\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **then**
9: $\text{Thin}_{\text{old}} \leftarrow \text{Thin}_{\text{new}}$
10: $\text{RelVol}_{\text{old}} \leftarrow \text{RelVol}_{\text{new}}$
11: **end if**
12: **if** $\mathcal{E} \neq \mathcal{E}_{\text{coarsest}}$ **and**
$(\text{RelVol}_{\text{new}} \cdot \kappa < \text{RelVol}_{\text{old}}$ **or** $\text{Thin}_{\text{new}} > \text{Thin}_{\text{old}})$ **then**
13: {contraction sufficient for restart}
14: $\mathcal{E} \leftarrow \mathcal{E}_{\text{coarsest}}$
15: $\text{Thin}_{\text{old}} \leftarrow \text{Thin}_{\text{new}}$
16: $\text{RelVol}_{\text{old}} \leftarrow \text{RelVol}_{\text{new}}$
17: **else if** box $\boldsymbol{x}$ is small enough **or** $\mathcal{E} = \mathcal{E}_{\text{finest}}$ **then**
18: break {terminate repeat-loop}
19: **else**
20: $\mathcal{E} \leftarrow$ next finer extended system
21: **end if**
22: **until** the loop body has been executed a maximum number of times

The success of contraction is monitored via the *relative volume*

$$\mathrm{RelVol}(\boldsymbol{x}) := \prod_{\mathrm{width}\,(\boldsymbol{x}_j)\neq 0} \mathrm{width}\,(\boldsymbol{x}_j)$$

of box $\boldsymbol{x}$. Replacing the width in Definition 1.10 by the modified volume we attain the new qualifier

$$\frac{\mathrm{RelVol}(\boldsymbol{x})}{\mathrm{RelVol}(\boldsymbol{x}_{new})} > \kappa$$

for the contraction success of a contractor. The default choice for factor κ in SONIC is $10^{0.025}$ (approximately 1.06). Thin components do not contribute to the relative volume, but are counted separately. A reduction of a component from a proper interval to a point interval is considered as a sufficient contraction success, too.

In addition to this simple strategy, Willems restarted the traversal of $\mathcal{H}_{\mathcal{E}}$, whenever a box is contracted successfully in a system $\mathcal{E} \neq \mathcal{E}_{\mathrm{coarsest}}$. In that case it can be expected that $\mathcal{E}_{\mathrm{coarsest}}$ may again be successful on the changed box $\boldsymbol{x}$.

The traversal will be aborted, when the box is contracted to the desired size for a solution box or when the extended systems have been applied for a predefined maximum number of times. Willems chose a regulation depending on the number of available extended systems. In the default settings the parameter *MaxRepsHierarchyPerSystem* is set to 2. With this parameter the number of applied extended systems is determined as twice the number of overall extended systems. This the finest system is applied at most twice, which happens if the hierarchy is restarted only once after the finest system was applied. However, it is also possible that the finest systems are never applied because coarser systems provide enough contraction and the hierarchy is restarted before reaching the finer systems themselves.

In SONIC the threshold value for restarting the traversal of $\mathcal{H}_{\mathcal{E}}$ is *RestartHierarchyFactor*. It determines κ by the relation $\kappa = 10^{RestartHierarchyFactor}$.

Willems already speculated that his regulation by hierarchy may not be the ideal heuristic for the application of the extended systems. In fact, the author will derive a more promising heuristic in the next sections.

3.4 Studies for selecting extended systems

One drawback we see in the box-intern hierarchy is that every box is considered independently of all others. Therefore the heuristic makes no use of information gained in earlier computations on other boxes. In the following, strategies are

developed to deduce additional information from the overall branch-and-bound algorithm and to exploit it for regulating the use of the different extended systems.

Eventually, an adaptive strategy for selecting systems is presented. Numerical tests will show that the new strategy significantly improves the average computation time compared to the box-intern hierarchy.

3.4.1 Skipping extended systems

One way to save computation time without waiving extended systems $\mathcal{E} \neq \mathcal{E}_{\text{orig}}$ completely is to apply them only a subset of the boxes evolving in the branch-and-bound algorithm. But how should we decide on which boxes to use them?

One approach would be to use all systems on the boxes in the upmost recursion levels. The more those boxes are contracted, the more subboxes can be spared in deeper recursion levels. However, this approach has two downsides. Firstly, one still has no clue in how many recursion levels the finer extended systems should be employed. Secondly, the interval Newton method is usually more powerful when applied to small boxes. Thus it may not be able to provide contraction for the wide boxes considered in upper recursion levels of the branch-and-bound algorithm.

Our second approach adapts the box-intern hierarchy and tries to estimate the contraction success of the extended systems. A system $\mathcal{E} \neq \mathcal{E}_{\text{orig}}$ is only employed in the next traversing of $\mathcal{H}_{\mathcal{E}}$ if it provided contraction for the current box. If it does not contract or (alternatively) not contract successfully, we assume that the considered box is too wide and has to be reduced in size before starting the extended systems anew.

The simplest way to secure a reduction in box size is to subdivide the box. We can even subdivide more than once before applying the extended systems again to save computation time on the finer systems. Algorithm 3.4 clarifies this "skipping" of extended systems. Within the algorithm we use the step size $\sigma^{\mathcal{E}}$ which is a fixed number that can be chosen separately for each extended system $\mathcal{E}$.

Note that the employed systems are not necessarily the same for all boxes in a recursion level. Which extended systems are allowed for a given box depends on their success in the ancestorboxes. One could also apply the approach of skipping systems on whole recursion levels. In this case, whether or not to apply extended systems would be chosen uniformly for all boxes in a recursion level. However, the decision whether to use finer extended systems on the following recursion level would be complicated. Further it is not recommendable to make decisions for whole recursion levels since the function may behave very differently in different subboxes of the startbox $\boldsymbol{x}^{(0)}$.

Tables 3.1 and 3.2 state the computation time and box number needed for the box-intern hierarchy as presented in section 3.3 and different strategies to employ extended systems. The modifications are described below. Shown are the results

for test set $\mathcal{T}_A$ (cf. appendix) and the default hierarchy $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$. Note that the CST system was not constructed for *Brent7* and *Eco9* because those problems do not contain any subterms with multiple occurrences. For conclusive time measurements, verification (see Chapter 4) was deactivated for all tests contained in this chapter.

The first part of tables 3.1 and 3.2 summarizes results for simple modifications of the box-intern hierarchy. In the first modification the box-intern hierarchy was allowed to restart after $\mathcal{E}_{\text{coarsest}}$ (here $\mathcal{E}_{\text{orig}}$) was applied (cf. line 12 in Alg. 3.3). In a second setting, the finer extended systems were started, but the expensive Newton method was conducted only on the original system. For the third modification every extended system was employed exactly once per box. In a last setting only the original system was allowed. The table easily reveals that test set $\mathcal{T}_A$ contains only one problem for which extended systems are beneficial in the standard version of BIH.

The second part of the tables 3.1 and 3.2 states the results for the skipping of extended systems. The headers state the extended systems $\mathcal{E}_{\text{skip}}$ that are "skipped", and the number $\sigma^\mathcal{E}$ of subdivision steps for which the systems are not applied. So all extended systems are applied until $\mathcal{E}_{\text{skip}}$ no longer provides enough contraction. Subsequently $\mathcal{E}_{\text{skip}}$ is skipped for $\sigma^\mathcal{E}$ recursion levels before giving it a new try.

If a value for κ is given in the table, the skipping is applied with respect to the success of the extended system. This more sophisticated strategy is employed according to Algorithm 3.4. The success of the contraction is computed as for the box-intern hierarchy, by a ratio of relative volumes. Parameter κ is used as the threshold to decide, whether a method provided enough contraction and should be used again. In any other case system $\mathcal{E}$ is skipped for $\sigma^\mathcal{E}$ subdivision steps. (The approximate values for κ are $10^{0.025} \approx 1.06$, $10^{0.2} \approx 1.58$ and $10^{0.5} \approx 3.2$.) Note that the finer extended systems do not have to be applied, although this would be allowed. When a box could be contracted enough by coarser systems, there is no need to apply further finer ones.

Table 3.2 shows that the average box number cannot be improved by any modification. This was to be expected due to the more seldom application of finer systems. Computation time on the other hand can be saved by skipping finer extended systems as is displayed in Table 3.1. For problem *7erSystem* savings in computation time up to 75% are observed when skipping systems. In fact, for test set $\mathcal{T}_A$, improved computation time can be attained for all problems except the *Trigonometric* problem. For this problem skipping the CST system leads to a significant increase in computation time.

For all problems but *Trigonometric* computation time decreases when increasing parameter $\sigma^\mathcal{E}$. This is accounted for by less time-consuming calculations when skipping systems more often. For parameter κ the value $10^{0.025}$ turned out to be slightly more advantageous than the other tested values.

Algorithm 3.4 SKIPPING EXTENDED SYSTEMS (box $\boldsymbol{x}$, step size $\sigma^{\mathcal{E}}$, success rate κ)

1: if $\boldsymbol{x}$ equals startbox $\boldsymbol{x}^{(0)}$: $\sigma \leftarrow \sigma = 1$
2: $\mathcal{E} \leftarrow \mathcal{E}_{\text{coarsest}}$
3: boolean value success $\leftarrow$ false
4: determine Thin_{old} as the number of thin variables of box $\boldsymbol{x}$
5: determine $\text{RelVol}_{\text{old}}$ as the relative volume of the box $\boldsymbol{x}$

6: **for** $\mathcal{E}$ traversing $\mathcal{H}_{\mathcal{E}}$ in coarse-to-fine order **do**
7: **if** $\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **or** $\sigma = 1$ **then**
8: run contractors on current system $\mathcal{E}$ (Alg. 2.8)
9: **end if**
{Which system to consider next?}
10: **if** $\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **then**
11: $\text{Thin}_{\text{old}} \leftarrow \text{Thin}_{\text{new}}$, $\text{RelVol}_{\text{old}} \leftarrow \text{RelVol}_{\text{new}}$
12: **end if**
13: **if** $\mathcal{E} \neq \mathcal{E}_{\text{coarsest}}$ **and**
$(\text{RelVol}_{\text{new}} \cdot \kappa < \text{RelVol}_{\text{old}}$ **or** $\text{Thin}_{\text{new}} > \text{Thin}_{\text{old}})$ **then**
14: success $\leftarrow$ true
15: **else if** box $\boldsymbol{x}$ is small enough **or** $\mathcal{E} = \mathcal{E}_{\text{finest}}$ **then**
16: break
17: **end if**
18: **end for**
19: **if** success = true **or** $\sigma > \sigma^{\mathcal{E}}$ **then**
20: $\sigma \leftarrow 1$
21: **else**
22: increase σ by one
23: **end if**

Further tests, not presented here, showed only minor differences in needed computation time when no restart is used in the box-intern hierarchy.

For a general approach on how to select the extended systems to be applied for an arbitrary problem, the different behavior of the problems has to be considered. For example, it emerged to be better to discard just the fullsplit or no system at all for problem *Trigonometric* whereas for *min-04-07* it is a better choice to skip both finer systems, the CST and fullsplit system.

Thus we easily see that none of the investigated simple approaches is suited for all problems. A more sophisticated method is needed to decide, which extended systems should be employed. One promising approach to do so is scrutinized in the next section.

setting			7erSystem	Brent7	Eco9	Trigonometric	min-04-07
default version of BIH			*436.7*	*78.5*	*84.5*	128.6	*1830.2*
also restart after $\mathcal{E}_{\text{orig}}$			*436.6*	*78.4*	*84.5*	128.6	*1835.6*
Newton only on $\mathcal{E}_{\text{orig}}$			*436.7*	*78.4*	*84.6*	128.7	*1835.9*
every system once per box			*433.9*	*78.5*	*84.6*	125.3	*1829.9*
only $\mathcal{E}_{\text{orig}}$			**83.7**	**27.1**	**47.0**	*472.8*	**1020.3**
$\mathcal{E}_{\text{skip}}$	$\sigma^{\mathcal{E}}$	κ					
$\mathcal{E}_{\text{full}}$	5	-	156.2	38.9	56.2	**111.7**	1319.2
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	-	155.8	38.9	56.1	175.6	1170.3
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.025}$	207.8	45.6	61.1	152.2	1287.2
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.2}$	213.4	45.7	61.1	152.2	1284.9
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.5}$	230.4	46.2	61.2	148.1	1284.7
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.025}$	156.5	38.9	56.1	174.1	1173.1
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.2}$	160.8	39.2	56.1	181.2	1171.9
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.5}$	168.0	39.6	56.2	176.6	1171.4
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.025}$	106.8	33.9	52.6	355.9	1114.1
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.2}$	107.8	34.1	52.6	357.7	1115.4
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.5}$	111.7	34.6	52.7	357.5	1113.6

Table 3.1: Computation time (in seconds) for variations of BIH (with markers for the **lowest** and *highest* values)

setting			7erSystem	Brent7	Eco9	Trigonometric	min-04-07
default version of BIH			**11523**	**51851**	**30109**	**18569**	**68185**
also restart after $\mathcal{E}_{\text{orig}}$			**11523**	**51851**	**30109**	**18569**	**68185**
Newton only on $\mathcal{E}_{\text{orig}}$			**11523**	**51851**	**30109**	**18569**	**68185**
every system once per box			**11789**	**51903**	**30171**	**18913**	**68185**
only $\mathcal{E}_{\text{orig}}$			*24035*	*55205*	*36889*	*204013*	*73287*
$\mathcal{E}_{\text{skip}}$	$\sigma^{\mathcal{E}}$	κ					
$\mathcal{E}_{\text{full}}$	5	-	**12623**	54253	30861	**19487**	70165
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	-	16707	54253	30861	55469	71771
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.025}$	14953	53489	30685	35011	70595
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.2}$	14611	53365	30631	33951	70595
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	3	$10^{0.5}$	13921	52683	30537	31593	70593
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.025}$	16677	54249	30861	49973	71771
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.2}$	16499	54037	30819	48211	71771
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	5	$10^{0.5}$	16165	53275	30787	45601	71769
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.025}$	17885	54833	30905	*127917*	71931
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.2}$	17801	54719	30889	*128721*	71931
$\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{full}}$	10	$10^{0.5}$	17627	53913	30871	*127937*	71929

Table 3.2: Box numbers for variations of BIH (with markers for the **lowest** and *highest* values)

3.4.2 Predicting contraction success

In the last subsection we simply presumed it to be more likely to contract a box $\boldsymbol{x}$ on a certain extended system $\mathcal{E}$ if its parentbox or ancestorbox could be contracted by $\mathcal{E}$. The theoretical backbone of this ansatz is our assumption that a function often behaves similarly on a box and its subboxes. Of course, this presumption cannot always be fulfilled, but we expect it to work for the majority of boxes (especially if they are sufficiently small).

The author thus proposes to use measurements of the success of contractors on the parentbox and ancestorboxes as a hint, which contractors are most probably suited for box $\boldsymbol{x}$. If our approach shows to be reasonable, we attain the possibility to select the extended system to be applied by evaluating the former successes in contracting boxes and thus to construct more sophisticated decision strategies.

Hence want to evaluate this idea in more detail. Tests were conducted to strengthen the made assumption. For that purpose, the contraction achieved in every extended system was measured. Every constructed system was started once in the box-intern hierarchy. For the upcoming tests the hierarchy $\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$ is used. (For the problems *Brent7* and *Eco9* again only the original system and the fullsplit systems are constructed.)

Note that the different systems are not necessarily applied to the same box since the boxes may have been contracted by coarser systems. Furthermore, for different options, different boxes emerge in the branch-and-bound algorithm. This is the point where we encounter an inherent problem of tests in the branch-and-bound algorithm: we can never test every possible regulation of contractors and extended systems depending on properties of the boxes. Thus we have to put up with an assortment of sensible tests. With the results of these tests we then build reasonable heuristics. (Within the tests we only examine the systems in hierarchies ordered by variable number this is the one in which we want to apply our extended systems later on.)

For the next test the numbers of contracted boxes are considered. Those are given as ratios with respect to the success of the extended systems for the parentboxes. The contraction is achieved either by the extended Newton method or by all used contractors.

For a formal denotation some events are introduced. Remember that for a contractor μ a box is called not contracted if $\mu(\boldsymbol{x}) = \boldsymbol{x}$ and contracted if $\mu(\boldsymbol{x}) \subsetneq \boldsymbol{x}$. However, no statement is made about the quality of a contraction.

B	box was contracted
$\overline{B}$	box was not contracted
P	parentbox was contracted
$\overline{P}$	parentbox was not contracted

Now we can count the occurrences of each event in the branch-and-bound algo-

rithm. The amount of boxes for which a certain event X is observed is denoted by $p(X)$. For two arbitrary events X and Y we define $p(X, Y)$ as the amount of boxes for which event X and Y were observed. Herein all numbers are normalized by the overall number of boxes. Due to the normalization the relation

$$p(B, P) + p(\overline{B}, \overline{P}) + p(B, \overline{P}) + p(\overline{B}, P) = 1$$

holds. (Since rounded numbers are displayed in the upcoming tables, the given values do not always sum up to 1 exactly.) The denotation is deliberately chosen to resemble that of probabilities. For example, $p(B, P)$ can be interpreted as the probability to observe that a box and its parentbox in the branch-and-bound algorithm are both contracted.

Tables 3.3 and 3.4 display the results calculated for test set $\mathcal{T}_A$. They also tell us how often the extended Newton method is not started because the current box was contracted enough before. This number gives an impression, how often constraint propagation and Taylor refinements are sufficient to contract a box. It also gives us a hint that less computation time was demanded for the extended Newton method (since it was called less often).

For the computation of the contraction rates an altered volume computation is used. This was necessary because the relative volume RelVol can yield zero or infinite volumes and is thus not suited for computing ratios. To surmount this obstacle we introduced two thresholds θ_j, sufficiently small, and Θ_j, sufficiently large, to handle very narrow and very wide intervals. We achieve a finite, nonzero approximate for the box volume, called the *thresholded volume*, defined by

$$\mathrm{ThVol}(\boldsymbol{x}) := \prod_j \mathrm{width}_{\,\mathrm{Th}}(\boldsymbol{x}_j),$$

where

$$\mathrm{width}_{\,\mathrm{Th}}(\boldsymbol{x}_j) := \begin{cases} \theta_j & \text{if } \mathrm{width}\,(\boldsymbol{x}_j) \leq \theta_j \\ \mathrm{width}\,(\boldsymbol{x}_j) & \text{if } \theta_j \leq \mathrm{width}\,(\boldsymbol{x}_j) \leq \Theta_j \\ \Theta_j & \text{if } \Theta_j \leq \mathrm{width}\,(\boldsymbol{x}_j) \end{cases}.$$

No further treatment is needed for unbounded or point intervals. One reason for doing so is that we see no great advantage in getting from a narrow interval to a point interval if the required precision for this interval is already reached and the contraction does not influence the function enclosure over the box. The volume of a box before the application of an extended system is denoted as $\mathrm{ThVol}_{\mathrm{old}}$, and afterwards as $\mathrm{ThVol}_{\mathrm{new}}$.

As an advantage of the thresholded volume, information about whether a box was contracted in any dimension is consolidated into a single number. And it is applicable as well for boxes with zero or infinite volume. A further virtue of the new definition is that the computed volume of a box cannot get larger. This was possible when using the relative volume RelVol if a box component was contracted

to a point interval. As already suggested in [SA02, Wil04], the volumes are used in a logarithmic form in the implementation for numerical reasons (especially to avoid over- or underflows).

The quotient $\gamma^{\mathcal{E}}_{\text{contr}}$ is computed as the arithmetic mean of the contraction rate over all boxes, for which the contractors on a given system were applied. A system $\mathcal{E}$ is rated as more successful the smaller $\gamma^{\mathcal{E}}_{\text{contr}}$ gets.

$$\gamma^{\mathcal{E}}_{\text{contr}} := \frac{1}{\text{overall calls of } \mathcal{E}} \cdot \sum_{\boldsymbol{x} \text{ considered in } \mathcal{E}} (\text{ThVol}_{\text{new}}(\boldsymbol{x})/\text{ThVol}_{\text{old}}(\boldsymbol{x}))$$

Moreover, the computation time is desired to be as small as possible, too. Thus a further quotient $\gamma^{\mathcal{E}}_{\text{time}}$ is computed to assess the success of the system with respect to time($\boldsymbol{x}$), the time spent on a box $\boldsymbol{x}$.

$$\gamma^{\mathcal{E}}_{\text{time}} := \frac{1}{\text{overall calls of } \mathcal{E}} \cdot \sum_{\boldsymbol{x} \text{ considered in } \mathcal{E}} (\text{ThVol}_{\text{new}}(\boldsymbol{x})/\text{ThVol}_{\text{old}}(\boldsymbol{x}) \cdot \text{time}(\boldsymbol{x}))$$

Hence a smaller value of $\gamma^{\mathcal{E}}_{\text{time}}$ indicates better average performance in contraction per time.

Tables 3.3 confirms the contraction success of $\mathcal{E}_{\text{cst}}$ for problem *Trigonometric* and that for *min-04-07* the extended Newton method seldom provides contraction, no matter on which system, as indicated by earlier measurements skipping systems.

Tables 3.3 and 3.4 further reveal that for test set $\mathcal{T}_A$ the contraction success of a box indeed is related to the success on the parentbox. In general

$$p(B, P) + p(\overline{B}, \overline{P}) > p(B, \overline{P}) + p(\overline{B}, P).$$

is valid. In many cases even

$$p(B, P) + p(\overline{B}, \overline{P}) > 0.7$$

holds, indication a strong correlation of the contraction in a box and its parentbox.

problem	system	calls	Newton not done	$p(B,P)$	$p(\overline{B},\overline{P})$	$p(B,\overline{P})$	$p(\overline{B},P)$	$\gamma^{\mathcal{E}}_{\text{contr}}$	$\gamma^{\mathcal{E}}_{\text{time}}$
Brent7	$\mathcal{E}_{\text{orig}}$	41274	78	0.03	0.78	0.10	0.10	0.99	0.0004
	$\mathcal{E}_{\text{full}}$	25961	15313	0.19	0.61	0.10	0.10	0.95	0.0019
Eco9	$\mathcal{E}_{\text{orig}}$	30134	0	0.36	0.21	0.22	0.22	0.66	0.0007
	$\mathcal{E}_{\text{full}}$	15438	14696	0.06	0.70	0.12	0.12	0.95	0.0022
Trigonometric	$\mathcal{E}_{\text{orig}}$	19245	83	0.11	0.68	0.11	0.11	0.85	0.0006
	$\mathcal{E}_{\text{cst}}$	14890	4355	**0.63**	0.12	0.12	0.12	0.40	0.0010
	$\mathcal{E}_{\text{full}}$	9615	5275	0.34	0.35	0.15	0.15	0.74	0.0017
7erSystem	$\mathcal{E}_{\text{orig}}$	11765	0	**0.94**	0.00	0.03	0.03	0.33	0.0007
	$\mathcal{E}_{\text{cst}}$	7444	4321	**0.66**	0.06	0.14	0.14	0.64	0.0023
	$\mathcal{E}_{\text{full}}$	6289	1155	0.02	0.84	0.07	0.07	0.96	0.0556
min-04-07	$\mathcal{E}_{\text{orig}}$	66039	0	0.01	*0.93*	0.03	0.03	0.98	0.0078
	$\mathcal{E}_{\text{cst}}$	34693	31346	0.01	*0.90*	0.04	0.04	0.96	0.0041
	$\mathcal{E}_{\text{full}}$	34064	629	0.01	*0.93*	0.03	0.03	0.97	0.0187

Table 3.3: Contraction success of the extended Newton method in relation to success on the parentbox on the same system

problem	system	calls	Newton not done	$p(B,P)$	$p(\overline{B},\overline{P})$	$p(B,\overline{P})$	$p(\overline{B},P)$	$\gamma^{\mathcal{E}}_{\text{contr}}$	$\gamma^{\mathcal{E}}_{\text{time}}$
Brent7	$\mathcal{E}_{\text{orig}}$	41274	78	0.60	0.06	0.17	0.17	0.56	0.0002
	$\mathcal{E}_{\text{full}}$	25961	15313	0.19	0.61	0.10	0.10	0.95	0.0019
Eco9	$\mathcal{E}_{\text{orig}}$	30134	0	0.86	0.01	0.06	0.06	0.31	0.0003
	$\mathcal{E}_{\text{full}}$	15438	14696	0.06	0.70	0.12	0.12	0.95	0.0023
Trigonometric	$\mathcal{E}_{\text{orig}}$	19245	83	0.66	0.11	0.12	0.12	0.55	0.0005
	$\mathcal{E}_{\text{cst}}$	14890	4355	0.63	0.12	0.12	0.12	0.40	0.0011
	$\mathcal{E}_{\text{full}}$	9615	5275	0.34	0.35	0.15	0.15	0.74	0.0019
7erSystem	$\mathcal{E}_{\text{orig}}$	11765	0	1.00	0.00	0.00	0.00	0.29	0.0007
	$\mathcal{E}_{\text{cst}}$	7444	4321	0.66	0.06	0.14	0.14	0.64	0.0025
	$\mathcal{E}_{\text{full}}$	6289	1155	0.02	0.84	0.07	0.07	0.96	0.0556
min-04-07	$\mathcal{E}_{\text{orig}}$	66039	0	0.50	0.13	0.19	0.19	0.46	0.0031
	$\mathcal{E}_{\text{cst}}$	34693	31346	0.01	0.90	0.04	0.04	0.96	0.0042
	$\mathcal{E}_{\text{full}}$	34064	629	0.01	0.93	0.03	0.03	0.97	0.0185

Table 3.4: Contraction success of all contractors on extended systems in relation to success on the parentbox on the same system

3.5 A new, adaptive strategy (AS)

The basis for our new heuristic is the observation made in the last section: for a given box the most efficient extended systems are probably those that have already been successful in contracting the parentbox. The new strategy deduces information from parent- and ancestorboxes to choose promising extended systems. This means that the selection of extended systems for a box is guided "by experience" and adapts itself to each specific problem during the computation. Moreover, the strategy adapts the usage of the extended systems for every individual box. Its main goal is to reduce the computation time by starting only extended systems for which it is assumed that they can contract a given box.

For each box the new heuristic processes through the hierarchy of extended systems (without restarts). Whether or not to apply each system is decided separately for each box. No system is employed more than once per box.

For every box the extended systems $\mathcal{E}$ are started depending on *gauges* or *key figures* $\gamma^{\mathcal{E}}$. Each $\gamma^{\mathcal{E}}$ is composed out of the contraction rates and the computation time of the contractors in system $\mathcal{E}$. All key figures are initialized with zero for the startbox. Every other box inherits the values $\gamma^{\mathcal{E}}_{\text{old}}$ of its parentbox. The value for each extended system $\mathcal{E}$ is updated if it is applied and stays the same in any other case.

To obtain smoother behavior, old values are taken into account when updating. Thus the update is done by averaging the previous value with $\gamma^{\mathcal{E}}_{\text{new}}$, which is the product of contraction ratio and computation time needed for the current run. Hence the gauge $\gamma^{\mathcal{E}}$ is computed by

$$\gamma^{\mathcal{E}} := \begin{cases} 0 & \text{for the startbox } \boldsymbol{x}^{(0)} \\ \beta \cdot \gamma^{\mathcal{E}}_{\text{new}} + (1-\beta) \cdot \gamma^{\mathcal{E}}_{\text{old}} & \text{for all other boxes} \end{cases}$$

for a fixed weight $\beta \in [0,1]$. In the computation of the average it emerged to be useful to weigh the new value with $\beta < 1/2$ to alleviate the impact of outliers in running time or contraction.

Using the $\gamma^{\mathcal{E}}$ we achieve a reliable, adaptive strategy. Moreover, we have to store and process only one additional value per box and system. To improve the start value of $\gamma^{\mathcal{E}}$, by default all extended systems are run for the three upmost recursion levels. If each box is subdivided into a maximum of two subboxes, these levels contain a maximum of only seven boxes, but these include the first three boxes in every path of the branch-and-bound tree beginning in the startbox $\boldsymbol{x}^{(0)}$. By virtue of this procedure, at least three values for averaging are calculated per system. As a positive side-effect, all boxes are contracted by all systems in the upmost recursion levels. This helps to reduce the number of boxes to be considered in deeper recursion levels.

In the deeper recursion levels two comparisons are used to decide whether a system $\mathcal{E}$ is applied. The coarsest system $\mathcal{E}_{\text{coarsest}}$ in the hierarchy of extended

systems is applied unless the gauge $\gamma^{\mathcal{E}_{\text{coarsest}}}$ is larger than the gauge of the next finer system $\mathcal{E}_{\text{coarsest}+1}$. The other systems $\mathcal{E}$ are applied if the gauge $\gamma^{\mathcal{E}}$ is small compared to $\gamma^{\mathcal{E}_{\text{coarsest}}}$. In this comparison a parameter α is used.

The adaptive strategy measures the contraction and computation time needed for all contractors on the extended systems. Although usually the extended Newton method accounts for the largest part of the computation time, the times for constraint propagation and Taylor refinements are included in the measurements. This is done because for some problems also constraint propagation and Taylor refinements cause a significant part of the computation time. So the adaptive strategy regulates the usage of the extended systems.

By calculating the key figures $\gamma^{\mathcal{E}}$ and using them to regulate the usage of the extended systems the author attains an adaptive strategy "driven by experience". The line of action for the adaptive strategy (AS) on each box is illustrated by Algorithm 3.5 and Figure 3.3.

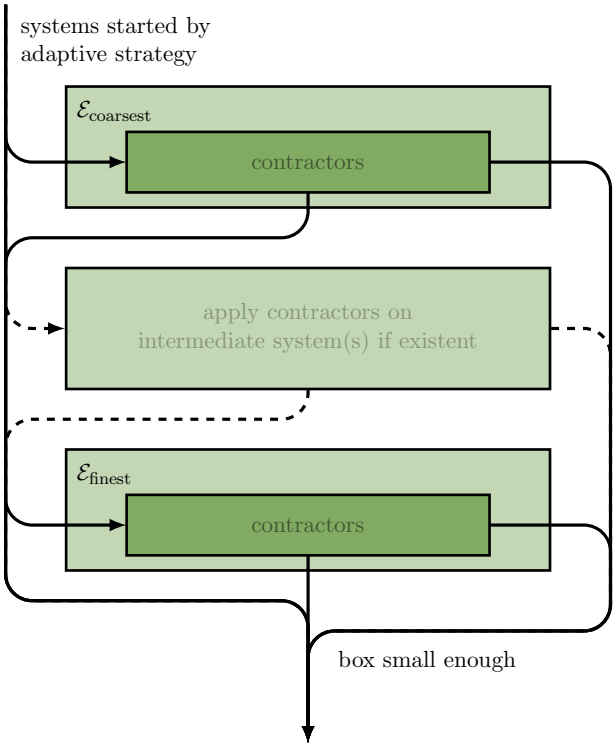


Figure 3.3: AS for using extended systems

Algorithm 3.5 AS FOR USING EXTENDED SYSTEMS (box $\boldsymbol{x}$, $\mathcal{H}_\mathcal{E}$)

for $\mathcal{E}$ traversing $\mathcal{H}_\mathcal{E}$ in coarse-to-fine order **do**
  if (recursion level ≤ 3) **or**
    ($\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}_{\text{coarsest}}} \leq \gamma^{\mathcal{E}_{\text{coarsest}+1}}$) **or**
    ($\mathcal{E} \neq \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}} < \alpha \cdot \gamma^{\mathcal{E}_{\text{coarsest}}}$) **then**
    $\text{ThVol}_{\text{old}} \leftarrow \text{ThVol}(\boldsymbol{x})$
    $\text{time}_{\text{start}} \leftarrow$ current time
    run contractors on current system $\mathcal{E}$ (Alg. 2.8)
    $\text{ThVol}_{\text{new}} \leftarrow \text{ThVol}(\boldsymbol{x})$
    $\text{time}_{\text{end}} \leftarrow$ current time
    $\gamma^{\mathcal{E}}_{\text{new}} \leftarrow (\text{time}_{\text{end}} - \text{time}_{\text{start}}) \cdot (\text{ThVol}_{\text{new}}/\text{ThVol}_{\text{old}})$
    $\gamma^{\mathcal{E}} \leftarrow \beta \cdot \gamma^{\mathcal{E}}_{\text{new}} + (1-\beta) \cdot \gamma^{\mathcal{E}}$
  end if
end for

3.5.1 Implementation details

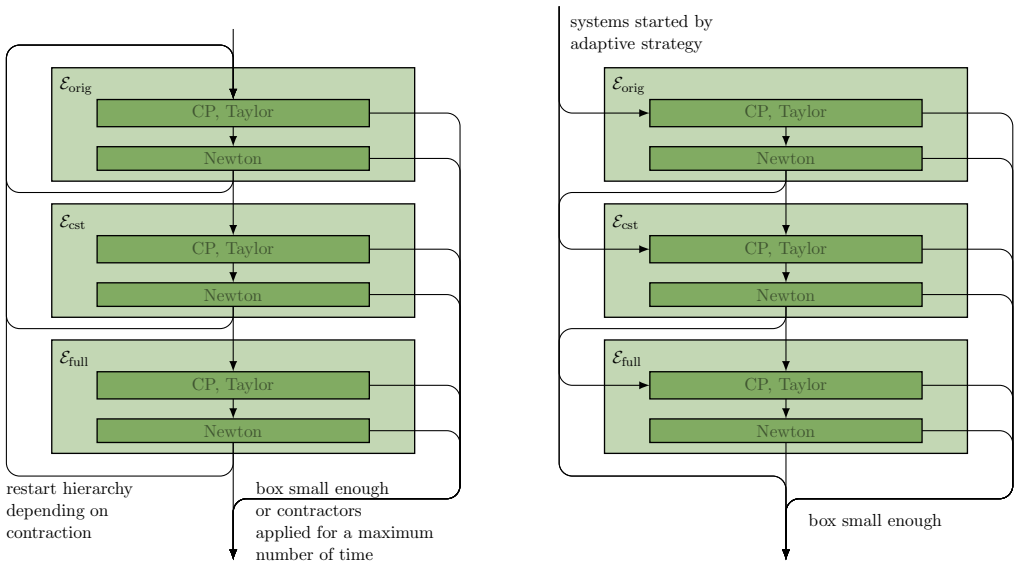
The adaptive strategy was implemented to be applicable for arbitrary hierarchies $\mathcal{H}_\mathcal{E}$. The gauges $\gamma^{\mathcal{E}}$ are stored in the data structure for the boxes and are inherited by the subboxes when the box is subdivided. Thus, no global variables are needed and the adaptive strategy can be applied in parallel versions of the program (cf. section 5.4).

The parameters α and β can be found in the Settings-File in SONIC as *AveragingGauge* and *ComparisonFactorForLargerSystems*, respectively. The default setting for the parameters are $\alpha = 4$ and $\beta = 0.25$.

Note that the box number, the computation time and the set of solution boxes may differ slightly in each run of the adaptive strategy. This is due to its dependence on the computation time, which can deviate in different runs. So this algorithm is not deterministic. In all conducted tests, however, times and box number were nearly the same. The observed differences between two runs were less than 1%. Note that the non-deterministic behavior is by no means a general downside, but an essential characteristic allowing the algorithm to adapt itself according to the computation time. If deterministic behavior is necessary for special purposes, e.g., debugging, one still can resort to the box-intern hierarchy.

3.6 Comparison of the strategies BIH and AS

We now want to compare the box-intern hierarchy (BIH) and the newly developed adaptive strategy (AS). For the numerical tests both strategies use $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$ (compare Figure 3.5).



(a) Box-intern hierarchy

(b) Adaptive strategy

Figure 3.4: BIH and AS for the default hierarchy $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$

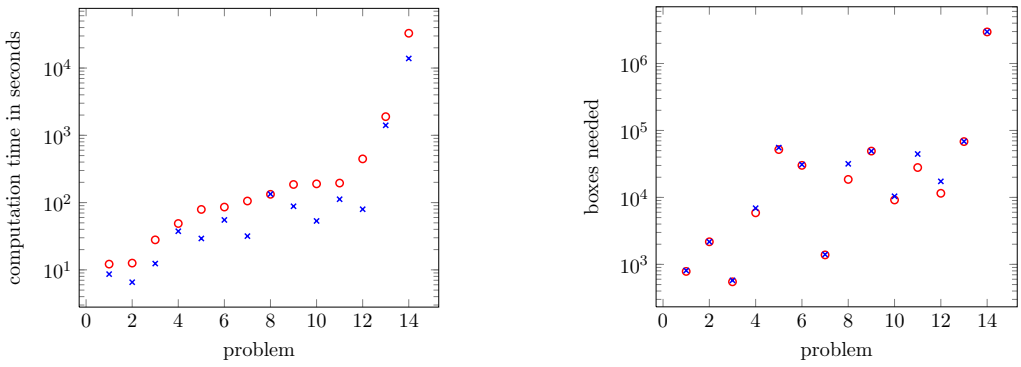
3.6.1 Comparison in computation time and box number

Considering that test set $\mathcal{T}_A$ (cf. appendix) was applied for testing and calibration of the adaptive strategy and the larger test set $\mathcal{T}_B$ is employed to validate improvements. Numerical tests yielded evidence that the new adaptive strategy is superior to the box-intern hierarchy with respect to computation time.

Table 3.5 lists the box numbers and computation time needed in both strategies. One can see that for some test problems the adaptive strategy achieves a major decrease in computation time (eg. *7erSystem*). Even more, there exists no problem for which the computation time increases significantly. The only problem for which the computation time is not reduced is *Trigonometric*. However, for this problem we already noted in earlier tests that a reduced usage of finer extended systems most often increases the computation time instead of reducing it (cf. Table 3.1).

problem	$\text{boxes}_{\text{BIH}}$	boxes_{AS}	time_{BIH}	time_{AS}
min-04-06	785	809	12.2	**8.6**
G7_gradientsystem	2175	2175	12.6	**6.6**
Reactor	549	577	27.9	**12.4**
Chemistry1	5889	*6907*	48.9	**37.5**
Brent7	51851	55421	79.1	**29.3**
Eco9	30109	30929	85.8	**55.2**
Trigexp1	1385	1405	105.6	**31.7**
Trigonometric	18569	*31751*	132.7	133.9
Chemistry2	49195	49201	185.9	**88.2**
DirectKinematics	9157	10401	189.8	**53.3**
DesignProblem9	27949	*44497*	195.0	**112.2**
7erSystem	11523	*17375*	446.3	**79.7**
min-04-07	68185	68699	1894.4	**1408.9**
Chemistry3	2959459	2976919	32902.3	**13886.7**

Table 3.5: Comparison of the box-intern hierarchy (BIH) and adaptive strategy (AS)



(a) Comparison in computation time

(b) Comparison in box number

Figure 3.5: Comparison of BIH (○) and AS (×) (problem numbers refer to order in test set $\mathcal{T}_B$)

Box numbers, on the other side, usually increase for the adaptive strategy. This happens because the large systems, anticipated to provide the best contraction, are applied less often. Therefore more boxes have to be considered in the branch-and-bound algorithm. However, the contractors applied for the boxes are potentially cheaper in computational costs. Thus, for some problems, like *DesignProblem*, the box number is significantly higher than for the box-intern hierarchy. Still the adaptive strategy achieves better results in computation time.

The overall results can be condensed in two simple numbers: the arithmetic mean of the ratio of the box number in box-intern hierarchy (BIH) and adaptive strategy (AS)

$$\frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{boxes}_{\text{AS}}}{\text{boxes}_{\text{BIH}}} = 1.17$$

and the ratio of the computation time needed for the considered test set

$$\frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{time}_{\text{AS}}}{\text{time}_{\text{BIH}}} = 0.53.$$

Hence, on average, the adaptive strategy considers 17% more boxes, but needs only about half the computation time of the box-intern hierarchy.

Note that the adaptive strategy cannot only adapt the usage of extended systems to a given problem and box. If the times needed for computing different operations vary for different hardware structures, the adaptive strategy still chooses time-efficient systems. Thus it can even adjust to the underlying soft- and hardware structure.

Our tests conducted with the interval library filib^{++} instead of C-XSC and another machine (Intel®Core$^{\text{TM}}$2 Duo CPU P8600 @ 2.40GHz) showed equally good results when comparing our adaptive strategy to the box-intern hierarchy. Using filib^{++}, the computation time decreased by 24% (box number raised by 16%). On the second machine, the computation time was reduced by 46% (box number increased by 13%) if C-XSC was used and using filib^{++} we observed a saving of 58% in computation time (and a box number risen by 15%).

3.7 Modifications of the adaptive strategy

The parameters for the adaptive strategy presented in section 3.5 were derived from various deliberations and extensive experiments. On this basis, the strategy has been provided with the most promising default parameters. In this section some possible modifications of the adaptive strategy are discussed. Since the new strategy was custom-built to provide good results for test set $\mathcal{T}_A$, the modifications are tested for the test set $\mathcal{T}_B$ to shown that the chosen approach is also suited for a larger set of problems.

For every modification the required box number is denoted as $\text{boxes}_{\text{mod}}$ and the time as time_{mod}. For easier comparison we set these values in relation to boxes_{AS} and time_{AS} which name the box number and computation time needed by the standard adaptive strategy (represented in Table 3.5). We compute the ratio

$$\text{boxes}_{\text{ratio}} := \frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{boxes}_{\text{mod}}}{\text{boxes}_{\text{AS}}}$$

for the box numbers and correspondingly for computation time

$$\text{time}_{\text{ratio}} := \frac{1}{\text{number of problems}} \cdot \sum_{\text{problems}} \frac{\text{time}_{\text{mod}}}{\text{time}_{\text{AS}}}.$$

The mentioned modifications that appear to be useful in reducing the computation time for at least some problems can be activated in the Controls-File of SONIC and combined if a user wants to do so.

3.7.1 Restarting

Our tests indicated that restarting the adaptive strategy provides no further advantage with respect to computation time. Thus restarts have not been implemented in the adaptive strategy. However, the box number decreases for some test problems when a restart was used, caused by the higher effort in contracting boxes before subdividing again and proceeding to deeper recursion levels.

3.7.2 Varying parameter α

To prefer finer or coarser extended systems we can adjust variable α in Alg. 3.5. The default value for α is 4. We tested the alternative values 3 (evoking more finer systems, see Table 3.6) and 5 (evoking fewer finer systems, see Table 3.7).

For $\alpha = 3$ the computation time for problem *Eco9* is reduced by 9%, for *Trigexp1* by 14% and for *min-04-07* even by 31%. Then again, the computation time needed for *Trigonometric* increased (by 19%). For $\alpha = 5$ the computation time for *Trigonometric* decreased by 10%. Anyway, none of the options could provide significantly better computation time on average.

As mentioned before, parameter α equals *ComparisonFactorForLargerSystems* in SONIC. The default value is retained as 4.

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	793	0.98	8.1	0.93
G7_gradientsystem	2175	1.00	6.6	1.00
Reactor	575	1.00	12.3	0.99
Chemistry1	6871	0.99	35.2	0.94
Brent7	55569	1.00	29.1	0.99
Eco9	30981	1.00	50.1	0.91
Trigexp1	1407	1.00	27.3	0.86
Trigonometric	45781	1.44	159.3	1.19
Chemistry2	49201	1.00	88.2	1.00
DirectKinematics	10401	1.00	53.8	1.01
DesignProblem9	45409	1.02	112.4	1.00
7erSystem	18737	1.08	79.8	1.00
min-04-07	70215	1.02	967.3	0.69
Chemistry3	2957333	0.99	13761.4	0.99
average		1.04		0.96

Table 3.6: Comparison of AS with the modification $\alpha = 3.0$ to default $\alpha = 4.0$

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	793	0.98	8.8	1.01
G7_gradientsystem	2175	1.00	6.7	1.03
Reactor	577	1.00	12.4	1.00
Chemistry1	6757	0.98	41.4	1.10
Brent7	55029	0.99	30.0	1.02
Eco9	30889	1.00	62.0	1.12
Trigexp1	1407	1.00	31.0	0.98
Trigonometric	22729	0.72	120.9	0.90
Chemistry2	49195	1.00	88.2	1.00
DirectKinematics	10401	1.00	53.8	1.01
DesignProblem9	40319	0.91	118.0	1.05
7erSystem	14657	0.84	83.4	1.05
min-04-07	68423	1.00	1534.1	1.09
Chemistry3	2963875	1.00	13799.3	0.99
average		0.96		1.03

Table 3.7: Comparison of AS with the modification $\alpha = 5.0$ to default $\alpha = 4.0$

3.7.3 Use all extended systems on designated recursion levels

If an extended system $\mathcal{E}^{(i)}$ has not been successful in a previous recursion level, it can happen that on some box $\boldsymbol{x}$, the gauge $\gamma^{\mathcal{E}^{(i)}}$ gets too large in comparison to the gauges $\mathcal{E}^{(\mathrm{j})}$ of the other systems $\mathcal{E}^{(j)}$ $(i \neq j)$. If the gauges for the $\mathcal{E}^{(\mathrm{j})}$ stay small, it can happen that $\mathcal{E}^{(i)}$ is not used for any subboxes of $\boldsymbol{x}$, even if this would be sensible in deeper recursion levels. One may want to prevent this situation. This and the next section are concerned with modifications that ensure that all systems are applied again, including those with high values of $\gamma^{\mathcal{E}^{(i)}}$.

In a first approach, a step size σ_{all} to apply all extended systems is introduced. The condition in of the if-statement beginning in line 2 of Algorithm 3.5 then changes to the following lines.

(recursionlevel ≤ 3) **or**
(recursionlevel mod $\sigma_{\text{all}} = 0$) **or**
($\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}_{\text{coarsest}}} \leq \gamma^{\mathcal{E}_{\text{coarsest}+1}}$) **or**
($\mathcal{E} \neq \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}} < \alpha \cdot \gamma^{\mathcal{E}_{\text{coarsest}}}$)

---

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	785	0.97	9.6	1.11
G7_gradientsystem	2175	1.00	7.3	1.12
Reactor	595	1.03	14.0	1.12
Chemistry1	6753	0.98	39.8	1.06
Brent7	54607	0.99	34.8	*1.19*
Eco9	30845	1.00	58.4	1.06
Trigexp1	1405	1.00	39.3	*1.24*
Trigonometric	28503	0.90	131.8	0.98
Chemistry2	49199	1.00	117.4	*1.33*
DirectKinematics	10193	0.98	65.8	*1.23*
DesignProblem9	40099	0.90	123.7	1.10
7erSystem	14873	0.86	110.3	*1.38*
min-04-07	69567	1.01	1171.2	**0.83**
Chemistry3	2952923	0.99	16539.6	*1.19*
average		0.97		1.14

Table 3.8: Comparison of the adaptive strategy and a modification applying all systems in every 10th recursion level

For a test, parameter σ_{all} was set to 10. Thus it is guaranteed that all extended systems are used in every tenth recursion level. The results are listed in Table 3.8. We see that for the *min-04-07* problem a better computation time is achieved. Even though, for nearly all other problems the computation time stayed

the same or increased slightly. The time needed on average increased by 14%, the box number dropped by 3%. The reason for this behavior lies in the additional extended systems used in the modification. They provide some contraction (and thus reduce the box number) but consume additional computation time.

The modification is assessed as not beneficial for reducing the average computation time.

In SONIC the step size σ_{all} may be chosen by setting the parameter *AllSystemsEveryXRecursionLevels*. In the default options, however, we did not activate the proposed modification.

3.7.4 Reducing $\gamma^{\mathcal{E}}$ if system $\mathcal{E}$ is not applied

Another way to ensure that systems with high $\gamma^{\mathcal{E}}$ are not completely excluded from the computation is to reduce $\gamma^{\mathcal{E}}$ whenever an extended system $\mathcal{E}$ is not applied. In detail, we set $\gamma^{\mathcal{E}}_{\text{new}}$ (line 8 in Alg. 3.5) as

$$\gamma^{\mathcal{E}}_{\text{new}} \leftarrow 0$$

when a system $\mathcal{E}$ was not applied and compute $\gamma^{\mathcal{E}}$ anew using this value.

As shown in Table 3.9, a significant reduction in box number and computation time can only be observed for the *Trigonometric* problem. Nevertheless, on average computation time increases by 34% while the box number decreases slightly because of the more frequent usage of finer systems. In general, the approach does not provide benefits in computation time, but it appears to be useful for individual problems.

Reducing $\gamma^{\mathcal{E}}$ is no default option in SONIC, but can be enabled by setting parameter *AverageWithZeroIfSystemNotUsed* in the Controls-File.

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	785	0.97	10.0	1.16
G7_gradientsystem	2175	1.00	9.5	1.45
Reactor	607	1.05	12.8	1.03
Chemistry1	6477	0.94	44.2	1.18
Brent7	53443	0.96	50.9	1.74
Eco9	30825	1.00	72.2	1.31
Trigexp1	1405	1.00	61.4	1.94
Trigonometric	21575	**0.68**	124.2	**0.93**
Chemistry2	49203	1.00	100.9	1.14
DirectKinematics	10295	0.99	84.0	1.58
DesignProblem9	32543	0.73	157.0	1.40
7erSystem	13807	0.79	134.6	1.69
min-04-07	68697	1.00	1472.9	1.05
Chemistry3	2964831	1.00	15479.8	1.11
average		0.94		1.34

Table 3.9: Comparison of AS and the modification reducing $\gamma^{\mathcal{E}}$ by setting $\gamma^{\mathcal{E}}_{\text{new}} = 0$ when an extended system $\mathcal{E}$ was not applied

3.7.5 Averaging $\gamma^{\mathcal{E}}$ depending on the recursion level

In the next modification, finer extended systems are used more often in the upper recursion levels. Thus we test whether it is sensible to invest more computation time to achieve good contraction in the upper recursion levels. The deliberation is to reduce the number of computed subboxes (and thus the spent computation time) in the following recursion levels.

Our try to achieve this goal was to compute the $\gamma^{\mathcal{E}}$ depending on the recursion level r. To that end we replace the computation of $\gamma^{\mathcal{E}}$ (line 9 in Alg. 3.5) by

$$\gamma^{\mathcal{E}} \leftarrow \tfrac{1}{4} \cdot \left(3 \cdot \gamma^{\mathcal{E}}_{\text{old}} + (\text{time}_{\text{end}} - \text{time}_{\text{start}}) \cdot (\text{ThVol}_{\text{new}}/\text{ThVol}_{\text{old}})^{(1+\frac{4}{r})}\right)$$

and continued in the algorithm using this value. However, it turned out that the modified $\gamma^{\mathcal{E}}$ change nearly equally for all extended systems. Thus the choice of the extended systems stays the same as in the standard version of the adaptive strategy.

A second attempt was made by modifying the comparison of the $\gamma^{\mathcal{E}}$ instead of changing the $\gamma^{\mathcal{E}}$ themselves. The comparison in line 2 of Alg. 3.5 was altered to rely on the recursion level. However, in our tests this change could only reduce the box numbers by a few percent while the computation time stayed nearly the

same.

(recursion level ≤ 3) **or**
$(\mathcal{E} = \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}_{\text{coarsest}}} \leq \gamma^{\mathcal{E}_{\text{coarsest}+1}})$ **or**
$(\mathcal{E} \neq \mathcal{E}_{\text{coarsest}}$ **and** $\gamma^{\mathcal{E}} < \left(\alpha + 10 \cdot \boldsymbol{e}^{-0.1 \cdot r}\right) \cdot \gamma^{\mathcal{E}_{\text{coarsest}}})$

3.7.6 Changing the averaging for the gauges $\gamma^{\mathcal{E}}$

For changing the computation of $\gamma^{\mathcal{E}}$, we can alter parameter β (which is equal to 0.25 by default).

$\gamma^{\mathcal{E}} \leftarrow \beta \cdot \gamma^{\mathcal{E}}_{\text{new}} + (1 - \beta) \cdot \gamma^{\mathcal{E}}$ for some fixed $\beta \in [0, 1]$

In Table 3.10 we give an overview of box number and time ratios for some values of β. As we see, none of the tested values for β can reduce the average computation time significantly. Although, the author observed that the box number and computation time does vary for single problems. But as the problems react differently to the adjusted use of the extended systems, their timings do also change individually. On average, positive and negative effects nearly compensate one another.

β	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\text{boxes}_{\text{ratio}}$	1.38	1.00	1.00	1.00	1.00	1.00	1.01	1.01	1.01	1.01	1.00
$\text{time}_{\text{ratio}}$	1.10	0.97	0.98	1.00	1.02	1.03	1.04	1.08	1.08	1.09	1.08

Table 3.10: Ratios for box number and computation time for different values of parameter β in the adaptive strategy (compared to default value 0.25)

Parameter β is equal to parameter *AveragingGauge* in SONIC.

3.7.7 Volume computation

Another test was conducted with respect to the threshold θ for point intervals used for the computation of the box volumes $\text{ThVol}(\boldsymbol{x})$ since a changed volume computation might influence the contraction rates. In our tests with $\theta = 10^{-20}$, 10^{-16} and 10^{-10} it became apparent that the threshold θ has no impact on the overall performance of the solver. Setting the threshold to the required precision for solution boxes also did not provide an advantage. We could even observe an

increase in computation time by a factor two for problem *Chemistry3*. Hence the default threshold value of 10^{-20} is retained.

3.7.8 Using all extended systems

At last the adaptive strategy was tested with a hierarchy containing all five types of extended systems presented in section 3.1.1. Namely $\mathcal{H}_\mathcal{E}$ was set to consist of the systems $\mathcal{E}_{\text{orig}}$, $\mathcal{E}_{\text{cst}}$, $\mathcal{E}_{\text{cstn}}$, $\mathcal{E}_{\text{lin}}$ and $\mathcal{E}_{\text{full}}$.

The computation time was not improved by adding the additional CSTN and linear system to the hierarchy of extended systems (cf. Tables 3.11 and 3.14). The average box number stayed the same, the average computation time increased by 8%. Especially for the *min-04-07* problem, the computation time increased significantly due to the expensive, but unhelpful application of the two additional extended systems.

Thus the default hierarchy of extended systems, comprising the three systems original, CST and fullsplit, remains unchanged in SONIC.

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	793	0.98	11.6	*1.35*
G7_gradientsystem	2175	1.00	6.6	1.01
Reactor	577	1.00	12.7	1.02
Chemistry1	6919	1.00	41.2	1.10
Brent7	55427	1.00	29.3	1.00
Eco9	30919	1.00	56.6	1.02
Trigexp1	1385	0.99	33.0	1.04
Trigonometric	33423	1.05	136.7	1.02
Chemistry2	49201	1.00	87.8	1.00
DirectKinematics	10401	1.00	54.1	1.02
DesignProblem9	44123	0.99	113.4	1.01
7erSystem	16869	0.97	80.1	1.01
min-04-07	68267	0.99	2185.1	*1.55*
Chemistry3	2974411	1.00	13827.0	1.00
average		1.00		1.08

Table 3.11: Comparison of the adaptive strategy using $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{cstn}}, \mathcal{E}_{\text{lin}}, \mathcal{E}_{\text{full}})$ to default using $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$

However, this result does not imply that other extended systems cannot be beneficial for individual problems. Variations of the proposed adaptive strategy or the usage of other extended systems might prove to be advantageous. Particularly beneficial may be the construction of extended systems that are fitted for a given problem. Still, until now we did not find an appropriate strategy to do

so because the dependence of contraction to the extended system is obscured by preconditioning and (for some contractors) linearization.

3.7.9 Using other extended systems

It is possible to construct further extended systems (and to use them in SONIC). However, a good strategy is needed to do so. As mentioned before, Willems constructed various extended systems in [Wil04] and rejected the ones not listed in our presentation as not beneficial.

Some further experiments were conducted with the new adaptive strategy, e.g., with a modified hierarchy $\mathcal{H}_{\mathcal{E}}$. For these tests the original and the fullsplit system are combined with one or two other systems.

In another test it was tried to decide a priori and individually for each problem, which extended systems should be applied. If we would assume the success and the computational costs of an extended system to depend only its variable number $|\mathrm{Var}(\mathcal{E})|$, the decision could be made depending on this parameter. We conducted some tests using three extended systems. Besides the original and fullsplit system, one further extended system was used for each test problem. This system $\mathcal{E}$ was chosen to minimize

$$\left(|\mathrm{Var}(\mathcal{E})| - \frac{|\mathrm{Var}(\mathcal{E}_{\mathrm{orig}})| + |\mathrm{Var}(\mathcal{E}_{\mathrm{full}})|}{2}\right)^2$$

so the variable number approximates the mean of the variable number of the original and the fullsplit system.

Because none of the extended systems implemented in SONIC may have the desired number of variables, for a further test an artificial system $\mathcal{E}$ with the prescribed number of variables

$$|\mathrm{Var}(\mathcal{E})| = \left\lfloor \frac{|\mathrm{Var}(\mathcal{E}_{\mathrm{orig}})| + |\mathrm{Var}(\mathcal{E}_{\mathrm{full}})|}{2} \right\rfloor$$

was constructed (using every second variable in $\mathcal{E}_{\mathrm{full}}$.)

None of the tests showed significant improvement for our test set $\mathcal{T}_B$, thus different extended systems suit different problems. However, the tests supported our assumption that the contraction success of an extended system does not only depend on its number of variables, but also on the structure of the system.

3.8 Survey of the utilization of the extended systems

For assessing the usage of the extended systems for different problems, it is interesting to see, how much computation time is spent on the individual systems. Because, for general extended systems, the preconditioned extended Newton method is the contractor with the highest influence on the computation time, emphasis will be laid on this contractor.

The given tables show the number of calls for the extended Newton method, how often it can discard or contract a box. Also given is the variable number of each system, the computation time spent for the Newton method in each extended system and the overall running time and the box number needed for each problem. The results are stated for BIH (Table 3.12) and AS (Table 3.13). In both cases the default hierarchy of extended system $\mathcal{H}_\mathcal{E} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$ is applied. A third table states the results when using five systems in the adaptive strategy (Table 3.14).

In the studies employing the adaptive strategy it is anticipated that the extended systems that have been invoked often provide efficient contractions. Still, for individual problems it can be even more beneficial to employ other systems. However, all tests concerning the adaptive strategy showed that the heuristic provides very good decisions on average.

The tables again indicate that the CST system is important for problem *Trigonometric* (corresponding to the results of the setting "only $\mathcal{E}_{\text{orig}}$" in Table 3.1). For *G7_gradientsystem* we see that the extended Newton method does never discard or contract a box and that the adaptive strategy "learns" this within the computation and does not start $\mathcal{E}_{\text{cst}}$ and $\mathcal{E}_{\text{full}}$ for all boxes as the box-intern hierarchy does.

system		min-04-06	G7_gradientsystem	Reactor	Chemistry1	Brent7	Eco9	Trigexp1	Trigonometric	Chemistry2	DirectKinematics	DesignProblem9	7erSystem	min-04-07	Chemistry3
$\mathcal{E}_{\text{orig}}$	variables	15	7	29	13	7	8	50	10	6	11	9	7	21	13
	called	581	1087	213	3451	26026	20847	716	18447	49182	7112	19439	11486	35048	2903774
	discarded boxes	186	0	22	444	25	4583	12	824	4	1701	1566	3915	316	1458
	contracted boxes	56	0	191	2157	5286	12807	370	2980	49157	4875	12936	7571	2021	2900959
	time	1.5	0.4	14.9	12.5	8.8	23.7	12.5	20.0	29.8	18.1	24.6	21.7	289.1	9522.1
$\mathcal{E}_{\text{cst}}$	variables	81	24	38	19	-	-	99	23	15	34	16	14	102	19
	called	395	1087	178	2945	-	-	704	17163	26649	5112	17537	7538	34722	2437696
	discarded boxes	0	0	0	38	-	-	0	4536	4	426	1490	1160	633	170
	contracted boxes	9	0	175	2101	-	-	52	12004	26643	3527	12575	5411	2679	2416131
	time	0.5	2.1	10.3	10.2	-	-	12.4	54.5	55.5	95.8	63.7	30.1	152.7	10499.9
$\mathcal{E}_{\text{full}}$	variables	280	79	171	82	61	65	832	86	52	184	91	430	359	82
	called	392	1087	168	2883	25978	15438	694	7693	26644	4619	15643	6087	34058	2437337
	discarded boxes	0	0	0	7	1	170	0	32	0	21	965	300	1050	116
	contracted boxes	0	0	114	761	12520	7935	668	5599	26620	2167	6537	5638	4099	1773511
	time	2.3	3.4	2.1	8.3	50.2	36.0	71.2	17.5	38.7	42.3	56.6	359.6	641.1	7978.5
	total boxes	785	2175	549	5889	51851	30109	1385	18569	49195	9157	27949	11523	68185	2959459
	total time	12.2	12.6	27.9	48.9	79.1	85.8	105.6	132.7	185.9	189.8	195.0	446.3	1894.4	32902.3

Table 3.12: Usage and success for $\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$ in BIH

system		min-04-06	G7_gradientsystem	Reactor	Chemistry1	Brent7	Eco9	Trigexp1	Trigonometric	Chemistry2	DirectKinematics	DesignProblem9	7erSystem	min-04-07	Chemistry3
$\mathcal{E}_{\text{orig}}$	variables	15	7	29	13	7	8	50	10	6	11	9	7	21	13
	called	64	1087	205	3903	27627	21206	517	20670	49178	7905	28291	16443	131	2923072
	discarded boxes	22	0	18	463	20	4899	12	3576	2	2354	5251	7522	0	1758
	contracted boxes	7	0	187	2259	6285	12853	100	4171	49155	5062	18787	8921	3	2919995
	time	0.2	0.4	11.1	13.0	9.5	23.8	6.4	20.2	29.7	19.6	36.9	28.7	1.0	9513.1
$\mathcal{E}_{\text{cst}}$	variables	81	24	38	19	-	-	99	23	15	34	16	14	102	19
	called	559	18	15	392	-	-	684	13216	5	10	280	1760	35140	579
	discarded boxes	164	0	0	1	-	-	0	4657	0	0	25	333	919	3
	contracted boxes	49	0	14	296	-	-	73	8276	5	2	209	1248	6203	484
	time	0.7	0.0	0.4	1.3	-	-	12.5	45.7	0.0	0.1	1.0	7.0	168.1	2.5
$\mathcal{E}_{\text{full}}$	variables	280	79	171	82	61	65	832	86	52	184	91	430	359	82
	called	148	9	18	457	197	2434	34	3722	22	33	597	6	26316	836
	discarded boxes	0	0	0	0	0	7	0	422	0	0	93	0	959	2
	contracted boxes	3	0	8	133	120	1413	31	2777	3	1	333	3	3708	218
	time	0.6	0.0	0.2	1.4	0.4	5.6	4.1	10.6	0.0	0.1	2.0	0.3	502.8	2.5
	total boxes	809	2175	577	6907	55421	30929	1405	31751	49201	10401	44497	17375	68699	2976919
	total time	8.6	6.6	12.4	37.5	29.3	55.2	31.7	133.9	88.2	53.3	112.2	79.7	1408.9	13886.7

Table 3.13: Usage and success for $\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{full}})$ in AS

system		min-04-06	G7_gradientsystem	Reactor	Chemistry1	Brent7	Eco9	Trigexp1	Trigonometric	Chemistry2	DirectKinematics	DesignProblem9	7erSystem	min-04-07	Chemistry3
$\mathcal{E}_{\text{orig}}$	variables	15	7	29	13	7	8	50	10	6	11	9	7	21	13
	called	64	1087	205	3895	27631	21203	649	24617	49178	7906	28060	15861	194	2920100
	discarded boxes	22	0	18	448	20	4900	12	4773	2	2354	5162	7101	2	1763
	contracted boxes	7	0	187	2260	6278	12850	108	4902	49155	5063	18619	8759	8	2916989
	time	0.2	0.4	11.1	12.6	9.4	23.4	7.8	22.2	29.5	19.5	36.2	26.9	1.5	9358.2
$\mathcal{E}_{\text{cst}}$	variables	81	24	38	19	-	-	99	23	15	34	16	14	102	19
	called	557	21	15	462	-	-	664	12553	5	10	704	2581	35030	843
	discarded boxes	164	0	0	1	-	-	0	4573	0	0	84	499	828	5
	contracted boxes	49	0	14	368	-	-	37	7903	5	2	557	1774	6193	557
	time	0.7	0.0	0.4	1.6	-	-	11.3	39.2	0.0	0.1	2.4	9.5	167.2	3.0
$\mathcal{E}_{\text{cstn}}$	variables	228	79	145	76	48	56	589	81	52	158	91	262	285	76
	called	335	9	18	681	17	8	47	1214	5	126	10	6	32554	988
	discarded boxes	0	0	0	1	0	0	0	155	0	0	1	0	1122	12
	contracted boxes	13	0	7	156	3	1	40	1058	5	42	4	6	3559	212
	time	1.1	0.0	0.1	1.8	0.0	0.1	4.0	6.6	0.0	0.5	0.1	0.3	354.9	2.7
$\mathcal{E}_{\text{lin}}$	variables	185	72	76	59	28	30	344	30	38	73	38	50	230	59
	called	308	8	18	486	8	778	8	580	20	37	11	6	25835	777
	discarded boxes	0	0	0	0	0	6	0	25	0	0	0	0	28	0
	contracted boxes	2	0	3	32	0	297	1	386	2	1	2	0	421	53
	time	1.2	0.0	0.2	1.3	0.0	1.9	0.6	1.7	0.0	0.1	0.1	0.1	363.8	2.2
$\mathcal{E}_{\text{full}}$	variables	280	79	171	82	61	65	832	86	52	184	91	430	359	82
	called	178	8	18	433	157	2322	6	2719	21	33	577	6	24854	739
	discarded boxes	0	0	0	0	0	4	0	269	0	0	67	0	9	0
	contracted boxes	0	0	4	18	101	1194	2	1923	1	2	358	0	120	33
	time	0.8	0.0	0.2	1.2	0.3	5.3	0.6	6.8	0.0	0.1	2.0	0.3	489.2	2.2
	total boxes	793	2175	577	6919	55427	30919	1385	33423	49201	10401	44123	16869	68267	2974411
	total time	11.6	6.6	12.7	41.2	29.3	56.6	33.0	136.7	87.8	54.1	113.4	80.1	2185.1	13827.0

Table 3.14: Usage and success for $\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\text{orig}}, \mathcal{E}_{\text{cst}}, \mathcal{E}_{\text{cstn}}, \mathcal{E}_{\text{lin}}, \mathcal{E}_{\text{full}})$ in BIH

3.9 Choosing the variables considered by the Newton method

In addition to the suitable selection of the extended systems to be applied, there surely exist further ways to reduce the overall computation time. One important way is to reduce the time spent on the contractors.

Our tests confirmed that it is not sensible to run the extended Newton method and C^W-LP-preconditioner for all variables of the extended systems. Table 3.15 compares the box-intern hierarchy with the default step sizes and $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$. For most problems we observe a considerable decrease in box number. We can thus infer that due to the step sizes we "miss chances" to contract boxes. (The slight increase in box number for *Chemistry2* is most probably caused by an unfavorable subdivision.) Nevertheless we also see that the computation time increases enormously for some of the problems, since we have to conduct a higher number of evaluations of the extended Newton method and of the time-consuming C^W-LP-preconditioner.

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	563	**0.72**	12278.3	*1018.35*
G7_gradientsystem	2175	1.00	110.5	*8.80*
Reactor	529	0.96	729.5	*31.85*
Chemistry1	3361	**0.67**	1591.2	*32.46*
Brent7	23579	**0.45**	2659.6	*33.11*
Eco9	28129	0.93	9668.3	*111.91*
Trigexp1	435	**0.35**	10294.1	*106.21*
Trigonometric	7525	**0.41**	348.1	*2.66*
Chemistry2	53279	*1.08*	2107.3	*11.21*
DirectKinematics	5455	**0.58**	34046.2	*186.97*
DesignProblem9	11027	**0.39**	3032.63	*15.89*

Table 3.15: Box-intern hierarchy with $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$ compared to results for default step sizes (three problems have not been computed due to their high computational costs)

So the step sizes $\sigma_{var}(\mathcal{E})$ and $\sigma_{C^W}(\mathcal{E})$ improve the overall branch-and-bound algorithm with respect to time issues. However, the subset of variables $V^{\mathcal{E}}$ for which the interval Newton method is considered is predefined and does not take into account the structure of individual problems, only the number of variables $|\text{Var}(\mathcal{E})|$. To alleviate the problem of always considering the same subset $V^{\mathcal{E}}$ for all boxes, one can elect the first variable to be included in $V^{\mathcal{E}}$ randomly in $\{|\text{Var}(\mathcal{E})|, \ldots, |\text{Var}(\mathcal{E})| - \sigma_{var}(\mathcal{E}) + 1\}$ (in Alg. 3.2 line 5) as suggested in [Bee06]. However, we could just find one test problem for which this option had any

influence (factor 0.90 in computation time for *7erSystem*). To achieve deterministic behavior the author also tested to traverse the first variable through $\{|\mathrm{Var}(\mathcal{E})|, \ldots, |\mathrm{Var}(\mathcal{E})| - \sigma_{var}(\mathcal{E}) + 1\}$ repeatedly in all calls of the extended Newton method, but this option was only as successful as the random choice.

The author wants to propose another approach to select the subset $V^{\mathcal{E}}$. This approach shall not rely on a fixed step size predefined by the size of the extended systems but is meant to select the variables in $V^{\mathcal{E}}$ based on the properties of the given problem and system $\mathcal{E}$. (The sets $V^{\mathcal{E}}$ can be determined when building the representation of the systems $\mathcal{E}$ in the computer. Thus the variables in $V^{\mathcal{E}}$ can be chosen based on the structure of the constraint system and $V^{\mathcal{E}}$ only has to be assigned once.)

In SONIC we yet provide the possibility to choose the variables in $V^{\mathcal{E}}$ based on system properties .

The purpose of the proposed "selection by structure" is to consider only "promising" variables. Our vision would be to select only those variables that provide (sufficient) contraction success and to save computation time by not considering any other variables. However, until now we have no general strategy to choose $V^{\mathcal{E}}$ prior to computations. We even do not know yet, whether such a selection is possible at all. No selection may, e.g., be possible if the extended systems provide equal, but only minor contraction (e.g. with $\kappa = 1.001$) on all variables.

We nevertheless want to give some results for tested choices of $V^{\mathcal{E}}$. For our tests we applied the selection of set $V^{\mathcal{E}_{\mathrm{full}}}$ for the fullsplit system, with and without using the step sizes. We began our considerations with a study of the contraction success in the constraints measured in a run with $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$ for all systems $\mathcal{E}$ in $\mathcal{H}_{\mathcal{E}} = (\mathcal{E}_{\mathrm{orig}}, \mathcal{E}_{\mathrm{cst}}, \mathcal{E}_{\mathrm{full}})$. These results indicated that considering variables defined by elementary constraints with operators $+, -, \cdot, /$ and constraints like $x_i = c$ for some constant c seldom result in good contraction. We therefore excluded intermediate variables defined by such constraints from $V^{\mathcal{E}_{\mathrm{full}}}$. The results of our tests for $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$ and the default step sizes can be found in tables 3.16 and 3.17.

Table 3.16 shows that our choice of $V^{\mathcal{E}_{\mathrm{full}}}$ did indeed reduce the computation time for $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$ (compared to Table 3.15)—but not enough to keep up with the default step sizes. If we combine the step sizes and our selection $V^{\mathcal{E}_{\mathrm{full}}}$, the results are beneficial for all tested problems. (For the combination we apply the step sizes as usual. Subsequently we check whether the variable to be considered next in the interval Newton method is contained in $V^{\mathcal{E}_{\mathrm{full}}}$. If not, we step to the very next variable. If we find a variable v in $V^{\mathcal{E}_{\mathrm{full}}}$ we apply the contractor and apply the step sizes again counting from v.)

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	563	0.72	3509.75	291.09
G7_gradientsystem	2175	1.00	35.28	2.81
Reactor	609	1.10	223.06	9.74
Chemistry1	3583	0.72	382.33	7.80
Brent7	49515	0.95	1461.60	18.19
Eco9	30091	1.00	894.25	10.35
Trigexp1	923	0.74	13018.20	134.32
Trigonometric	7523	0.41	186.44	1.43
Chemistry2	53279	1.08	2107.28	11.21
DirectKinematics	6761	0.72	6579.46	36.13
DesignProblem9	18611	0.67	1012.95	5.31

Table 3.16: BIH with selection of $V^{\mathcal{E}_{\text{full}}}$ with $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{CW}(\mathcal{E}) = 1$ compared to default (using step sizes and no selection of $V^{\mathcal{E}_{\text{full}}}$)

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	785	1.00	10.9	0.91
G7_gradientsystem	2175	1.00	11.9	0.94
Reactor	553	1.00	22.9	1.00
Chemistry1	4983	1.00	47.8	0.98
Brent7	52633	1.02	77.4	0.96
Eco9	30233	1.00	69.4	0.80
Trigexp1	985	0.79	83.8	0.86
Trigonometric	18599	1.00	131.6	1.01
Chemistry2	-	-	-	-
DirectKinematics	9377	1.00	170.4	0.94
DesignProblem9	28841	1.03	181.9	0.95
7erSystem	12635	1.10	151.6	0.34
min-04-07	68185	1.00	1601.8	0.86
Chemistry3	-	-	-	-

Table 3.17: BIH with selection of $V^{\mathcal{E}_{\text{full}}}$ with default step sizes $\sigma_{var}(\mathcal{E})$ compared to default (using step sizes and no selection of $V^{\mathcal{E}_{\text{full}}}$)

Other selections of $V^{\mathcal{E}}$ combining information of more than one constraint have not yet proved to be helpful in improving the computational costs of our solver. However, they would allow for more sophisticated selection strategies for $V^{\mathcal{E}}$.

For *min-04-07* we also tested to choose $V^{\mathcal{E}_{\text{full}}}$ "by hand" based on the a posteriori evaluation of contraction success over all variables. But it became obvious that this selection still consumed more computation time than BIH or AS with step sizes.

For further considerations one might also introduce $V^{C^W} \subseteq V^{\mathcal{E}}$ as the set of variables for which the interval Newton method is combined with the optimal preconditioner. Moreover, one could not only use the proposed set $V^{\mathcal{E}}$ to choose the considered variables for the interval Newton method but also other expensive contractors. We could even again think of an adaptive method that adds and removes variables from $V^{\mathcal{E}}$ relying on the success of the contractors for each variable. Still, for extended systems $\mathcal{E}$ with many variables this adaption may itself require vast amounts of computational cost if starting with $V^{\mathcal{E}} = \text{Var}(\mathcal{E})$ since many variables are considered in the beginning. (Experience showed that calculating the extended Newton method on all systems with the C^W-LP-preconditioner for one single box can already cause higher computation time than the overall solution of a problem with BIH or AS using step sizes.)

Another method would be to select a subset $V^{\mathcal{E}}$ separately for each box, relying on information as provided by the constraints or other system values like the Jacobian matrix. Yet, the author assumes that this approach would most probably be too costly.

3.10 Improvements for preconditioners

Especially interesting for a reduction of the overall computation time are ways to reduce the costs for the computation of preconditioners and the multiplication with them. For most problems these operations cause a large portion of the computation time. On average these costs amount up to about 30% for the adaptive strategy and about 50% of the overall computation time for the box-intern hierarchy. Even worse, for individual problems, preconditioning consumes up to 90% of the computation time.

We see different options to reduce the time spent on preconditioning. A first method, of course, is to reduce the computation time spent for computing a single preconditioner, especially the expensive C^W-LP-preconditioner. Possibilities for doing so were discussed in section 2.3.5.

A further possibility is to deactivate all or at least some types of preconditioners for all extended systems. But mind, this would affect the strength of the contractors. It is thus probable not to result in a reduction but in an overwhelming increase of computation time.

Beyond that, the author proposes a third way to save computation time. The basic idea for this approach is to "recycle" preconditioner rows, thus saving the computation time for computing them anew. (A resembling approach, reusing preconditioning information for sibling boxes in the branch-and-bound algorithm, was also mentioned in [KHN91].)

Note first that every preconditioner computed for a box $\boldsymbol{x}$ can also be applied as a preconditioner for all other boxes. Thus, if we store already computed preconditioner rows, we get a preconditioner with computational costs next to naught. Of course, it may not be as good as a preconditioner computed directly for the respective subbox. For a promising strategy using "recycled preconditioners row", one needs to weigh the advantages of reusing information and the presumably better contraction provided by "new" preconditioners.

An enhancement of this strategy can be reached by storing information about the success of the preconditioners or of single preconditioner rows. With this additional information, the application of preconditioners could be controlled by a strategy "based on experience" similar to the adaptive strategy for selecting extended systems. One could think of storing "successful" preconditioner rows and computing a new preconditioner row only if no row has been stored so far or if the previously saved preconditioner row did not provide sufficient contraction.

Our tests indicated that the usage of "recycled preconditioners" can indeed provide benefits in computation time, see Table 3.18. On the other hand, for attaining these benefits, one has to store the preconditioner rows. Since we again assume that it is best to use the preconditioner rows computed for parent- or ancestorboxes, the preconditioner rows have to be processed and stored individually for every box. The efforts in storing and processing the preconditioner rows grows in correspondence with the number of preconditioners to save and the variable number in the considered systems. Because of this downside, the author refrained from the option of saving preconditioning information for the time being.

problem	$\text{boxes}_{\text{mod}}$	$\text{boxes}_{\text{ratio}}$	time_{mod}	$\text{time}_{\text{ratio}}$
min-04-06	563	1.00	8326.69	0.68
G7_gradientsystem	2175	1.00	110.56	1.00
Reactor	503	0.95	377.93	0.52
Chemistry1	3623	1.08	168.39	0.11
Brent7	43479	1.84	262	0.10
Eco9	29367	1.04	207.27	0.02
Trigexp1	435	1.00	8556.61	0.83
Trigonometric	10355	1.38	194.67	0.56
Chemistry2	53481	1.00	781.83	0.37
DirectKinematics	6761	1.24	3447.7	0.10
DesignProblem9	20973	0.79	643.34	0.65

Table 3.18: Box-intern hierarchy with saved preconditioner rows for $\sigma_{var}(\mathcal{E}) = 1$ and $\sigma_{C^W}(\mathcal{E}) = 1$ compared to results with newly computed preconditioner rows in all steps (three problems have not been computed due to their high computational costs)

3.11 Interaction of branching and bounding strategies

At this point we want to append some more test results for the different subdivision strategies as presented in Table 2.1. This time however, we present the results in combination with the adaptive strategy, see Table 3.19 and compare to Table 2.1.

We can see that the subdivision strategies do not provide equal ratios for the different strategies. In particular, the ratio for SubMinNew = 16 does not result in reduced average computation time for the adaptive strategy. We ascribe this to the reduced similarity of functions between the box and its subboxes when we partition into many small boxes. Note that, although the computed averaged ratios are higher, the absolute running times for the adaptive strategy are most often still smaller than for the box-intern hierarchy.

The combination of extended systems and subdivision strategies is just one further example for the subtle interactions of methods and parameter settings in the solver. This interplay also depends on the considered problem and cannot often be foretold. This makes it all the more important for us to find promising subdivision strategies as well as techniques such as the adaptive strategy for applying contractors on the extended systems that provide good contraction without being a burden with regard to computation time.

setting		$\text{boxes}_{\text{ratio}}$	$\text{time}_{\text{ratio}}$
SubMinNew	2	1.00	1.00
SubMinNew	3	1.05	0.88
SubMinNew	4	1.11	0.88
SubMinNew	5	1.23	0.90
SubMinNew	8	1.56	0.97
SubMinNew	16	2.32	1.16
η	0.1	8.29	8.05
η	0.2	2.24	2.27
η	0.3	1.31	1.29
η	0.4	5.64	5.59
η	0.45	1.17	1.15
η	0.49	96.32	93.42
η	0.5	1.00	1.00
η	0.51	0.87	0.87
η	0.55	1.08	1.08
η	0.6	2.57	2.24
η	0.7	1.44	1.24
η	0.8	2.25	1.94
η	0.9	6.62	5.20
trisection			
for all boxes		1.58	1.17
once per direction		0.99	0.98

Table 3.19: Comparison of different subdivision strategies in box number and computation time

Chapter 4

Verification

All truths are easy to understand once they are discovered;
the point is to discover them.

Galileo Galilei

The branch-and-bound algorithm, yields a solution list after repeated subdivision and application of contractors. The elements of the solution list, the solution boxes, are guaranteed to cover all solutions of a (real-valued) system of nonlinear equations in a given startbox.

What is not ensured is that every solution box corresponds to exactly one solution. It is still possible that the system has multiple solutions in a solution box $\boldsymbol{x}$ or none at all. A solution box containing multiple solutions can occur, when the precision requirements are not sharp enough to enclose the solutions by separate boxes or because we deal with a manifold of solutions. That a box $\boldsymbol{x}$ is incorporated into the solution list although it does not contain a solution, can happen if the box cannot be discarded in the branch-and-bound algorithm because the function enclosure was overestimated ($0 \in \boldsymbol{f}(\boldsymbol{x})$ but $0 \notin \text{range}_f(\boldsymbol{x})$). Yet, even if we can compute the ranges of the function components exactly and they contain zero ($0 \in \text{range}_{f_i}(\boldsymbol{x})$ for all $i \in \{1, \ldots, n\}$), this is no sufficient condition for box $\boldsymbol{x}$ to contain a solution. There exists no solution if the function components vanish at different points ($f_i(x^{(1)}) = 0$ and $f_j(x^{(2)}) = 0$ for $x^{(1)}, x^{(2)} \in \boldsymbol{x}$ and $i, j \in \{1, \ldots, n\}$, but $x^{(1)} \neq x^{(2)}$ and $f_i(x^{(2)}) \neq 0$, $f_j(x^{(1)}) \neq 0$).

Nevertheless, we have several options to prove the existence or uniqueness of a solution in a given box with the help of interval analysis—in exact interval analysis as well as when transferring interval concepts to machine numbers.

Definition 4.1
If we can prove the existence or uniqueness of a solution in a given box $\boldsymbol{x}$ *we say* $\boldsymbol{x}$ *is* verified. *The* verification *of solutions is done by* verification methods. *The implementation of a verification method is referred to as the corresponding* verification test. *A verification method or test is called* successful *for box* $\boldsymbol{x}$ *if it can verify a solution in* $\boldsymbol{x}$.

Verification tests in interval analysis rely on checking sufficient conditions based on function enclosures. However, the tests can fail due to overestimation although it would be possible to verify a solution in exact arithmetic. Thus the success of verification tests relies heavily on the quality of the employed function enclosures.

All our verification tests are restricted to bounded boxes, verification for arbitrary sets is not considered. In particular, half-bounded solution boxes are excluded from verification. The verification methods presented in section 4.3 can further only be applied to square systems. Possibilities to handle non-square systems are discussed prior to these tests in section 4.2.

Note that every solution box is considered separately to verify solutions. Verification tests can thus be applied every time a new solution box is added to the solution list or after the branch-and-bound algorithm is completed. The advantages of both options will be discussed in section 4.5.2.

4.1 Definitions

Some definitions are needed when we want to speak about verification. Note thatall of them are applicable since we only conduct verification tests on bounded boxes.

4.1.1 Preconditioning

For attaining sharper function enclosures, we again make use of preconditioners. A preconditioner in this case is a nonsingular matrix $C \in \mathbb{R}^{n\times n}$ and yields an affinely scaled preconditioned function $g(x) = C \cdot f(x)$. Preconditioning can be dismissed by setting C to the identity matrix. A second preconditioner appropriate for all verification tests is the inverse midpoint preconditioner. For the Borsuk test one further has the freedom to apply a specially designed preconditioner (see section 4.3.3).

Note that, in contrast to the preconditioners for contractors, we only allow nonsingular preconditioners in verification because we need the functions f and g to have the same zeros. Hence all verification tests can be applied to f as well as to g to prove zeros of f and the tests can be formulated directly for the preconditioned function g.

4.1.2 Epsilon-inflation

The so-called *epsilon-inflation* realizes a slight inflation of an interval or a box. In verification, epsilon-inflation can be most useful due to its influence on the function enclosures evaluated in the verification tests. In addition, it was constructed to have only small enlarging impact on the boxes to be verified. This is important because we want verification to yield boxes as small as possible.

Definition 4.2 (Epsilon-inflation)
For a given $\epsilon \in \mathbb{R}_+$ the epsilon-inflation of an interval $\boldsymbol{x}$ yields

$$\boldsymbol{x}_\epsilon := \begin{cases} \boldsymbol{x} + [-\epsilon, \epsilon] & \text{if } \operatorname{width}(\boldsymbol{x}) = 0 \\ (1+\epsilon) \cdot \boldsymbol{x} - \epsilon \cdot \boldsymbol{x} & \text{otherwise.} \end{cases}$$

For boxes epsilon-inflation is applied componentwise.

Further competing definitions and applications of the epsilon-inflation can, e.g., be found in [May95]. For any implementation of an epsilon-inflation in combination with interval analysis, the real values $(1+\epsilon)$ and ϵ have to be enclosed by intervals.

When verifying an inflated box, one cannot draw conclusions about the uninflated box. Thus we have to replace the smaller box we started with by a larger, verified one in the list of solution boxes. Note furthermore that the usage of epsilon-inflation can lead to solution boxes with intersecting interior.

In SONIC the value ϵ can be set in the Controls-File (as parameter *Verification-EpsInflationValue*).

4.1.3 Facets and subfacets

This section provides definitions needed for some verification tests.

Definition 4.3 (Facets)
For a bounded box $\boldsymbol{x} \in \mathbb{IR}^n$, we define n pairs of opposite facets. *For that purpose the components of the box are replaced by their infimum or supremum one at a time. The facets are denoted as*

$$\boldsymbol{x}^{i,+} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \overline{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)^T \in \mathbb{IR}^n \text{ and}$$
$$\boldsymbol{x}^{i,-} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \underline{x}_i, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)^T \in \mathbb{IR}^n.$$

for $1 \leq i \leq n$. Given this definition, the topological boundary $\partial\boldsymbol{x}$ of a box $\boldsymbol{x}$ can be represented as the union of its $2n$ facets

$$\partial\boldsymbol{x} = \bigcup_{i=1}^{n} \left(\boldsymbol{x}^{i,+} \cup \boldsymbol{x}^{i,-} \right).$$

To reduce overestimation of the function enclosures, the facets are dissected. Any member of a covering of a facet is called a *subfacet*. All tests formulated for facets can also be used checking the same conditions on subfacets (according to the enclosure property in (1.8)).

Definition 4.4 (Subfacets)
Consider an arbitrary component $i \in \{1, \ldots, n\}$ *of a box* $\boldsymbol{x} \in \mathbb{IR}^n$. *For two integers* $K_i, L_i \in \mathbb{N}$ *the* subfacets *of the facets* $\boldsymbol{x}^{i,+}$ *and* $\boldsymbol{x}^{i,-}$ *are denoted by* $\boldsymbol{x}^{i,+,k}$ *with* $k \in \{1, \ldots, K_i\}$ *and* $\boldsymbol{x}^{i,-,l}$ *with* $l \in \{1, \ldots, L_i\}$.
Subfacets have to fulfill the following properties

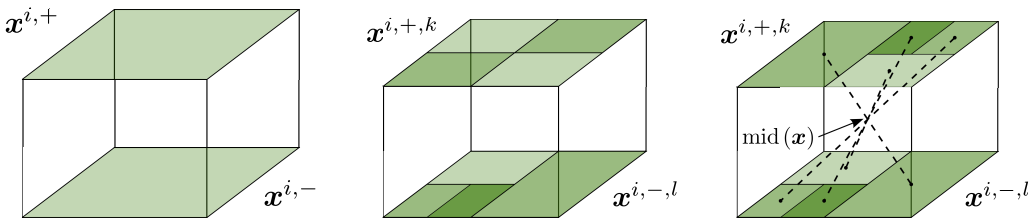
- $\boldsymbol{x}^{i,+,k}, \boldsymbol{x}^{i,-,l} \in \mathbb{IR}^n$ *for all* $k \in \{1, \ldots, K_i\}$ *and* $l \in \{1, \ldots, L_i\}$
- *the union of the subfacets is the facet itself*

$$\boldsymbol{x}^{i,+} = \bigcup_{k \in \{1,\ldots,K_i\}} \boldsymbol{x}^{i,+,k}, \quad \boldsymbol{x}^{i,-} = \bigcup_{l \in \{1,\ldots,L_i\}} \boldsymbol{x}^{i,-,l}.$$

Subfacets are called symmetric *if they are pairwise symmetric with respect to the midpoint of the box (for all* $y \in \boldsymbol{x}^{i,+,k}$ *exists* $z \in \boldsymbol{x}^{i,-,k}$ *such that* $z = 2 \cdot \mathrm{mid}\,(\boldsymbol{x}) - y$ *and vice versa).*

If we do not need symmetry of subfacets, the subdivision of the facets can be done independently and can result in different numbers of subfacets. If subfacets are required to be pairwise symmetric, this yields the same number of subfacets on opposite facets. For attaining symmetric facets, we have to subdivide in the same direction and to take care of the choice of the subdivision points (especially in an implementation, cf. section 4.5.5).

Mostly one can additionally demand the subfacets to be disjoint up to their relative boundaries, so the subfacets form a subdivision of the facet. The only case for which this requirement cannot be fulfilled in the upcoming verification tests is the implementation of symmetric subfacets. Methods to subdivide facets into subfacets will be discussed in section 4.5.5 and 4.5.6. At this point we only want to introduce the parameters *MaxSub* for the maximum number of subfacets considered per facet and *MaxDepth* for the maximum subdivision depth for subfacets as those will be needed to note the upcoming algorithms.



(a) Facets in component i (b) Asymmetric subfacets (c) Symmetric subfacets

Figure 4.1: Facets and subfacets for a 3-dimensional box $\boldsymbol{x}$

4.1.3.1 Facet-centered function enclosure

If we use facets and want to evaluate function enclosures on them, we can utilize a function enclosure similar to the mean value form discussed in section 1.8.2. Herein a point within the facet is used as center.

For every facet $\boldsymbol{x}_{\text{facet}} \in \{\boldsymbol{x}^{i,+}, \boldsymbol{x}^{i,-}\}$ a center $\widetilde{x}_{\text{facet}} \in \boldsymbol{x}_{\text{facet}}$ has to be determined. Furthermore, an interval slope vector $\boldsymbol{s}_{\text{facet}}$ fulfilling

$$g_i(x) - g_i(\widetilde{x}_{\text{facet}}) \in \boldsymbol{s}_{\text{facet}} \cdot (x - \widetilde{x}_{\text{facet}}) \quad \text{for all } x \in \boldsymbol{x}_{\text{facet}}$$

is needed for each facet. With these settings we can state

$$\boldsymbol{g}_i(\boldsymbol{x}_{\text{facet}}) \subseteq g_i(\widetilde{x}_{\text{facet}}) + \boldsymbol{s}_{\text{facet}} \cdot (\boldsymbol{x}_{\text{facet}} - \widetilde{x}_{\text{facet}})$$

for every facet. This enclosure is called a *facet-centered function enclosure*. We choose every center $\widetilde{x}_{\text{facet}}$ as the midpoint of the facet in question. The slope vector $\boldsymbol{s}_{\text{facet}}$ is taken as the ith row of $g'(\boldsymbol{x}_{\text{facet}}) = C \cdot \boldsymbol{f}'(\boldsymbol{x}_{\text{facet}})$.

4.2 Verification for non-square systems

As emphasized before, the verification tests to be described in section 4.3 are only applicable for square systems of nonlinear equations.

For non-square systems we can only verify on appropriate square subsystems. However, precautions have to be taken when conveying information about solutions of the square system to the non-square system.

4.2.1 Underdetermined systems

Given an underdetermined system, a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with fewer constraints than variables ($n > m$) has to be considered. The usual approach to handle underdetermined systems is to fix $n - m$ variables of the system and to verify the remaining square system. "Fixing a variable" means that the variable is handled

rather as a parameter (with interval or point value) than as variable in the square subsystem.

We are interested in fixing the variables in which the system is least sensitive. Thus a subsystem is built containing the m most influential variables, $n-m$ fixed variables and all of the m given equations. ("Most influential" or "most sensitive" here refers to the influence of changes of the variable j on the function value of function component i. An indicator for this influence is the $(i,j)th$ entry of the Jacobian matrix.)

At this point, another definition is introduced to shorten the following notations.

Definition 4.5
For a list of n arguments and an index set $K \subseteq \{1,\ldots,n\}$, the sublist x_K of x is composed by the components x_k with $k \in K$. The complement of this list, containing the x_k with $k \notin K$, is denoted as $x_{\notin K}$. The same notation applies for interval values.

Let the index set K contain the indices of the m components in a solution box $\boldsymbol{x}$ most influential on the function enclosure of function f and let x and y be real-valued elements in $\boldsymbol{x}$. The function for the desired square subsystem can now be defined as

$$f_{square}(x_K) = f(x_K, y_{\notin K}) = f(v) \text{ with } \quad v_k = \begin{cases} x_k & \text{if } k \in K \\ y_k & \text{if } k \notin K \end{cases}.$$

For an interval valued function, let $\boldsymbol{y}$ for now be an arbitrary but fixed subbox of $\boldsymbol{x}$. (Methods for choosing $\boldsymbol{y}$ will be discussed soon.)

$$\boldsymbol{f}_{square}(\boldsymbol{x}_K) := \boldsymbol{f}(\boldsymbol{x}_K, \boldsymbol{y}_{\notin K}) = \boldsymbol{f}(\boldsymbol{v}), \quad \boldsymbol{v}_k = \begin{cases} \boldsymbol{x}_k & \text{if } k \in K \\ \boldsymbol{y}_k & \text{if } k \notin K. \end{cases} \tag{4.1}$$

Verification tests for the subsystem are based on the evaluation of the function enclosures $\boldsymbol{f}_{square}(\boldsymbol{x}_K)$ and thus of $\boldsymbol{f}(\boldsymbol{x}_K, \boldsymbol{y}_{\notin K})$. Hence, if a solution $x_K \in \boldsymbol{x}_K$ of $f_{square} = 0$ is verified, it is not only shown that

$$\exists\, x_K \in \boldsymbol{x}_K \text{ with } f_{square}(x_K) = 0 \tag{4.2}$$

but even that

$$\exists\, x_K \in \boldsymbol{x}_K \text{ with } f(x_K, y_{\notin K}) = 0 \quad \text{ for all } y_{\notin K} \in \boldsymbol{y}_{\notin K}$$

holds.

Which variables are to be fixed The next question arising is how to determine the variable indices to be comprised in the index set K. For that purpose

we utilize a QR decomposition with column pivoting based on an algorithm given in [GVL89]. This algorithm is applied to the (real-valued) matrix containing the midpoint matrix of the Jacobian matrix. It determines the desired set K containing the indices of the m most influential variables (by a permutation of the matrix). Notice that set K has to be determined separately for every solution box because the variables may have different influence on the function enclosures for different boxes.

Fixation to interval values We now want to discuss how to choose the subbox $\boldsymbol{y}$ of $\boldsymbol{x}$. When considering formula (4.1) we can argue to "fix" variables to subsets in $\boldsymbol{y}_{\notin K}$.

The first option to fix a variable is to set it to its interval value in the solution box by defining

$$\boldsymbol{y} := \boldsymbol{x}.$$

Hence the verification of (4.2) includes that a solution is verified for all possible values of $x_{\notin K} \in \boldsymbol{x}_{\notin K}$ and

$$\exists\, x_K \quad \text{with} \quad f_{square}(x_K) = f(x_K, x_{\notin K}) = 0 \quad \text{for all } x_{\notin K} \in \boldsymbol{x}_{\notin K}.$$

Thus the existence of a manifold of solutions for the underdetermined system is verified whenever width $(\boldsymbol{x}_{\notin K}) > 0$ and a verification test proves the existence of a solution for the corresponding square system.

Fixation to point values Another option for fixing variables is the fixation to a single number $\widetilde{x}$ (or better: the point interval $[\widetilde{x}]$) from the given interval leading to

$$\boldsymbol{y} := \widetilde{x} \in \boldsymbol{x}.$$

In this case the verification of (4.2) again implies

$$\exists\, x_K \quad f_{square}(x_K) = f(x_K, y_{\notin K}) = 0 \quad \text{for all } y_{\notin K} \in \boldsymbol{y}_{\notin K}$$

and results in

$$\exists\, x_K \quad f(x_K, \widetilde{x}_{\notin K}) = 0$$

for the special choice of $\boldsymbol{y}$. Fixing the variables x_i with $i \notin K$ to point instead of interval values has the advantage to reduce overestimation in the verification tests. It might therefore be possible to verify more often with this choice.

Mind that none of these options allows to verify unique solutions in the solution boxes. Even if uniqueness is verified for the box $\boldsymbol{x}_K$ in the square subsystem, only the existence of a solution can be claimed in the corresponding box $\boldsymbol{x}$ for the whole system. The reason for this "loss of information" is that for the fixed variables only one single value from each $\boldsymbol{y}_{\notin K}$ is considered in the verification test. Note

further that a verification test may really verify a manifold of solutions if the point values cannot be enclosed by point intervals but have to be rounded outward to proper intervals $\boldsymbol{y} = \text{flint}(\widetilde{x})$. In implementations one can evade this problem by simply choosing $\widetilde{x}$ as a machine number in $\boldsymbol{x}$.

Try fixation to intervals The author proposes a third option to fix variables for underdetermined systems. It combines the two simpler fixation options. Our strategy basically tries first to verify setting all variables to interval values. If this is not possible, we verify the existence or uniqueness of a solution with fixation of the variables to single numbers. Afterwards, we test whether it is still possible to verify if some of the variables are reset to their intervals values.

For a formal representation, the variables fixed to intervals have to be distinguished from those fixed to point values. In the following lines, the index set I corresponds to the variables fixed to interval values and P is the index set relating to variables fixed to point values. For the index sets the following relations have to hold

$$I, P \subseteq \{1, \ldots, n\},\ I \cap P = I \cap K = K \cap P = \emptyset \text{ and } K \cup I \cup P = \{1, \ldots, n\}.$$

We can now write the square function as

$$f_{square}(x_K) = f(x_K, y_I, z_P) = f(v) \text{ with } \quad v_k := \begin{cases} x_k & \text{if } k \in K \\ y_k & \text{if } k \in I \\ z_k & \text{if } k \in P \end{cases}$$

with $x, y, z \in \boldsymbol{x}$. For an interval notation, $\boldsymbol{y}_I$ is chosen to be equal to $\boldsymbol{x}$ and $\boldsymbol{z}_P$ is set to the midpoint vector of $\boldsymbol{x}_k$ yielding

$$\boldsymbol{f}_{square}(\boldsymbol{x}_K) := \boldsymbol{f}(\boldsymbol{x}_K, \boldsymbol{y}_I, \boldsymbol{z}_P) = f(\boldsymbol{v}) \text{ with } \quad \boldsymbol{v}_k := \begin{cases} \boldsymbol{x}_k & \text{if } k \in K \\ \boldsymbol{x}_k & \text{if } k \in I \\ \text{mid}\,(\boldsymbol{x}_k) & \text{if } k \in P \end{cases}.$$

In this case $0 \in \boldsymbol{f}_{square}(\boldsymbol{x}_K)$ implies

$$\exists\, x_K \in \boldsymbol{x}_K \text{ with } f(x_K, y_I, z_P) = 0 \quad \text{ for all } y_I \in \boldsymbol{y}_I.$$

The exact approach is given in Algorithm 4.1. It proceeds stepwise. First all variables are fixed to interval values. If no solution can be verified with this setting, all intervals are fixed to their midpoints and the verification is repeated. If the second verification succeeds, the variables are reset to their interval values step by step. To maximize the chances to remain able to verify, in each step the variable with the least influence on the function values is reset to its interval value. This is done as long as verification is still possible.

Algorithm 4.1 TRY TO FIX TO INTERVALS (box $\boldsymbol{x}$)

```
1: fix all variables to their interval values
2: run verification tests for square subsystem
3: if neither uniqueness nor existence could be proved then
4:    fix all variables to their midpoints
5:    run verification tests for square subsystem
6:    if solution was verified then
7:       for i = 1, ..., n − m do
8:          set the least influential variable back to its interval value
9:          run verification tests for square subsystem
10:      end for
11:      set x to last box that was verified
12:   end if
13: end if
```

The author implemented all three options to fix variables into SONIC. In case of fixing to numbers, all variables to be fixed are set to the midpoints of their interval values.

4.2.2 Overdetermined systems

For overdetermined systems we have to consider a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with more constraints than variables ($m > n$). For those systems we can only check whether a solution can be verified for an appropriate square subsystems and whether this solution fulfills the constraints not in the subsystem. A verification of a solution within a box as for square systems is not possible for overdetermined systems.

As mentioned by Beelitz [Bee06]), the rank-revealing QR factorization can be applied to select n constraints to build a square system. All verification methods can then be applied to the evolving subsystem.

SONIC handles overdetermined systems as indicated. The purpose of this approach is to give the user a hint for the existence or uniqueness of a solution. To distinguish the attained information from verified solutions, two additional states *ContainsSolutionForSubsystem* and *UniqueSolutionForSubsystem* have been added in the implementation.

4.3 Verification for square systems

We now want to discuss some tests to verify solutions for square systems of nonlinear equations. It is worth noting in advance that the Newton test utilizes an interval Newton step and can be applied to prove the uniqueness of a solution, while the Miranda, Borsuk and degree test work based on interval evaluations on the boundary of the solution boxes and prove the existence of a solution in a given box.

Thus tests can be improved by sharper function enclosures. Two common approaches to do so are preconditioning and, for the latter three tests, subdivision of the boundary of the box to compute function enclosures on smaller domains. If overestimation is too large, however, the tests fail although they would be able to verify solutions given the exact range instead of function enclosures.

4.3.1 Newton test

The Newton test is based on the interval Newton method as discussed in section 2.3.4. To verify the uniqueness of a solution in a given box, we simply apply one step of the interval Newton operator. Herein derivatives (or slopes including these derivatives) have to be employed to be able to prove uniqueness (cf. Theorem 2.1). If the inflated box $\boldsymbol{x}$ can be contracted to a smaller box $\boldsymbol{y}$ by the interval Newton operator, one has proved the uniqueness of a solution in $\boldsymbol{x}$ and even in $\boldsymbol{y}$. Note further that the Newton test may also be applied implicitly in the interval Newton method while computing the solution boxes in the branch-and-bound algorithm.

Since the interval Newton operator is a contracting operator, in some cases, solutions cannot be verified. In this case it can thus be beneficial to apply the interval Newton operator to a solution box $\boldsymbol{x}_\epsilon$ that has been widened by an epsilon-inflation (cf. section 4.1.2). Even if starting with this inflated box $\boldsymbol{x}_\epsilon$, one might verify a solution in a box smaller than the original solution box $\boldsymbol{x}$.

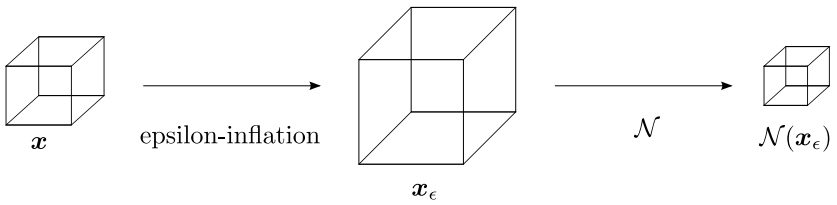


Figure 4.2: Anticipated interaction of epsilon-inflation and the interval Newton operator $\mathcal{N}$

4.3.2 Miranda test

In [Mir40], Miranda gives a theorem generalizing the intermediate value theorem to higher dimensions. It leads to the claim that a box contains a zero for a continuous function if the function enclosures over the pairs of opposite facets have different signs.

Theorem 4.1 (Miranda)
Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function and $\boldsymbol{x} \in \mathbb{IR}^n$ a box. If

$$f_i(x^{i,-}) \leq 0 \leq f_i(x^{i,+}) \tag{4.3}$$

$$\text{or} \quad f_i(x^{i,-}) \geq 0 \geq f_i(x^{i,+}) \tag{4.4}$$

holds for all $x^{i,+} \in \boldsymbol{x}^{i,+}$, all $x^{i,-} \in \boldsymbol{x}^{i,-}$ and for all $i \in \{1, \ldots, n\}$, then f has a zero in $\boldsymbol{x}$.

In practice, conditions (4.3) and (4.4) are checked for a preconditioned function $g(x) = C \cdot f(x)$. Note that condition (4.3) is equivalent to condition (4.4) when replacing f_i by $-f_i$. We continue considering solely the first condition (4.3) for all f_i, since any system fulfilling a combination of (4.3) and (4.4) for its components f_i can be verified using only (4.3) when applying an appropriate diagonal matrix with entries in $\{-1, 1\}$ as preconditioner. (In an implementation, however, all possible combinations of the conditions have to be checked.) Thus we can proceed with the condition

$$g_i(x^{i,-}) \leq 0 \leq g_i(x^{i,+})$$

for the preconditioned function g for all $x^{i,+} \in \boldsymbol{x}^{i,+}, x^{i,-} \in \boldsymbol{x}^{i,-}$ and all $i \in \{1, \ldots, n\}$. Using interval analysis, this condition can be checked by confirming the inequality

$$\sup(\boldsymbol{g}_i(\boldsymbol{x}^{i,-})) \leq 0 \leq \inf(\boldsymbol{g}_i(\boldsymbol{x}^{i,+})) \tag{4.5}$$

for all $i \in \{1, \ldots, n\}$. If this condition is fulfilled, the existence of a zero in box $\boldsymbol{x}$ is revealed.

As for all verification tests, overestimation could lead the test of condition (4.5) to fail although all prerequisites of the above theorem are fulfilled. To reduce overestimation, we use subfacets instead of facets. The subdivision can be conducted independently for all facets (for details see section 4.5.5) and we get a condition for the subfacets. Function f has a zero in box $\boldsymbol{x}$ if

$$\begin{cases} \sup(\boldsymbol{g}_i(\boldsymbol{x}^{i,-,l})) & \leq 0 \\ \inf(\boldsymbol{g}_i(\boldsymbol{x}^{i,+,k})) & \geq 0 \end{cases} \quad \text{and}$$

can be shown for all $i \in \{1, \ldots, n\}$ and all subfacets (i.e. for all $k \in \{1, \ldots, K_i\}$ and all $l \in \{1, \ldots, L_i\}$). The evolving test is called the *Miranda test* and given in Algorithm 4.2. For shorter notations the representation info($\boldsymbol{x}$) is introduced for the available information about solutions in a given box $\boldsymbol{x}$.

Algorithm 4.2 MIRANDA TEST (bounded box $\boldsymbol{x}$)

1: **for** $i = 1, \ldots, n$ **do**
2: **for** facet $\boldsymbol{x}_{\text{facet}} \in \{\boldsymbol{x}^{i,+}, \boldsymbol{x}^{i,-}\}$ **do**
3: initialize list $L^{(1)}_{\text{facet}}$ with $\boldsymbol{x}_{\text{facet}}$
4: $n_{\text{sub}} \leftarrow 1$
5: **for** $u = 1, \ldots,$ MaxDepth **do**
6: **while** $L^{(u)}_{\text{facet}}$ is not empty **do**
7: take first element $\boldsymbol{y}$ in $L^{(u)}_{\text{facet}}$
8: **if** $\boldsymbol{y}$ together with previously handled facets contradicts (4.3) or (4.4) as condition for Miranda test **then**
9: exit {verification not possible}
10: **end if**
11: **if** $\boldsymbol{y}$ fulfills condition for Miranda test **then**
12: delete $\boldsymbol{y}$ from list $L^{(u)}_{\text{facet}}$
13: **else** {no decision possible, $\boldsymbol{y}$ has to be subdivided}
14: **if** $u <$ MaxDepth and $n_{\text{sub}} + 2 \leq$ MaxSub **then**
15: subdivide $\boldsymbol{y}$ into two subfacets
16: $n_{\text{sub}} \leftarrow n_{\text{sub}} + 2$
17: insert subfacets into list $L^{(u+1)}_{\text{facet}}$
18: **else**
19: exit {no further subdivision allowed, no solution verified}
20: **end if**
21: **end if**
22: **end while**
23: **end for**
24: **end for**
25: **end for**
26: info($\boldsymbol{x}$) $\leftarrow$ *ContainsSolution*

4.3.3 Borsuk test

The next test is based on a theorem originally given by Borsuk in [Bor33]. From a geometrical point of view, the theorem of Borsuk guarantees the existence of a zero if a function points in different directions for each pair of opposite points on the boundary of the box.

Theorem 4.2 (Borsuk [FL05])
Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function and $\boldsymbol{x}$ a box in $\mathbb{IR}^n$. Function f has a zero in box $\boldsymbol{x}$ if

$$f(\mathrm{mid}\,(\boldsymbol{x}) + y) \neq \lambda \cdot f(\mathrm{mid}\,(\boldsymbol{x}) - y) \tag{4.6}$$

holds true for all positive real numbers $\lambda \in \mathbb{R}_+$ and for all y, for which $(\mathrm{mid}\,(\boldsymbol{x})+y)$ is an element of the facet $\boldsymbol{x}^{i,+}$ (and $(\mathrm{mid}\,(\boldsymbol{x}) - y)$ is element of $\boldsymbol{x}^{i,-}$).

As before, we have the option to reduce overestimation by subdividing facets into subfacets. For the Borsuk test, symmetrical subfacets $\boldsymbol{x}^{i,+,l}$ and $\boldsymbol{x}^{i,-,l}$ ($l \in \{1, \ldots, L_i\}$, $L_i \in \mathbb{N}$) have to be used for opposite pairs of facets.

Further we can again conduct preconditioning with a nonsingular preconditioning matrix $C \in \mathbb{R}^{n \times n}$ and apply the verification test to the preconditioned function $g(x) = C \cdot f(x)$. Like in the Miranda test the inverse midpoint preconditioner can be used for preconditioning.

As stated in [FL04], one has even more freedom of choice concerning the preconditioner for the Borsuk test. By contrast to the Miranda test, where the same preconditioner has to be applied for all facets, for the Borsuk test one has the opportunity to employ different preconditioners for the pairs of facets. Even more, one can choose a different preconditioner for each offset y in (4.6).

Furthermore, a new preconditioner for the Borsuk test, which can be computed separately for each pair of symmetric subfacets, is presented in the same paper [FL04]. The ansatz for this preconditioner is to choose the preconditioning matrix $C^{i,l}$ in such a way that $g = C^{i,l} \cdot f$ points in almost opposite directions on opposite subfacets $\boldsymbol{x}^{i,+,l}$ and $\boldsymbol{x}^{i,-,l}$. To attain this property, the function values at the midpoints of the subfacets are considered. The matrix $C^{i,l}$ is chosen such that the preconditioned function for one facet is

$$C^{i,l} \cdot f(\mathrm{mid}\,(\boldsymbol{x}^{i,+,l})) = (100, 1, 0, \ldots, 0)^T$$

and for the other

$$C^{i,l} \cdot f(\mathrm{mid}\,(\boldsymbol{x}^{i,-,l})) = (-100, 1, 0, \ldots, 0)^T.$$

A numerical example for the application of the Borsuk test with an evaluation of its success can be found in [FL05]. For an example system the authors gave a study of the success of preconditioners for the Borsuk test with respect to the number of subfacets. Further comparisons of the verification tests discussed in this thesis can be found in section 4.6.

Besides, for a practical verification test, another range based-condition is used that can be checked by interval analysis. Condition (4.6) holds for the preconditioned function g if

$$\bigcap_{j=1}^{n} \frac{\boldsymbol{g}_j(\boldsymbol{x}^{i,+})}{\boldsymbol{g}_j(\boldsymbol{x}^{i,-})} \cap (0, \infty) = \emptyset$$

holds for each $i \in \{1, \ldots, n\}$. (At this point the denotation of an open interval is allowed as an exception for the sake of consistency with the original paper [FL05].

By using subfacets as well as the preconditioned function g, we attain the formulation

$$\underbrace{\bigcap_{j=1}^{n} \frac{\boldsymbol{g}_j(\boldsymbol{x}^{i,+,l})}{\boldsymbol{g}_j(\boldsymbol{x}^{i,-,l})}}_{=:G} \cap (0, \infty) = \emptyset \tag{4.7}$$

for all $i \in \{1, \ldots, n\}$ and all pairs of symmetric subfacets (meaning for all $l \in \{1, \ldots, L\}$) for proving a zero of f in the box $\boldsymbol{x}$. We refer to an implementation of this formulation as the *Borsuk test*, see Algorithm 4.3. Note that in all formulae divisions are to be understood as relational divisions (cf. Definition 1.6) whenever an interval in the denominator contains zero. In an implementation condition, the intersection G can further be checked to be a subset of $\mathbb{R}_+$.

Algorithm 4.3 BORSUK TEST (bounded box $\boldsymbol{x}$)

for $i = 1, \ldots, n$ **do**
  initialize list $L_{\text{facet}}^{(1)}$ with $\boldsymbol{x}^{i,+}$
  $n_{\text{sub}} \leftarrow 1$
  set $\boldsymbol{z} \leftarrow (0, \infty)$
  for $u = 1, \ldots,$ MaxDepth **do**
    while $L_{\text{facet}}^{(u)}$ is not empty **do**
      take first element $\boldsymbol{y}$ in $L_{\text{facet}}^{(u)}$
      compute symmetric facet or subfacet $\boldsymbol{y}_{\text{symm}}$ of $\boldsymbol{y}$
      test condition (4.7) for Borsuk test with pair of symmetric subfacets
      $\boldsymbol{z} \leftarrow \boldsymbol{z} \cap \bigcap_{j=1}^{n} \dfrac{\boldsymbol{g}_j(\boldsymbol{y})}{\boldsymbol{g}_j(\boldsymbol{y}_{\text{symm}})}$
      if condition (4.7) not proved by now ($\boldsymbol{z} \neq \emptyset$) **then**
        delete $\boldsymbol{y}$ from list $L_{\text{facet}}^{(u)}$
      end if
      if condition (4.7) is proved ($\boldsymbol{z} = \emptyset$) **then**
        info($\boldsymbol{x}$) $\leftarrow$ *ContainsSolution*
        exit {algorithm is terminated}
      else {no decision possible, $\boldsymbol{y}$ has to be subdivided}
        if $u <$ MaxDepth and $n_{\text{sub}} + 2 \leq$ MaxSub **then**
          subdivide $\boldsymbol{y}$ into two subfacets
          $n_{\text{sub}} \leftarrow n_{\text{sub}} + 2$
          insert subfacets into list $L_{\text{facet}}^{(u+1)}$
        else
          exit {no further subdivision allowed, no solution verified}
        end if
      end if
    end while
  end for
end for

4.3.4 A test based on the topological degree

The next test is called the "degree test" within this work. The framework of the degree test is presented following [FHL07] and [BFLW09]. Given

- an open and bounded set $D \subseteq \mathbb{R}^n$ with closure $\mathrm{cl}(D)$ and
- a continuous function $f : \mathrm{cl}(D) \to \mathbb{R}^n$ with $f(x) \neq y$ for all $x \in \partial D$

the *topological degree* $d(f, D, y)$ is an integer providing information about the number of solutions $x \in D$ of $f(x) = y$. For the intended verification test we do not have to determine this number, but will make use of the invariance of the topological degree under homotopies as claimed by the following theorem.

Theorem 4.3 (Properties of the topological degree [FHL07, BFLW09])
If the prerequisites given above are fulfilled, the following properties of the topological degree can be proved.

(i) If a homotopy $h : \mathrm{cl}(D) \times [0,1] \to \mathbb{R}^n$ is continuous and fulfills

$$h(x,t) \neq y \text{ for all } x \in \partial D \text{ and all } t \in [0,1],$$

then $d(h(x,t), D, y)$ does not depend on t.

(ii) If f is (Fréchet-) differentiable and the Jacobian matrix $f'(x)$ is nonsingular at all points $x \in D$ with $f(x) = y$, then

$$d(f, D, y) = \sum_{\substack{x \in D \\ f(x)=y}} \operatorname{sign} \det f'(x).$$

(iii) If $d(f, D, y) \neq 0$, then there exists an element $x \in D$ with $f(x) = y$.

The proof for Theorem 4.3 and further information regarding the topological degree can, e.g., be obtained from [Dei85].

Since we are interested in verifying a zero of f in a box $\boldsymbol{x}$, we consider part (iii) of Theorem 4.3, set y to zero and D to the interior of the box $\boldsymbol{x}$.

At this point it has to be emphasized that the topological degree is not defined if $D = \boldsymbol{x}$ and the box contains a zero of f on its boundary—which is not unlikely for small solution boxes. In this case the following test is *not* applicable.

As before, we consider the preconditioned function $g(x)$ instead of f. Thus we can prove the existence of a zero of f by considering the homotopy

$$h : \boldsymbol{x} \times [0,1] \to \mathbb{R}^n \quad \text{defined by}$$
$$h(x,t) = t \cdot g(x) + (1-t) \cdot (x - \widetilde{x}).$$

Within that formula, x and $\widetilde{x}$ are inner points of $\boldsymbol{x}$ and $\widetilde{x}$ is fixed. Provided h has no zero in $\partial\boldsymbol{x} \times [0,1]$, due to part (i) of Theorem 4.3 we know that $d(h, \operatorname{int}(\boldsymbol{x}), 0)$ does not depend on t. Thus, $g(x)$ has the same topological degree as function $(x - \widetilde{x})$. Considering $\widetilde{x} \in \operatorname{int}(\boldsymbol{x})$ and $x \in \partial\boldsymbol{x}$, part (ii) of Theorem 4.3 tells us that

$$d(x - \widetilde{x}, \operatorname{int}(\boldsymbol{x}), 0) = 1.$$

By combining these two results we get

$$d(x - \widetilde{x}, \operatorname{int}(\boldsymbol{x}), 0) = d(h, \operatorname{int}(\boldsymbol{x}), 0) = d(g, \operatorname{int}(\boldsymbol{x}), 0) = 1.$$

Now part (iii) of Theorem 4.3 tells us that there exists a zero for g in $\boldsymbol{x}$. To sum up, it is sufficient to show

$$0 \neq h(x,t) = t \cdot g(x) + (1-t) \cdot (x - \widetilde{x}) \quad \text{for all } x \in \partial\boldsymbol{x},\ t \in [0,1] \tag{4.8}$$

to prove the existence of a zero of g (and therefore of f) in $\boldsymbol{x}$.

The purpose of the preconditioner is to get the preconditioned function g as similar to $(x - \widetilde{x})$ as possible. Because $\widetilde{x}$ is chosen from the interior of $\boldsymbol{x}$, this choice of the preconditioner reduces the odds of h having a zero in $\partial\boldsymbol{x} \times [0,1]$. To prove that h has indeed no zero in $\partial\boldsymbol{x} \times [0,1]$, interval analysis is applied. The next theorem evolves from a reformulation of condition (4.8) using facets to cover $\partial\boldsymbol{x}$.

Theorem 4.4
Let $\boldsymbol{t}$ be the interval $[0,1]$ and $i \in \{1, \ldots, n\}$. The function f has a zero in $\boldsymbol{x}$ if none of the $2n$ systems

$$h(x,t) = 0 \quad \textit{with } (x,t) \in \boldsymbol{x}^{i,+} \times \boldsymbol{t} \textit{ or } (x,t) \in \boldsymbol{x}^{i,-} \times \boldsymbol{t}$$

has a solution. The facets $\boldsymbol{x}^{i,+} \times \boldsymbol{t}$ and $\boldsymbol{x}^{i,-} \times \boldsymbol{t}$ herein have to fulfill

$$\bigcup_{i=1}^{n} \left(\boldsymbol{x}^{i,+} \times \boldsymbol{t}\right) \cup \bigcup_{i=1}^{n} \left(\boldsymbol{x}^{i,-} \times \boldsymbol{t}\right) = \partial\boldsymbol{x} \times \boldsymbol{t}.$$

A verification test based on the last theorem checks that $h(x,t)$ has no zero for $x \in \partial\boldsymbol{x}$ and $t \in [0,1]$. For doing so we have to consider h on $2n$ boxes in $\mathbb{R}^{n+1}$. At first glance it may seem unreasonable to consider $2n$ boxes instead of just verifying a zero of f in a single box. But in many cases it emerges to be much easier and cheaper in computational costs to exclude the existence of a zero in a given box than proving it.

To avoid overestimation, the facets are again subdivided into subfacets. For a computational test using interval analysis, one further has to subdivide the interval $\boldsymbol{t}$ and to consider h on all emerging parts of $\partial\boldsymbol{x} \times [0,1]$. The subdivision of $\boldsymbol{t}$ is necessary because $[0,1] \cdot \boldsymbol{g}(x) + [0,1] \cdot (x - \widetilde{x})$ contains zero, independently of the choice of function g or the subdivision of $\boldsymbol{x}$. Hence every calculation with $\boldsymbol{t} = [0,1]$ has to fail in proving the non-existence of zero. In the following theorem, both subdivisions are applied.

Theorem 4.5 ([FHL07])
Let $\boldsymbol{x}$ *be an interval vector in* $\mathbb{R}^n$ *and* $\widetilde{x} \in \operatorname{int}(\boldsymbol{x})$. *Let*

$$\partial \boldsymbol{x} = \bigcup_{k=1}^{K} \boldsymbol{y}^{(k)} \quad \text{and} \quad [0,1] = \bigcup_{l=1}^{L} \boldsymbol{t}^{(l)}.$$

be subdivisions of $\partial \boldsymbol{x}$ *and* $[0,1]$. *If for all* $k \in \{1,\dots,K\}$ *and* $l \in \{1,\dots,L\}$ *there exists an index* $j = j(k,l) \in \{1,\dots,n\}$ *such that*

$$0 \notin \boldsymbol{t}^{(l)} \cdot \boldsymbol{g}_j(\boldsymbol{y}^{(k)}) + (1 - \boldsymbol{t}^{(l)}) \cdot (\boldsymbol{y}^{(k)} - \widetilde{x})_j = \boldsymbol{h}_j(\boldsymbol{y}^{(k)}, t^{(l)}) \tag{4.9}$$

then the function g *(and thus* f*) has a zero in* $\operatorname{int}(\boldsymbol{x})$.

The proof is simply done by showing that zero is not included in the union of all $\boldsymbol{h}(\boldsymbol{y}^{(k)}, \boldsymbol{t}^{(l)})$. Since this union includes $\operatorname{range}_h(\boldsymbol{x} \times \boldsymbol{t})$, the range also cannot contain zero. Condition (4.9) can be checked using interval analysis to get a computational test. We refer to this test as the *degree test*. It is represented in Algorithm 4.4. For the degree test the same preconditioner has to be used for all facets and subfacets.

Depending on how to split $\boldsymbol{t}$ and the subfacets, how to compute the function enclosures, how to check for zeros of h, and the employed homotopy this leads to different computational existence tests. Some variations and realizations of the degree test are examined in section 4.6.2.

Algorithm 4.4 DEGREE TEST (bounded box $\boldsymbol{x}$)

$t \leftarrow [0,1]$
for $i = 1, \ldots, n$ **do**
  for facet $\boldsymbol{x}_{\text{facet}} \in \{\boldsymbol{x}^{i,+}, \boldsymbol{x}^{i,-}\}$ **do**
    initialize list $L^{(1)}_{\text{facet}}$ with $\boldsymbol{x}_{\text{facet}}$
    $n_{\text{sub}} \leftarrow 1$
    for $u = 1, \ldots,$ MaxDepth **do**
      while $L^{(u)}_{\text{facet}}$ is not empty **do**
        take first element $\boldsymbol{y}$ in $L^{(u)}_{\text{facet}}$
        for all parts $\boldsymbol{t}^{(l)}$ of $\boldsymbol{t}$ **do**
          if $(\boldsymbol{y}, \boldsymbol{t}^{(l)})$ fulfills condition for degree test **then**
            delete $\boldsymbol{y}$ from list $L^{(u)}_{\text{facet}}$
          else {facet has to be divided}
            if $u <$ MaxDepth and $n_{\text{sub}} + 2 \leq$ MaxSub **then**
              subdivide $\boldsymbol{y}$ into two subfacets
              $n_{\text{sub}} \leftarrow n_{\text{sub}} + 2$
              insert subfacets into list $L^{(u+1)}_{\text{facet}}$
            else
              exit {no further subdivision allowed, no solution verified}
            end if
          end if
        end for
      end while
    end for
  end for
end for
info($\boldsymbol{x}$) $\leftarrow$ *ContainsSolution*

4.3.5 Intersecting boxes

Solution boxes can always have intersecting boundaries. Due to epsilon-inflation the intersections of two solution boxes can be even larger. In this section we want to present approaches that can reduce the number of solution boxes without losing information or solutions.

In some cases the reduction of the number of solution boxes is possible if one solution box is contained in another ($\boldsymbol{x}_{\text{sup}} \supseteq \boldsymbol{x}_{\text{sub}}$). Which one of these boxes can be discarded relies on the information about existence and uniqueness of solutions in the boxes. We proceed according to the (pre-existing) Algorithm 4.5 and follow the simple logic to reduce the box number and to save smaller boxes under the condition that no information is lost. Note especially that the subbox $\boldsymbol{x}_{\text{sub}}$ is not discarded if it is known to contain a unique solution and the number of solutions in the superbox $\boldsymbol{x}_{\text{sup}}$ is unknown. In this case, box $\boldsymbol{x}_{\text{sub}}$ could be discarded without loss of solutions, but information about the location of a unique solution would be lost. The handling for the values *ContainsSolutionForSubsystem* and *UniqueSolutionForSubsystem* was added. It is done as for the corresponding values for square systems if the same subsystem is chosen for two boxes and is not noted explicitly.

An example for which Algorithm 4.5 can reduce the number of boxes is problem *G7_gradientsystem*. For this problem our solver ends up with a list containing 320 solution boxes. However, the box $[0, 0]^7$ is contained $64 = 2^6$ times within this set. Other boxes are contained $32 = 2^5$ or $16 = 2^4$ times, so we observe clusters of different sizes. Here we clearly see a result of the cluster effect, where subdivision yielded multiple subboxes containing the same solution on their boundaries. Since these boxes were contracted to point intervals in all components by the subsequent application of contractors, they result in the same solution boxes (equal to vectors of point intervals corresponding to the real-valued solutions). The collect algorithm reduced the number of solution boxes to 13.

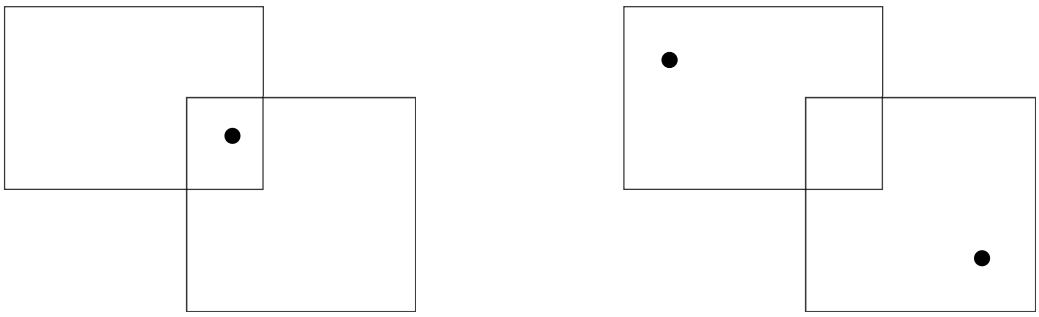
We introduce a further check for intersecting boxes. If for two intersecting boxes the uniqueness of a solution has been shown by verification, there exist two possible scenarios. There could be two solutions in the non-intersecting parts of the boxes or only one solution in the intersection (as depicted in Figure 4.3). Thus each search for intersecting unique boxes has to be followed by a verification of the intersection. Herein no epsilon-inflation is allowed. If the verification proves the uniqueness or existence of a solution in the intersection, the solutions in the boxes have to coincide. Hence the two intersecting boxes can be replaced in the solution list by just one, smaller box known to contain a unique solution. The algorithm is summarized in Algorithm 4.6.

Algorithm 4.5 COLLECT BOXES (list L_s of solution boxes)

for all pairs of solution boxes $\boldsymbol{x}_{\text{sup}}$ and $\boldsymbol{x}_{\text{sub}}$ with $\boldsymbol{x}_{\text{sup}} \supseteq \boldsymbol{x}_{\text{sub}}$ **do**
  if info($\boldsymbol{x}_{\text{sup}}$) = *NoInformation* **then**
    if info($\boldsymbol{x}_{\text{sub}}$) = *NoInformation* **then**
      delete $\boldsymbol{x}_{\text{sub}}$
    else
      info($\boldsymbol{x}_{\text{sup}}$) $\leftarrow$ *ContainsSolution*
    end if
  end if
  if info($\boldsymbol{x}_{\text{sup}}$) = *ContainsSolution* **then**
    if info($\boldsymbol{x}_{\text{sub}}$) = *NoInformation* **then**
      delete $\boldsymbol{x}_{\text{sub}}$
    end if
  end if
  if info($\boldsymbol{x}_{\text{sup}}$) = *UniqueSolution* **then**
    if info($\boldsymbol{x}_{\text{sub}}$) = *NoInformation* **then**
      delete $\boldsymbol{x}_{\text{sub}}$
    else
      if info($\boldsymbol{x}_{\text{sub}}$) = *ContainsSolution* **then**
        info($\boldsymbol{x}_{\text{sub}}$) $\leftarrow$ *UniqueSolution*
        delete $\boldsymbol{x}_{\text{sup}}$
      else
        if info($\boldsymbol{x}_{\text{sub}}$) = *UniqueSolution* **then**
          delete $\boldsymbol{x}_{\text{sup}}$
        end if
      end if
    end if
  end if
end for

Algorithm 4.6 COLLECT UNIQUE BOXES (list L_s of solution boxes)

for each ordered pair of intersecting solution boxes $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ containing unique solutions **do**
  if $\boldsymbol{x}^{(1)} \subseteq \boldsymbol{x}^{(2)}$ **then**
    delete $\boldsymbol{x}^{(2)}$
  else
    $\boldsymbol{x}^{\cap} \leftarrow \boldsymbol{x}^{(1)} \cap \boldsymbol{x}^{(2)}$
    if existence or uniqueness can be verified in $\boldsymbol{x}^{\cap}$ **then**
      add $\boldsymbol{x}^{\cap}$ to L_s
      delete $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$ from L_s
    end if
  end if
end for



(a) Boxes cover the same solution

(b) Boxes cover different solutions

Figure 4.3: Possible location of solutions in intersecting boxes containing unique solutions

To provide some further helpful information, the user of SONIC is informed of the number of pairs of intersecting unique boxes.

4.4 Verification scheme

For verifying a single box, the author proposes the scheme represented by Algorithm 4.7. The sequence of the verification tests is influenced by considerations about the strength of the verification tests as will be given in section 4.6. For clarification, the information about solutions that can be obtained in this scheme is illustrated by Figure 4.4.

For shorter notation only the procedure for square systems is given. Over- and underdetermined systems are handled, as described in the previous sections. We discuss the values *UniqueSolution*, *ContainsSolution* and *NoInformation* with obvious meanings. The values *ContainsSolutionForSubsystem* and *UniqueSolutionForSubsystem* were introduced by the author for storing information about the verification success for subsystems of non-square systems. For these systems the information about existence and uniqueness of a solution are adjusted after running the verification tests (to *ContainsSolutionForSubsystem* or *UniqueSolutionForSubsystem* according to the claims in section 4.2).

Algorithm 4.7 VERIFICATION FOR SQUARE SYSTEMS (box $\boldsymbol{x}$, ϵ, maximum number of inflations)

```
1: copy box (y ← x)
2: if uniqueness not verified then
3:    while existence not verified and not reached maximum number of inflations
      do
4:       apply Newton test
5:       if uniqueness not verified then
6:          apply epsilon-inflation with given ε (x ← x_ε)
7:       end if
8:    end while
9: end if
10: if neither uniqueness nor existence verified then
11:    set box back to value before inflation (x ← y)
12:    while existence not verified and not reached maximum number of inflations
       do
13:       apply Miranda test (Alg. 4.2)
14:       if existence not verified then
15:          apply Borsuk test (Alg. 4.3)
16:       end if
17:       if existence not verified then
18:          apply degree test (Alg. 4.4 or 4.8)
19:       end if
20:       if existence not verified then
21:          apply epsilon-inflation with given ε (x ← x_ε)
22:       end if
23:    end while
24: end if
25: if existence not verified then
26:    set box back to value before inflation (x ← y)
27: end if
```

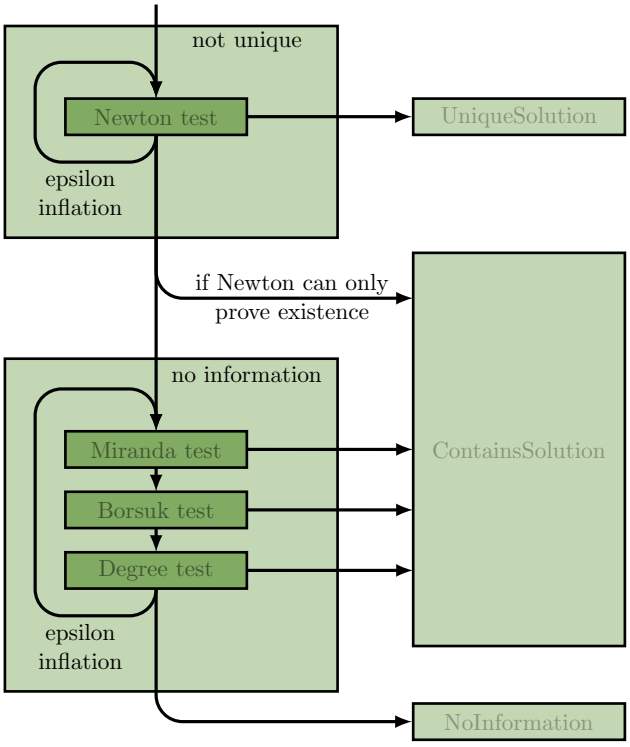


Figure 4.4: Verification scheme for square systems with attainable information about solutions in a box

4.5 Practical details concerning verification

Because former implementations of the Miranda test and the Borsuk test in SONIC were only available in a draft implementation, not maintained and no longer functional, the author implemented those two tests anew. Among other things, the new versions handle memory space more efficiently and save facets and subfacets in lists. Moreover, special care was taken concerning the subdivision of symmetric subfacets (see section 4.5.6). The degree test was added to SONIC's toolbox. The implementation is complemented by a general handling of non-square systems. This involves the (non-trivial) handling for facets with fixed components in the verification tests and the two additional states *ContainsSolutionForSubsystem* and *UniqueSolutionForSubsystem*

The most important options for all verification methods can be chosen in the Settings-File. Among other options one can choose the function enclosure(s) for each test. At choice are the naive, centered and facet-centered evaluation with or without using Baumann's formula. Finally, detailed statistics for all verification tests have been added in the Statistics-File.

4.5.1 Lists of facets and subfacets

The algorithms of the Miranda, Borsuk and degree test were noted with one list $L_{\text{facet}}^{(u)}$ per subdivision level u. This description was chosen for a clear notation of the algorithms. In our implementations, however, only one single list L_{facet} is needed for each test to attain the same functionality. L_{facet} is initialized with the facet under consideration and traversed MaxDepth times.

Discarding elements is done as described for multiple lists. If it becomes necessary to subdivide a facet or subfacet $\boldsymbol{y}$ in the list, the next element to be considered is chosen as the successor of $\boldsymbol{y}$ in L_{facet} and $\boldsymbol{y}$ itself is replaced by its parts.

4.5.2 When to apply verification tests

As already mentioned, the verification routines can be started every time a new solution box is found or after all solution boxes have been determined. Both possibilities can be chosen in SONIC. Using the former option helps to reduce workload and data transfer in parallel computations (since the solution boxes are not distributed again for verification). For details about parallelization see section 5.4.

Verified solutions could also be written into a file instantly, reducing the memory requirements during the computation (not implemented). A (pre-existing) feature of SONIC is the option to stop after a single solution has been found and verified (*TerminationOnFirstHit*). If one is interested in finding only one solution of the system, this early termination can reduce the computation time dramatically.

4.5.3 Segregation of computation and verification

As a new option in SONIC, the verification can be segregated from the computation of the solution boxes. For this purpose we store the computed solution boxes (in binary format to avoid rounding errors due to conversion from and into decimal numbers). In the verification step the computation of the solution boxes can be skipped and the saved boxes are loaded and verified directly. Herein the stored data itself is not changed when running verification tests. Thus, we can execute different verification tests and settings on the exactly same boxes and without the need of computing the solution boxes anew.

4.5.4 Row-wise verification

In his thesis [Bee06] Beelitz comments on the possibility to apply the presented verification methods row-wise. He proposes to use the Miranda test first for every row if it fails to apply the Borsuk test and finally the degree test.

This is possible, but we do not recommend to do so. The reasons for this advice are as follows. Firstly, the Miranda test and degree test can only verify a solution in a given box after *all* pairs of facets have been considered. It can thus not be aborted in advance. Secondly, if we do not apply the Borsuk test for rows in which the Miranda test succeeds, its conditions are not checked on all facets. Thus we lose information about the considered values and weaken the test.

4.5.5 Subdividing facets

In all tests employing subfacets, different subdivision strategies can be used to split the facets and subfacets and to attain sharper function evaluations. Further one might wish to apply different subdivision strategies for the branch-and-bound algorithm and for verification. Thus the subdivision strategies in SONIC can be chosen separately for the branch-and-bound algorithm and the three verification tests utilizing subfacets. With this option it is, e.g., possible to make use of a sophisticated subdivision strategy for getting the best boxes for branch-and-bound and to choose a simple and cheap strategy within verification.

In verification we always apply an equidistant subdivision into two subfacets, gaps are not used. For determining the subdivision direction, similar subdivision strategies as in the basic branch-and-bound algorithm (see section 2.2.3) are utilized. Two options have been implemented by the author. One can choose between the simple MaxDiam strategy and a variation of the hybrid subdivision strategy (Alg. 2.4). In the latter, the ZeroNearBound strategy is excluded because it is not very powerful for small boxes (as solution boxes) [Bee06, p. 61] and we always assume a zero to be near to the boundary for a solution box.

The computational cost of all verification tests rises with the number of considered subfacets. To ensure the termination of the tests and to restrict their computational cost, it is thus sensible to limit the number of subfacets to be considered per facet or the subdivision depth. These limits can be chosen individually for every test. Of course, the tests may not need the maximum number of subfacets allowed.

In SONIC the number of subdivisions of a facet and of overall subfacets can be restricted in the Controls-File. The Statistics-File informs about the number of subfacets actually considered for each test.

For asymmetric facets subdivision is done independently for all facets. To attain symmetric facets for the Borsuk test, we need some further deliberations.

4.5.6 Subdivision of symmetrical facets

When we consider asymmetric facets and divide facet $\boldsymbol{y}$ in subdivision direction d, it does not matter whether we take the exact midpoint as subdivision point or whether the rounded midpoint $\mathrm{fl}(\mathrm{mid}\,(\boldsymbol{y}_d)) \in \mathbb{R}_M$ is used as long as $\mathrm{fl}(\mathrm{mid}\,(\boldsymbol{y}_d)) \in \boldsymbol{y}$. We could even subdivide at any other point in $\mathrm{int}\,(\boldsymbol{y}_d)$, but we prefer two subboxes of approximately the same size.

For attaining subfacets which are symmetric with respect to the midpoint of the given box, we have to take some precautions when subdividing. Clearly, we always have to choose the same subdivision direction $d \neq i$ for each pair of symmetric facets $\boldsymbol{x}^{i,+}$ and $\boldsymbol{x}^{i,-}$ or subfacets $\boldsymbol{x}^{i,+,k}$ and $\boldsymbol{x}^{i,-,k}$. As for the subdivision point, subdividing the facet $\boldsymbol{x}^{i,+}$ and $\boldsymbol{x}^{i,-}$ in their midpoints along the subdivision direction would always result in symmetric subfacets in exact arithmetic. The same holds true when subdividing a pair of symmetric subfacets. The downside of a computational test is that we lose this property when calculating in the finite set of machine numbers $\mathbb{R}_M$. This happens, whenever the rounded midpoints $\mathrm{fl}(\mathrm{mid}\,(\boldsymbol{x}_d^{i,+}))$ and $\mathrm{fl}(\mathrm{mid}\,(\boldsymbol{x}_d^{i,-}))$ themselves are not symmetric with respect to the midpoint of box $\boldsymbol{x}$. Figure 4.5 shows a scenario for which the intuitive approach would not result in symmetric facets.

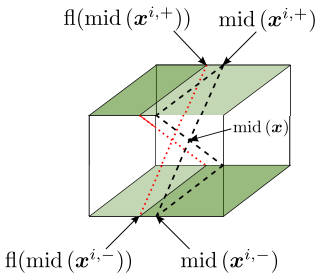


Figure 4.5: Symmetry of subfacets lost due to rounding

Nevertheless, it is still possible to compute with symmetric subfacets in an implementation. However, we will not attain a subdivision of the facets but only a covering. We still can prove the conditions of the verification tests since also the function enclosure over the elements of coverings include the range over the facets (see relation (1.8)). The author wants to propose two ways to attain symmetry.

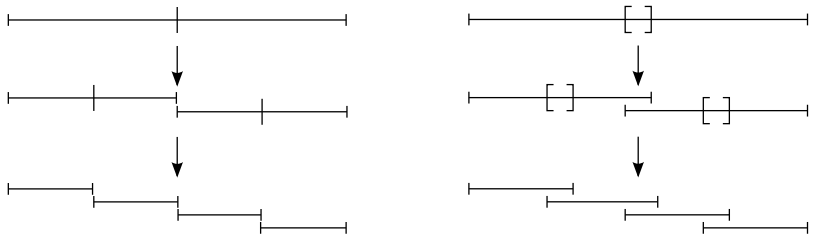
The underlying principle is the same for both options: we compute a covering with parts enclosing the symmetric subfacets. Since all computational verification tests are based on checking conditions for the function range over the facets, we only need to compute function enclosures including the true range over the facets. Thus, rigorous computations are ensured and whenever a solution can be verified,

we can be sure of its existence. Still, overestimation may lead the verification tests to fail although they would succeed for exact computations, thus weakening the verification tests.

A first ansatz to attain enclosures of symmetrical subfacets is to compute the midpoints as intervals. For facet $\boldsymbol{x}^{i,+}$ and subdivision direction d this results in a box $\boldsymbol{m}_d^{i,+} = [\underline{m}_d^{i,+}, \overline{m}_d^{i,+}]$ containing the true midpoint. We can then subdivide facet $\boldsymbol{x}^{i,+}$ in direction $d \in \{1, \ldots, n\} \setminus \{i\}$ by setting

$$\boldsymbol{x}_d^{i,+} = [\inf(\boldsymbol{x}_d^{i,+}), \overline{m}_d^{i,+}] \cup [\underline{m}_d^{i,+}, \sup(\boldsymbol{x}_d^{i,+})].$$

All other components of the subfacets are retained as in $\boldsymbol{x}^{i,+}$. The opposite facet $\boldsymbol{x}^{i,-}$ can be subdivided in the same manner. The resulting subfacets are guaranteed to contain the mathematically exact symmetrical subfacets. A downside of the proposed ansatz is that the overestimation of the subfacets grows each time we subdivide. This growth gets even larger when we subdivide more than once per direction. This happens since the interval $\boldsymbol{m}_d^{i,+}$ has to be considered as one boundary when subdividing again in direction d. When calculating the next midpoints we thus have to use the upper and lower bounds of interval $\boldsymbol{m}^{i,+}$. We sketched the situation in Figure 4.6.



(a) In exact arithmetic (b) Enclosing the midpoint

Figure 4.6: Computing subfacets (1D)

We counteract overestimation by using another approach to compute symmetrical subfacets. For this ansatz, one facet out of each pair of opposite facets is subdivided with the usual methods described in the last section—even if it is not subdivided at the exact midpoint. If we always subdivide $\boldsymbol{x}^{i,+}$, we attain subfacets $\boldsymbol{x}^{i,+,l}$. Now the symmetric subfacets $\boldsymbol{x}^{i,-,l}$ are computed with the help of interval analysis. Thus a box $\boldsymbol{m}$ enclosing the exact midpoint of the box $\boldsymbol{x}$ is needed. The symmetric subfacets are then yielded by

$$\boldsymbol{x}^{i,-,l} = 2\boldsymbol{m} - \boldsymbol{x}^{i,+,l}.$$

This interval computation results in a box which is guaranteed to contain the subfacet symmetric to $\boldsymbol{x}^{i,+,l}$ since for every $x \in \boldsymbol{x}^{i,+,l}$ and the exact midpoint m, the symmetrical point $2m - x$ is contained in $\boldsymbol{x}^{i,-,l}$. A sketch of the computed enclosures for subfacets is given in Figure 4.7.

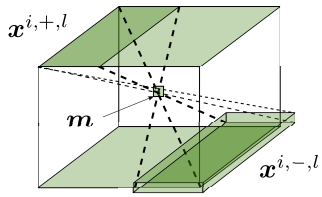


Figure 4.7: Computing an enclosure for subfacet $\boldsymbol{x}^{i,-,l}$ from the opposite subfacet $\boldsymbol{x}^{i,+,l}$ and an enclosure $\boldsymbol{m}$ of the midpoint of the box

We furthermore know that all points symmetric to $\boldsymbol{x}^{i,+,l}$ have to be part of the facet $\boldsymbol{x}^{i,-}$. Hence we reduce overestimation by replacing the computed $\boldsymbol{x}^{i,-,l}$ with

$$\boldsymbol{x}^{i,-,l} \cap \boldsymbol{x}^{i,-}.$$

For easier notation, the box resulting from the last intersection containing the symmetric subfacet is itself also referred to as a subfacet.

In an implementation the second approach has the additional advantage that only one subfacet has to be stored, the corresponding subfacet can be computed when needed. Note that we still have to accept overestimation of the enclosures of subfacets.

For the realization in SONIC the author picked the second approach to ensure the correct handling of symmetric subfacets.

4.5.7 Variations of the degree test

Just as well as the Borsuk test, the degree test can be realized in various ways. The condition $h(x,t) = 0$ for arbitrary $x \in \boldsymbol{x}$ and $t \in [0,1]$ is equivalent to

$$0 \in \text{range}_h(\boldsymbol{x} \times [0,1]).$$

Hence it is sufficient to prove the non-existence of zeros in the function enclosure for this range to fulfill the degree test. Of course, the function enclosure can be varied by utilizing the methods discussed earlier in this work. Namely these are the naive, centered or facet-centered evaluation, the method by Baumann and preconditioning. These modifications are well-known and applicable for all verification tests. They are thus not discussed in detail.

In this section further variations of the degree test are presented and investigated. In particular, we address the newly discussed and implemented issues of varying homotopies and subdivision strategies for the degree test.

4.5.7.1 Homotopy

The most simple way to alter the degree test is to change the considered homotopy. We also attain different realizations of the test by changing the terms used to express the homotopy because different expressions potentially lead to differing function enclosures. Function enclosures yielded by different expressions of one and the same homotopy can further be intersected to attain a sharper enclosure for the homotopy.

For tests concerning the strength of the degree test the following homotopies and formulations are examined. Herein formulation h_{1a} is the standard expression used in [FHL07, BFLW09].
First homotopy

$$h_{1a}(t,x) = t \cdot g(x) + (1-t) \cdot (x - \widetilde{x})$$
$$h_{1b}(t,x) = t \cdot (g(x) - (x - \widetilde{x})) + (x - \widetilde{x})$$

Second homotopy

$$h_{2a}(t,x) = t^2 \cdot g(x) + (1-t^2) \cdot (x - \widetilde{x})$$
$$h_{2b}(t,x) = t^2 \cdot (g(x) - (x - \widetilde{x})) + (x - \widetilde{x})$$

Third homotopy

$$h_{3a}(t,x) = t^{10} \cdot g(x) + (1-t^{10}) \cdot (x - \widetilde{x})$$
$$h_{3b}(t,x) = t^{10} \cdot (g(x) - (x - \widetilde{x})) + (x - \widetilde{x})$$

4th homotopy

$$h_4(t,x) = t^{10} \cdot g(x) + (1-t) \cdot (x - \widetilde{x})$$

5th homotopy

$$h_5(t,x) = t \cdot g(x) + (1-t^{10}) \cdot (x - \widetilde{x})$$

Numerical tests of these homotopies were conducted. The results are summarized in section 4.6.2.4.

4.5.7.2 Sufficient Condition

Another computationally verifiable sufficient condition for the non-existence of solutions of

$$h_{1a}(x,t) = 0 \text{ with } x \in \boldsymbol{x} \text{ and } t \in \boldsymbol{t}$$

was presented in [BFLW09]. Instead of a simple test for zeros in the range of $h_{1a}(x,t)$ on the box $\boldsymbol{x} \times \boldsymbol{t}$, a division in standard extended arithmetic is utilized in that condition.

Theorem 4.6 ([BFLW09])
Assume $\widetilde{x} \in \operatorname{int}(\boldsymbol{x})$, $t \in \boldsymbol{t}$ *and* $\boldsymbol{y} \subset \partial\boldsymbol{x}$ *and* $\boldsymbol{t} \subseteq [0,1]$. *For* $j \in 1, \ldots, n$ *let* $\boldsymbol{g}_j(\boldsymbol{y})$ *be a function enclosure for the preconditioned function component* g_j *over* $\boldsymbol{y}$. *If*

$$\bigcap_{j=1}^{n} \frac{\boldsymbol{g}_j(\boldsymbol{y})}{(\boldsymbol{y} - \widetilde{x})_j} \cap \left(1 - \frac{1}{\boldsymbol{t}}\right) = \emptyset$$

is fulfilled, homotopy h *(=*h_{1a}*) has no zero in* $\boldsymbol{y} \times \boldsymbol{t}$.

A proof of the theorem can be found in [BFLW09].

A third way to prove the non-existence of zeros of the homotopy is to analyze the homotopy itself by a result-verifying nonlinear solver. Some considerations and tests concerning this possibility can be found in [BFLW09]. An advantage of feeding the homotopy to a verified solver is that all techniques provided by the solver can be used to solve the problem. However, to apply our solver SONIC directly to the problem $h(x,t) = 0$, symbolic preprocessing would be needed for testing the homotopy and the additional computations would be very expensive. Hence this idea has not been realized.

4.5.7.3 Subdivision

Within the degree test not only the facets can be subdivided, but the interval $\boldsymbol{t} = [0,1]$ as well. In Algorithm 4.4 usually a fixed subdivision of $\boldsymbol{t}$ is employed. In [FHL07] it is proposed to subdivide interval $\boldsymbol{t}$ into a given number $tSub \in \mathbb{N}$ of disjoint intervals with equal width. The facets of $\boldsymbol{x}$ are subdivided iteratively by the MaxDiam strategy. One has to consider that the homotopy has to be computed on at least $tSub$ boxes for each subfacet of $\boldsymbol{x}$. Thus a small number is to be chosen for $tSub$ to restrain computational costs.

The author implemented the described strategy using an equidistant multisection for $\boldsymbol{t}$. The default value for $tSub$ in SONIC is 4. As a further improvement we allow not only the MaxDiam strategy to subdivide facets but the modified hybrid subdivision strategy described earlier.

We now want to introduce a further option for applying the existing subdivision strategies. We use them to subdivide the parameter interval $\boldsymbol{t}$ together with the boundary of the box. For this purpose, we accumulate $\boldsymbol{t}$ and the facets $\boldsymbol{x}^{i,+}$ and $\boldsymbol{x}^{i,-}$ from $\partial\boldsymbol{x}$ in interval vectors

$$\boldsymbol{x}_{\boldsymbol{t}}^{i,+} := \begin{pmatrix} \boldsymbol{x}^{i,+} \\ \boldsymbol{t} \end{pmatrix} \text{ and } \boldsymbol{x}_{\boldsymbol{t}}^{i,-} := \begin{pmatrix} \boldsymbol{x}^{i,-} \\ \boldsymbol{t} \end{pmatrix}.$$

We will speak of these vectors as *augmented facets*. For every augmented facet $\boldsymbol{y} = (\boldsymbol{z}, \boldsymbol{u})^T$ with $\boldsymbol{z} \in \mathbb{IR}^n$, $\boldsymbol{z} \subseteq \partial\boldsymbol{x}$ and $\boldsymbol{u} \in \mathbb{IR}$ with $\boldsymbol{u} \subset \boldsymbol{t}$ we define

$$h(\boldsymbol{y}) := h(\boldsymbol{z}, \boldsymbol{u})$$

such that we can directly reformulate the conditions for the degree test to augmented facets. The aim of our approach is to find that coordinate of the augmented facet $\boldsymbol{x}_{\boldsymbol{t}}^{i,+}$ (or $\boldsymbol{x}_{\boldsymbol{t}}^{i,-}$) with the largest influence on the function enclosures and to avoid having to fix the number of subintervals of $\boldsymbol{t}$. However, if employed on the augmented facets, the simple MaxDiam strategy may be less useful since the interval $\boldsymbol{t}$ is usually much bigger than the intervals in the facets $\boldsymbol{x}^{i,+}$ and $\boldsymbol{x}^{i,-}$ and only $\boldsymbol{t}$ would be subdivided. Note that the maximum number of considered subfacets per facet (*MaxSub*) is multiplied by *tSub* when subdividing the augmented facets. This is done for a fair comparison, since we additionally have to subdivide $\boldsymbol{t}$, not only the facets of $\boldsymbol{x}$.

For some of the subdivision strategies in SONIC the Jacobian matrix is needed. If it has been computed before, it is available without causing further computational costs. If the hybrid strategy is applied to the augmented facets, however, the Jacobian matrix of the homotopy is needed. This leads to higher computational costs than for a fixed subdivision of $\boldsymbol{t}$.

The modified degree test is displayed in Algorithm 4.8. Both strategies to subdivide $\boldsymbol{t}$ have been implemented and will be compared in the next section.

Algorithm 4.8 DEGREE TEST WITH COMBINED SUBDIVISION FOR $\boldsymbol{x}$ AND $\boldsymbol{t}$ (bounded box $\boldsymbol{x}$)

```
1: MaxSub_t := MaxSub · tSub
2: for i = 1, ..., n do
3:    for facet x_facet ∈ {x^{i,+}, x^{i,-}} do
4:       initialize list L^(1)_facet with (x_facet, t)^T
5:       n_sub ← 1
6:       for u = 1, ..., MaxDepth do
7:          while L^(u)_facet is not empty do
8:             take first augmented facet y in L^(u)_facet
9:             if y fulfills condition for degree test (0 ∉ h(y)) then
10:               delete y from list L^(u)_facet
11:            else {facet has to be divided}
12:               if u < MaxDepth and n_sub + 2 ≤ MaxSub_t then
13:                  subdivide y into two parts
14:                  n_sub ← n_sub + 2
15:                  insert parts into list L^(u+1)_facet
16:               else
17:                  exit {no further subdivision allowed, no solution verified}
18:               end if
19:            end if
20:         end while
21:      end for
22:   end for
23: end for
24: info(x) ← ContainsSolution
```

4.6 Comparison of the verification tests

All computational verification tests are based on numerical evaluations of function enclosures. With their help one can draw conclusions based on theorems. Thus computational tests only check sufficient conditions and may suffer from overestimation of the enclosures they rely on. In addition, one has to take into account the methods used to compute the function enclosures.

Due to these reasons we have to distinguish whether we compare theorems, theoretical tests or implemented tests. In this section we first focus on the strength of the theorems, afterwards on the strength of the tests they result in. Comparisons of the implemented tests follow in the subsection 4.6.2.

4.6.1 Theoretical comparisons

Note that there exist more verification methods than presented up to this point. One further example makes use of the following theorem dating back to Moore [Moo77].

Theorem 4.7 (Moore)
Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function, $\boldsymbol{S}$ an interval slope matrix, C a nonsingular preconditioning matrix in $\mathbb{R}^{n\times n}$, I the identity matrix in $\mathbb{R}^{n\times n}$, $\boldsymbol{x} \in \mathbb{IR}^n$ a box and $\widetilde{x} \in \boldsymbol{x}$ a point in the box. If the slope based Krawczyk operator

$$\mathcal{K}(\boldsymbol{x}, \widetilde{x}, \boldsymbol{S}, C) := \widetilde{x} - Cf(\widetilde{x}) + (I - C\boldsymbol{S})(\boldsymbol{x} - \widetilde{x})$$

fulfills

$$\mathcal{K}(\boldsymbol{x}, \widetilde{x}, \boldsymbol{S}, C) \subseteq \boldsymbol{x},$$

then $\mathcal{K}(\boldsymbol{x}, \widetilde{x}, \boldsymbol{S}, C)$ (and therefore $\boldsymbol{x}$) contains a zero of f.

By virtue of this theorem, Moore's existence test proves a zero by checking whether the Krawczyk operator $\mathcal{K}(\boldsymbol{x}, \widetilde{x}, \boldsymbol{S}, C)$ maps a box $\boldsymbol{x}$ into itself.

Like the Newton test and the Miranda test, Moore's existence test is based on Brouwers fixed-point theorem. By contrast, the Borsuk and degree test are based on properties of the topological degree.

In [FLS04] a version of the Miranda test is used that is comparable to the presented formulation of the Miranda test combined with a function evaluation in centered form and without the usage of subfacets. This test requires only one function evaluation at the point $\widetilde{x}$. It has been proved that for this special version the Miranda test is always at least as powerful as a test based on Moore's theorem. The paper also gives some conditions under which the tests are equivalent. (Nevertheless, we implemented a test based on Moore's theorem for numerical tests. Because it could not verify any solution for the tested problems, it has not been incorporated into SONIC.)

A further theorem applicable to verify solutions of interval systems is given by Kantorovich. In [AMFH02] the theorem of Kantorovich is compared with the theorems of Miranda and Borsuk. A generalization of Miranda's theorem for arbitrary norms is given and it has been proved that the Kantorovich theorem is a special case of this generalized theorem as well as a special case of Borsuk's theorem. For the exact conditions confer [AMFH02]. Remind that this is not a hierarchy of computational tests but of the theorems with respect to the existence of a zero. The theorem of Kantorovich can be used furthermore to prove the uniqueness of a zero. However, according to [Ral80], a computational test based on Kantorovich's theorem would be significantly more costly, but not much stronger than a test based on Moore's theorem. Thus we refrained from implementing this test.

In [FL05] it is proved that the Miranda theorem is a special case of Borsuk's theorem—under the assumption that no zeros exist on the boundary of the box. In theory this means that we can prove the existence of a zero with the Borsuk test whenever this is possible using the Miranda test. Thus the Borsuk test may succeed in more situations than the Miranda test. However, for numerical implementations one has to regard that the above claim is only true if there exists no zero in the function enclosure on the boundary of the box. (And a zero on the boundary is not unlikely for small solution boxes). In this case the Miranda test might still work if the Borsuk test fails.

In [FL05] it is shown that the Miranda as well as the Borsuk test become more powerful (but also more expensive) if we use facet-centered function evaluation instead of centered evaluation. (The center of the facet in this case is chosen as the projection of the box center.) Furthermore, it is proved that the Borsuk test with naive, centered and facet-centered function evaluation is successful in at least as many situations as the Miranda test using the same function evaluation. To illustrate this fact, the paper gives an example for which the Borsuk test is successful whereas Miranda- and Moore-based tests have to fail. Although, it is also mentioned that the superiority may not pertain for small boxes like those we want to verify.

[FHL07] gives a proof that the degree test is more general than the Miranda test, provided there exists no zero on the boundary of the box. Also given is an example for which the Miranda test has to fail whereas the degree test succeeds in verifying a solution.

[BFLW09] states that the degree test is more general than the test based on Moore's theorem and a test similar to the one given by Moore and Kioustelidis in [MK80]. The proof of this statement can be found in [FHL07].

Until now it is not shown that the degree test is more powerful than the Borsuk test. Nevertheless, numerical experiments seem to sustain this hypothesis. In [FHL07,BFLW09] a numerical example is given for which the degree test always performs better than the tests based on the theorems of Miranda and Borsuk if all

tests are conducted uniformly either without preconditioning or with the inverse midpoint preconditioner. Still, the Borsuk test can verify more solutions if applied in combination with the new preconditioner—but it is more costly as well. Recall furthermore that for the Borsuk test we could use different preconditioners for the facets whereas we have to use the same preconditioner for all subfacets for the Miranda and degree test. This gives an advantage to the Borsuk test if varying preconditioners are allowed when comparing the tests.

4.6.2 Numerical comparisons using SONIC

There are many implementation issues to concern when comparing the strength of numerical verification tests. It is possible to find examples of systems for which the Borsuk test can verify a solution although the degree test fails. This is caused by the implementation of the midpoint computation. For the considered small boxes and facets, the midpoint cannot always be computed exactly for numerical reasons. As for subdivision, the midpoint is evaluated as

$$\operatorname{mid}(\boldsymbol{x}) = \max\left\{\underline{x}, \min\left\{\overline{x}, \frac{\underline{x}+\overline{x}}{2}\right\}\right\}$$

and can be equal to one of the boundaries. Hence the calculated midpoint may also lie on the boundary of the box. If $\widetilde{x}$ is chosen as $\operatorname{mid}(\boldsymbol{x})$ for the degree test, this causes that $x - \widetilde{x} = x - \operatorname{mid}(\boldsymbol{x})$ may get zero, thus affecting the enclosure for the homotopy used in the degree test.

We can even find examples for which only the Miranda test verifies a solution because the considered box is too small to be compressed by the Newton test and the Borsuk as well as the degree test fail because the function enclosures over the boundary contain zero.

We now want to evaluate the tests implemented in SONIC. For this purpose the problems in test set $\mathcal{T}_V$ are used. Those contain all problems from test set $\mathcal{T}_B$ that yielded at least one solution box but no manifold of solutions. In a first step, the solution boxes have been computed with the box-intern hierarchy and stored without verification. Subsequently all tests were run on identical boxes.

Considering the test results, note again that the numbers of solutions and solution boxes does not have to coincide. It is possible that a solution is contained by more than one box or that a box contains more than one solution. It can further happen that boxes are contained in the list more than once if boxes containing the same solution were contracted to thin boxes containing the solution. These multiple occurrences of solution boxes are only diminished by collecting boxes *after* the verification tests have been applied.

In each test using facets we apply the naive, centered and facet-centered function enclosure as well as the centers proposed by Baumann (see sections 4.1.3.1 and 1.8.2.1). The further parameters for the facet-based tests are chosen uniformly.

Unless stated otherwise, the inverse midpoint preconditioner and the MaxDiam strategy are employed and the maximum number of subfacets considered per facet (*MaxSub*) is set to 8. (The maximum subdivision depth for subfacets (*MaxDepth*) is set to 100 and has no influence on the presented test results.) Furthermore, for the degree test we use a fixed subdivision for $\boldsymbol{t}$ into 4 parts by default (*Degree_FixedPartitionForT*=on, *Degree_PartsOfT=4*) and make use of the sufficient condition as given in Theorem 4.6.

It will become apparent that one should make use of epsilon-inflation to verify more solutions. Nevertheless, for the sake of more lucid comparisons, epsilon-inflation is not applied unless mentioned explicitly.

Note that a deviant choice of parameters can influence the received results of the verification tests seriously.

4.6.2.1 Epsilon-inflation

In a first step, an impression of the influence of the epsilon-inflation is given. As mentioned earlier, an epsilon-inflation can be helpful if applied in combination with a Newton test (since the interval Newton operator has to contract the box to verify a solution and may not be successful for boxes that are too small). Furthermore, by inflating the box, one tries to elude zeros on the boundary of a box which interfere with the Borsuk and degree test.

For our test, the verification strategy given in section 4.4 has been applied. The solution boxes were enlarged by an epsilon-inflation with $\epsilon = 0.25$ in every inflation step. Note that the verification strategy includes a separate application of the epsilon-inflation in the Newton method. For Newton and all other tests the same number of epsilon-inflations was allowed. Moreover, the verification scheme stops as soon as no test can provide any further information about the uniqueness or existence of a solution in a box.

Table 4.1 states the number of solution boxes for each problem as well as the number of verified boxes for each test and in total. For further comparisons, the number of subfacets considered for the Miranda test, Borsuk test and degree test is listed. Note, however, that this number cannot be used directly for a comparison of the computational costs of the verification tests (due to the different evaluations conducted in the computational tests). Verification tests are applied to an overall amount of 1492 solution boxes. If no epsilon-inflation is employed (*VerificationMaxEpsInflations*=0), 343 boxes can be verified. Note that in this case most boxes are verified by the Miranda test and only one by a Borsuk test. This implies that only existence can be proved for the verified boxes. We assume that the uninflated boxes are too small to be contracted by the interval Newton operator and zeros on the boundaries cause the Borsuk test and degree test to fail in most cases.

	boxes	verified	done tests				successful tests				subfacets		
no epsilon-inflation			N	M	B	D	N	M	B	D	M	B	D
G7_gradientsystem	320	192	320	320	128	128	0	192	0	0	3328	1408	704
Reactor	1	0	1	1	1	1	0	0	0	0	4	8	4
Brent7	128	110	128	128	19	18	0	109	1	0	1716	250	182
Eco9	16	8	16	16	8	8	0	8	0	0	181	90	56
Trigexp1	1	1	1	1	0	0	0	1	0	0	100	0	0
Trigonometric	1024	32	1024	1024	992	992	0	32	0	0	6762	10256	6359
DesignProblem9	1	0	1	1	1	1	0	0	0	0	6	10	6
7erSystem	1	0	1	1	1	1	0	0	0	0	7	10	16
total	1492	343	1492	1492	1150	1149	0	342	1	0	12104	12032	7327
one epsilon-inflation			N	M	B	D	N	M	B	D	M	B	D
G7_gradientsystem	320	320	320	256	128	128	192	128	0	0	2432	1408	704
Reactor	1	1	1	2	1	1	0	1	0	0	62	8	4
Brent7	128	128	128	47	19	18	99	28	1	0	582	250	182
Eco9	16	16	16	17	7	7	6	10	0	0	219	82	52
Trigexp1	1	1	1	0	0	0	1	0	0	0	0	0	0
Trigonometric	1024	1012	1024	1987	1004	1004	29	983	0	0	25878	10352	6431
DesignProblem9	1	1	1	2	1	1	0	1	0	0	24	10	6
7erSystem	1	1	1	2	1	1	0	1	0	0	21	10	16
total	1492	1480	1492	2313	1161	1160	327	1152	1	0	29218	12120	7395

Table 4.1: Success of the verification scheme when allowing no or one single epsilon-inflation (N=Newton test, M=Miranda test, B=Borsuk test, D=degree test)

Using epsilon-inflation many more boxes are verified. Indeed, for our test set, verification was successful for all but 12 boxes. Of the verified boxes, 327 are verified by the Newton test and contain unique solutions. When allowing to apply epsilon-inflation twice (or more often), the Newton test can verify unique solutions in all 1492 boxes. (The other tests are then not started again for the twice-inflated boxes.) Note that for manifolds this would not be possible.

4.6.2.2 Success of the verification tests

We now want to consider the verification tests separately and to present more detailed results. As a reminder: all tests have been conducted with the inverse midpoint preconditioner. Our findings are stated in Table 4.2.

Further tests not represented here show that the individual tests can verify little more solutions when allowing to consider more subfacets per facets. Setting this parameter to 100 instead of 8, the Miranda test can verify two more solutions, the Borsuk test and degree test three more solutions. As computation time is subordinate in verification, we recommend not to choose this parameter too small.

4.6.2.3 Borsuk test

For the Borsuk test the three available options for preconditioners are tested. Table 4.3 shows the results for each option. We see clearly that a preconditioning improves the results. However, for this special test set, the subfacet-based preconditioner could not be shown to perform superior to the inverse midpoint preconditioner.

4.6.2.4 Degree test

In the preceding sections referring the degree test we spoke about different options that can influence its performance.

Tests showed that the discussed evaluation strategies yield nearly the same results for the homotopies and the sufficient condition given in Theorem 4.6. Only the application of the new test in connection with the homotopies h_{1b}, h_{2b} or h_{3b} are slightly superior to their counterparts h_{1a}, h_{2a} or h_{3a} as well as h_4 and h_5 with respect to the considered number of facets. For all homotopies and the changed sufficient condition (Theorem 4.6) the existence of solutions in 149 boxes could be verified by the degree test (without epsilon-inflation).

More varying results could be observed for the subdivision strategies applied in the degree test. Although the results do not vary for $tSub$ between 2 and 10 or between MaxDiam and the hybrid strategy, differences can be observed when subdividing $\boldsymbol{t}$ differently (cf. Table 4.4). The results indicate that a fixed subdivision for $\boldsymbol{t}$ should be preferred over a subdivision of augmented facets.

	boxes	done tests				successful tests				subfacets		
no epsilon-inflation		N	M	B	D	N	M	B	D	M	B	D
G7_gradientsystem	320	320	320	320	320	0	192	0	0	3328	2688	1376
Reactor	1	1	1	1	1	0	0	0	0	4	8	4
Brent7	128	128	128	128	128	0	109	94	108	1716	1888	1723
Eco9	16	16	16	16	16	0	8	6	8	181	208	184
Trigexp1	1	1	1	1	1	0	1	0	1	100	8	100
Trigonometric	1024	1024	1024	1024	1024	0	32	26	32	6762	10926	6999
DesignProblem9	1	1	1	1	1	0	0	0	0	6	10	6
7erSystem	1	1	1	1	1	0	0	0	0	7	10	16
total	1492	1492	1492	1492	1492	0	342	126	149	12104	15746	10408
one epsilon-inflation			N	M	B	D	N	M	B	D	M	BD
G7_gradientsystem	320	640	448	640	640	192	320	320	320	5120	7168	5856
Reactor	1	2	2	2	2	0	1	1	1	62	66	62
Brent7	128	256	147	162	148	99	128	128	128	1982	2364	2003
Eco9	16	32	24	26	24	6	15	15	15	307	372	306
Trigexp1	1	2	1	2	1	1	1	1	1	100	108	100
Trigonometric	1024	2048	2016	2022	2016	29	1012	1012	1012	26458	30790	26703
DesignProblem9	1	2	2	2	2	0	1	1	1	24	28	24
7erSystem	1	2	2	2	2	0	1	1	1	21	24	30
total	1492	2984	2642	2858	2835	327	1479	1479	1479	34074	40920	35084
two epsilon-inflations			N	M	B	D	N	M	B	D	M	BD
G7_gradientsystem	320	768	448	640	640	320	320	320	320	5120	7168	5856
Reactor	1	3	2	2	2	1	1	1	1	62	66	62
Brent7	128	285	147	162	148	128	128	128	128	1982	2364	2003
Eco9	16	42	25	27	25	16	15	15	15	311	380	310
Trigexp1	1	2	1	2	1	1	1	1	1	100	108	100
Trigonometric	1024	3043	2028	2034	2028	1024	1012	1012	1012	26506	30886	26763
DesignProblem9	1	3	2	2	2	1	1	1	1	24	28	24
7erSystem	1	3	2	2	2	1	1	1	1	21	24	30
total	1492	4149	2655	2871	2848	1492	1479	1479	1479	34126	41024	35148

Table 4.2: Success of the individual verification tests with no, one or two allowed epsilon-inflations (N = Newton test, M= Miranda test, B=Borsuk test, D= degree test)

	boxes	no preconditioning		inverse midpoint		subfacet based	
		successful tests	subfacets	successful tests	subfacets	successful tests	subfacets
G7_gradientsystem	320	0	2688	0	2688	0	2688
Reactor	1	0	8	0	8	0	8
Brent7	128	0	1158	94	1888	23	1618
Eco9	16	0	134	6	208	0	134
Trigexp1	1	0	8	0	8	0	8
Trigonometric	1024	0	8216	26	10926	0	8328
DesignProblem9	1	0	8	0	10	0	8
7erSystem	1	0	10	0	10	0	10
total	1492	0	12230	126	15746	23	12802

Table 4.3: Success of the Borsuk test with different preconditioners, without epsilon-inflation

	fixed subdivision of t		subdivision of the augmented facets	
	successful tests	facets	successful tests	facets
G7_gradientsystem	0	1376	0	1856
Reactor	0	4	0	4
Brent7	108	1723	90	1690
Eco9	8	184	6	152
Trigexp1	1	100	0	7
Trigonometric	32	6999	21	6164
DesignProblem9	0	6	0	6
7erSystem	0	16	0	5
total	149	10408	117	9884

Table 4.4: Different subdivision strategies for parameter t of the degree test (using homotopy t and the hybrid subdivision strategy)

Chapter 5

SONIC

The hardest thing is to go to sleep at night, when there are so many urgent things needing to be done. A huge gap exists between what we know is possible with today's machines and what we have so far been able to finish.

Donald E. Knuth

As mentioned earlier, the acronym SONIC stands for **S**olver and **O**ptimizer for **N**onlinear Problems based on **I**nterval **C**omputation. SONIC has been developed at the RWTH Aachen and the University of Wuppertal. Its main contributors include Dr. Thomas Beelitz, Prof. Dr. Bruno Lang, Dipl. Math. Klaus Schulte Althoff, Prof. Dr. Peer Ueberholz, Dr. Paul Willems and the author. SONIC was partially supported by VolkswagenStiftung within the project "Konstruktive Methoden der Nichtlinearen Dynamik zum Entwurf verfahrenstechnischer Prozesse".

The initial goal in its development was to solve dynamic systems arising from chemical processes [BBLW04, BBLSA04, MMB+07]. However, in the meantime SONIC has been augmented to a rigorous general purpose solver for nonlinear systems of equations and enhanced by an optimizer.

The implementation of the mathematical methods presented in the previous chapters has been discussed next to the theory. Parameters for adapted or newly implemented methods can be found there.

The goal of the current chapter is to give a brief glimpse into further, previously implemented methods. Although this chapter is not a manual, it provides some technical background for potential users of SONIC.

Our presentation is strongly influenced by the current version of SONIC as well as previous publications such as [BBLW04], [Bee06] and [BLB06].

5.1 Features and concept

The features of SONIC comprise, but are not limited to, the possibilities yet described in this work. This section states SONICs main features for a clear overview. Those features not discussed within the last chapters will be introduced briefly in the following.

The main features of SONIC include

- the implementation in C++
- the storage of many settings, all problems and results in plain text form
- default settings that are suitable for a wide range of problem classes
- easy modification of parameters by the user
- the possible utilization of three basic interval libraries
- the handling of unbounded intervals (including a special stopping criterion)
- internal storage of finite unions of intervals
- a solver for nonlinear systems of equations utilizing
  - a basic branch-and-bound scheme
  - various subdivision strategies
  - contractors such as
    - constraint propagation
    - Taylor refinement of first and second order
    - an interval Newton method (row-wise, hybrid version)
  - preconditioners such as
    - the inverse midpoint preconditioner
    - a width optimal preconditioner (C^W-LP-preconditioner)
  - multiple extended systems combined with
    - the possibility to select the extended systems to be utilized in computations
    - two sophisticated strategies to apply these systems
    - many parameters to adapt the strategies for specific problems
  - verification of solution boxes
    - including the handling of non-square systems
    - verification scheme utilizing epsilon-inflation and different verification tests (Newton test, Miranda test, Borsuk test and degree test)

  - methods to collect solution boxes
  - the option to execute computation and verification separately

- a verified solver for optimization problems

- parallel versions for

  - shared memory (OpenMP)
  - distributed memory (MPI).

In addition, a restart facility is implemented, allowing the user to interrupt the calculation after a predefined amount of time (*CheckpointingTime*) and to continue later. If one is only interested in whether or not a system has a solution in a given startbox, one can profit from another option of SONIC (*TerminationOnFirstHit*). It stops the calculation as soon as a single verified solution has been found and can save huge amounts of computation time.

Most features of SONIC can be activated in the Controls-File. In the same file one finds a variety of parameters to adjust the features to special needs. The default options are chosen to work well for a wide range of problems. cd Work If a user is, however, inclined to adjust the parameters for a special problem, this can be accomplished by simple changes in the Controls-File.

5.2 Install and run SONIC

General advice for building and running SONIC can be found in the *readme* in the main folder of SONIC. Note that in addition to the program itself at least one interval library has to be installed.

All test problems are given in files called *problemname.in* and are provided in the folder *data*. The problem files carry information about the variables, the desired precision of the solution boxes and the constraint system. A commented example file can be found in *example.in*.

The folder *settings* contains the Controls-File, in which parameters can be adapted to adjust SONIC, as well as a file called *Rules*. The latter contains the rules applied for expression optimization (see section 1.8.7)

Results are written into a folder named *results* into a subfolder with the same name as the problem. The two most important files in this folder are the Statistics-File in *problemname.sta*, giving overall statistics for the solver and for the individual methods, and *problemname.res*, containing the computed solution boxes.

5.3 Interval libraries

SONIC was designed to make use of different basic interval libraries. By now we can work with

- C-XSC [HK03, cxs],
- filib++ [LTG+06, fil] and
- the interval library by Sun [Sun].

These libraries implement the basic operations of interval arithmetic with special care to performance factors such as the number of rounding mode changes.

As mentioned before, SONIC can handle unbounded variables. A further special feature is the possibility to store finite unions of boxes and to compute with the hull of this union until the additional information is exploited for subdivision (as discussed in section 2.2.1). For realizing these features SONIC makes use of a special interval class taking care of unbounded intervals and unions of intervals. This class serves as a wrapper. Interval computations for bounded intervals are forwarded to the interval libraries. For more details concerning this part of the implementation see [Wil04].

For comparable results, the C-XSC library in version 2.2.4 has been utilized for all tests shown until now. A test with the recent version 2.5.3 yielded nearly uniform savings in computation time by a factor of 0.75. We also tested filib++ in version 3.0.2. It could reduce the average timing by a factor 0.6. However, the time savings differed considerably between the test problems (for timings see Table 5.2). Unfortunately, for one test problem, the solution was lost when employing filib++. Up to this point the author could not detect the exact source of this error.

5.4 Parallelization of the nonlinear solver

In addition to sophisticated algorithms and techniques, SONIC is provided with two efficient parallel versions. The author will only outline the basic parallelization strategies in SONIC. This is done according to [Bee06, BLB03, BLB06, Ueb07]. More specific information can be found in these publications.

5.4.1 Fundamental concepts

Parallelization enables us to tackle tasks that require immense computing power by conducting computations in parallel whenever possible. The improvement of running the parallel version of an algorithm on more than one execution unit (which we call a *core*) is measured by the speedup.

Definition 5.1 (Speedup)
The speedup *of a parallel algorithm is defined as*

$$\text{Speedup}(p) := \frac{\text{time}_1}{\text{time}_\text{p}}$$

where time_1 *is the computation time needed when running the algorithm on one core and* time_p *is the computation time of the same algorithm running on* p *cores.*

The optimal speedup is equal to the number of cores. To attain it, the whole algorithm is required to run in parallel on all cores from beginning to the end of the calculation. Optimal speedup is rarely reached since portions of the algorithm cannot be performed in parallel. A relation between the maximal speedup and the fraction of the algorithm that can be run in parallel is given by Amdahl's Law.

Theorem 5.1 (Amdahl's law [SCB05])
If p *cores can work in parallel on a fraction* $a \in [0,1]$ *of a given algorithm and the remaining part of the algorithm has to be run in serial, the maximum attainable speedup is given as*

$$\text{Speedup}(p) = \frac{1}{(1-a) + \frac{a}{p}}.$$

Thus, to obtain the best possible speedup, one has to maximize the fraction of the algorithm that is run in parallel. If the fraction is too small, this necessarily leads to a low speedup even if a high number of cores is employed (an upper bound for the speedup is given by $\sup_{p\in\mathbb{N}} \text{Speedup}(p) = 1/(1-a)$). For example, if only half of a algorithm can be run in parallel ($a = 1/2$), the best speedup achievable is 2—independent of the number of cores ($\sup_{p\in\mathbb{N}} \text{Speedup}(p) = 2$).

For further details concerning parallel computations we refer to the dedicated literature.

In SONIC parallelization can be applied in two "levels". Firstly, all boxes can be analyzed independently. Secondly, for each box the enclosures of the function components and the entries of the Jacobian matrix can be computed concurrently.

5.4.2 OpenMP

OpenMP (**Open** **M**ulti**p**rocessing) is used for the parallelization using shared memory for parallel threads. Two approaches to integrate OpenMP into the framework of SONIC were examined in earlier works [BLB03, BLB06].

One is an easy to implement method that analyzes all boxes in a given recursion level in parallel. After completing one recursion level r, computations are started for the next level $r+1$. The approach is working fairly well for low core numbers. Given a high core number and few boxes per recursion level, many cores will be idle during the computation. A further downside of the approach is its load balancing if it is not appropriate, again there will be idle cores. These cores could already

work on a box of the next recursion level but have to wait until all work on the current level is completed.

The second approach harnesses a task queue for better load-balancing. It was shown to be preferable to the first approach [BLB06].

5.4.3 MPI

MPI (**M**essage **P**assing **I**nterface) is used widely for parallelization with distributed memory. Again, two strategies are available in SONIC.

The first is a master-worker algorithm. Given p cores, one of them acts as the master and does not analyze any boxes but only distributes them to the workers. The workers solely analyze boxes and send them back to the master. Obviously, the master-worker approach can attain a maximum speedup of $\sup_{p\in\mathbb{N}} \text{Speedup}(p) = p - 1$.

In a second algorithm all cores analyze boxes. For load-balancing the cores exchange information about their workload and, if required, also boxes. A distinguished process keeps track of additional information about the work already done. This ansatz was shown to offer a nearly perfect speedup [Ueb07] and was successfully applied for computational proofs concerning the existence of spherical t-designs as described in [BLUW09]. Furthermore, comparing numerical studies indicate a better performance of the MPI versions in relation to the OpenMP implementations [BLB06].

◇

In [BLB03] a hybrid parallelization scheme was proposed. It applies different parallelization strategies for the two levels of parallelism. MPI is employed to run different boxes in parallel, OpenMP for parallel calculations on each box.

5.5 Optimization

Interval analysis does not only allow to solve nonlinear systems of equations reliably, but also to find guaranteed global solutions of nonlinear optimization problems.

This section gives a definition for nonlinear optimization problems followed by an overview of the most important methods to solve nonlinear optimization problems using interval methods. (Our description relies on [Bee06].) Herein only minimization problems are considered. Maximization problems can easily be transformed into this formulation by a simple negation of the objective ($\min\{-f(x)\}$).

Definition 5.2 (Nonlinear optimization problem)
Let m, n and r be positive integers. For an arbitrary objective function $f : \mathbb{R}^n \to \mathbb{R}$

and a startbox $\boldsymbol{x}^{(0)}$ *as well as the inequality constraints* $p : \mathbb{R}^n \to \mathbb{R}^m$ *and equality constraints* $q : \mathbb{R}^n \to \mathbb{R}^r$ *the problem*

$$\min f(x) \quad \text{with } x \in \boldsymbol{x}^{(0)} \subseteq \mathbb{R}^n$$

under

$$\begin{aligned} p_i(x) &\leq 0 \quad (i = 1, \ldots, m) \\ q_j(x) &= 0 \quad (j = 1, \ldots, r) \end{aligned}$$

is known as an optimization problem.

If only the objective is given ($m = r = 0$)*, this problem is an* unconstrained optimization problem*. In any other case we speak of a* constrained optimization problem*.*

The aim of global optimization in interval analysis is to find a sharp enclosure of the global minimum f^* for an optimization problem and a list of small enough boxes comprising the set of solutions or minimal points

$$\{x \in \boldsymbol{x}^{(0)} \mid f(x) = f^*\}$$

for the unconstrained optimization problem or

$$\{x \in \boldsymbol{x}^{(0)} \mid f(x) = f^*,\ p(x) \leq 0,\ q(x) = 0\}$$

for the constrained optimization problem.

5.5.1 General approach

As for the solution of nonlinear systems of equations, a branch-and-bound algorithm is the backbone of the optimization strategy. The implementation in SONIC is based on investigations by Berner [Ber95]. The basic algorithm is initialized with the startbox $\boldsymbol{x}^{(0)}$, stores boxes evolving in the branch-and-bound algorithm in a working list and yields a solution list covering the set of minimal points. As for nonlinear systems of equations, the boxes evolving in the branch-and-bound algorithm can be stored either in a list or a heap. The box to be considered next in the algorithm is chosen as the one minimizing $\inf(\boldsymbol{f}(\boldsymbol{y}))$ over all boxes $\boldsymbol{y}$ in the working list.

For subdividing boxes, the current implementation follows the recommendations in [Ber95]. Per default a box is subdivided into four subboxes. However, most of the subdivision strategies for nonlinear systems of equations could be applied as well.

As for nonlinear systems, we speak of a contractor if a function maps a box into itself while preserving all contained solutions. Some basic strategies will be presented in the following sections. We refer to the books of Hansen [Han92] and Kearfott [Kea96b] as starting points to gain more information about the rigorous solution of optimization problems.

5.5.2 Unconstrained optimization

Several contractors have been developed to approach unconstrained optimization problems by the means of interval analysis. Using them speeds up the branch-and-bound algorithm. The contractors for unconstrained optimization are presented in order of ascending requirements with regard to the objective function.

5.5.2.1 Cut-off-test

Since it only makes use of function enclosures, the *cut-off-test* can be applied for arbitrary functions. It calculates an upper bound $\widetilde{f}$ for the global minimum. Since it is obvious that no box $\boldsymbol{x} \subseteq \boldsymbol{x}^{(0)}$ fulfilling $\boldsymbol{f}(\boldsymbol{x}) > \widetilde{f}$ or simply $\inf(\boldsymbol{f}(\boldsymbol{x})) > \widetilde{f}$ can contain a minimal point, those boxes can be discarded.

The upper bound $\widetilde{f}$ can further be updated during the computation. Updating with the infimum of the function evaluation of a box in $\boldsymbol{x}^{(0)}$ seems tempting. However, function enclosures overestimate the true range of a function, thus we cannot infer the minimum of the function from (proper) function enclosures. A guaranteed upper bound $\widetilde{f}$ can be attained by updating with the minimum of the former value of $\widetilde{f}$ and the supremum of the function enclosure over the box $\boldsymbol{x}$ or at some point $\widetilde{x} \in \boldsymbol{x}$. To avoid rounding errors, in implementations we would have to calculate $\widetilde{f}' = \min\left\{\widetilde{f}, \sup\left(\boldsymbol{f}(\mathrm{flint}(\widetilde{x}))\right)\right\}$ for some $\widetilde{x} \in \boldsymbol{x}$ with $\mathrm{flint}(\widetilde{x}) \in \boldsymbol{x}$.

5.5.2.2 Monotonicity test

Monotonicity can be checked in every variable x_i in which the objective is continuously differentiable. If the function is strictly monotonic in x_i on a given box, the box can either be contracted or discarded (see Algorithm 5.1).

5.5.2.3 Non-convexity test

If a function is twice continuously differentiable, one can make additional use of information about second derivatives. For example, the Hessian matrix can be utilized to determine whether a function is non-convex. Because for boxes in the interior of the startbox, convexity is a necessary condition for a local minimum (and therefore also for a global minimum in the interior), a box $\boldsymbol{x} \subseteq \mathrm{int}\left(\boldsymbol{x}^{(0)}\right)$ can be discarded by showing $\boldsymbol{f}(\boldsymbol{x})$ to be non-convex. Boxes intersecting the boundaries of $\boldsymbol{x}^{(0)}$ have to be considered separately. They can only be reduced to their intersection with the boundary of the startbox $\boldsymbol{x}^{(0)}$. For a computational test, the fact that every convex function has a positive semidefinite Hessian matrix is utilized. In turn, a necessary condition for positive semidefiniteness of the Hessian is the non-negativity of all its diagonal elements [Han92, p. 125]. The last condition can be checked using interval analysis.

Algorithm 5.1 MONOTONICITY TEST (startbox $\boldsymbol{x}^{(0)}$, box $\boldsymbol{x}$, objective f)

for $i = 1, \ldots, n$ **do**
  if $\sup\left(\left[\frac{\partial f}{\partial x_i}\right]\right) < 0$ **then** $\{f$ strictly decreasing in $x_i\}$
    if $\sup(\boldsymbol{x}_i) = \sup(\boldsymbol{x}_i^{(0)})$ **then**
      $\boldsymbol{x} \leftarrow (\boldsymbol{x}_1, \ldots, \sup(\boldsymbol{x}_i), \ldots, \boldsymbol{x}_n)^T$
    else
      discard $\boldsymbol{x}$
    end if
  end if
  if $\inf\left(\left[\frac{\partial f}{\partial x_i}\right]\right) > 0$ **then** $\{f$ strictly increasing in $x_i\}$
    if $\inf(\boldsymbol{x}_i) = \inf(\boldsymbol{x}_i^{(0)})$ **then**
      $\boldsymbol{x} \leftarrow (\boldsymbol{x}_1, \ldots, \inf(\boldsymbol{x}_i), \ldots, \boldsymbol{x}_n)^T$
    else
      discard $\boldsymbol{x}$
    end if
  end if
end for

5.5.2.4 Special Newton process

For twice continuously differentiable functions there exists a further possibility to contract a box $\boldsymbol{x} \subseteq \operatorname{int}(\boldsymbol{x}^{(0)})$ without losing any minimal points. Since the gradient has to be zero in each local minimum in the interior of the startbox, one simply applies the interval Newton method to the gradient system $\nabla f(\boldsymbol{x}) = 0$. Thus the given box can be contracted preserving all zeros of the gradient system and therefore all the minimal points of the original optimization problem.

5.5.3 Constrained optimization

As an important restriction in constrained optimization, one always has to check the feasibility of the boxes. In the context of interval analysis feasibility means that a box contains at least one element x fulfilling the constraints $p(x) \leq 0$ and $q(x) = 0$.

One well known method to handle constrained optimization problems is to check necessary conditions for local and global minima, namely the Fritz John conditions. Consult for instance [Han92].

5.5.4 Parallelization of the optimizer

A distributed memory parallelization of the unconstrained optimization part of SONIC is available. This parallelization goes back to the work of Berner [Ber95].

Its backbone is a master-worker approach, augmented by a propagation of new upper bounds for the global minimum between the cores. Test results in [Bee06] show that the parallel version offers a good speedup for "hard" problems. For other problems many cores are said to be working on boxes that are "not promising", meaning unlikely to contain an optimal point. The author refers to [Bee06] and [Ber95] for further details and an analysis of the strategy.

5.6 Other nonlinear solvers and optimizers

Note that time comparisons to other implementations are difficult. Overall comparisons only have a restricted validity since they do not take into consideration that different implementations may use different languages, methods and varying strategies to combine these methods. Thus even setting parameters appropriately cannot make different implementations comparable.

GlobSol

GlobSol is an interval optimizer, but has an option for solving nonlinear systems. Comparisons of an older version of SONIC with GlobSol in the version of 2003 can be found in [Bee06]. It shows favorable results for SONIC.

The officially available version is from 2003. By courtesy of Baker Kearfott, we could conduct some further tests with a more current developers version (end of 2012, with settings suiting for *7erSystem*). Although the performance of GlobSol might be improved in the next release version, these tests showed SONIC to be competitive in the solution of nonlinear interval systems.

problem	GlobSol	SONIC (BIH, C-XSC)
Brent7	92.4	29.3
Trigexp1	>36000	31.7
Trigonometric	>36000	133.9
DesignProblem9	279.5	112.2
7erSystem	6195.4	79.7

Table 5.1: Comparison of the computation time in GlobSol and SONIC

Ibex

We further chose to compare our solver to the program Ibex, which contains a ready-to-use solver for nonlinear systems and an optimizer. Ibex is a customizable interval software which is developed by a group of researchers around Gilles

Chabert. It is available online [ibe], can deal with different interval libraries (Gaol, filib++ and Profil/Bias) and contains some other techniques than these described in this work.

We conducted our tests with the solver for nonlinear problems in version 2.0.6 of Ibex. An overview of the results is shown in Table 5.2. To achieve comparable results, both SONIC and Ibex are run on the same machine and with the same version of filib++.

problem	Ibex	SONIC
min-04-06	5.6	0.02
G7_gradientsystem	5.0	4.5
Reactor	10.4	-
Chemistry1	24.38	5.0
Brent7	20.46	1.9
Eco9	45.53	16.1
Trigexp1	0.5	(31.7)
Trigonometric	60.38	44.7
Chemistry2	61.76	207.3
DirectKinematics	36.15	35.7
DesignProblem9	58.77	36.0
7erSystem	55.64	49.6
min-04-07	556.49	97.0
Chemistry3	8265.48	40296.9

Table 5.2: Comparison of the computation time in SONIC and Ibex

We want to add some remarks concerning this table. Firstly, we did not run problem *Reactor* in Ibex, since it has different precision requirements ψ for the variables and the default input in Ibex is one uniform value. (Even though, according to Gilles Chabert, the subdivision strategies can be adapted to model differing ψ for the variables.) Secondly, as said before there is some problem in computing the solution for *Trigexp1* using SONIC in combination with filib++. In this case we gave the measurement using C-XSC. For this problem the result of Ibex is remarkably good, demanding further investigations into the techniques used to attain this result.

As well as SONIC, Ibex is able to handle unbounded intervals, but there exists no stopping criterion like our criterion using the threshold vector Ψ. For the single unbounded problem *Chemistry1* in our test set this was no hurdle. (It was, however, when we changed one problem manually to contain two intervals $[-\infty, \infty]$ in the startbox $\boldsymbol{x}^{(0)}$. In this case the solver produced an enormous number of solution boxes containing intervals like $[\infty]$.)

Overall, we see that each solver (with default options) wins the comparison in

computation time for some of the test problems. To the author, the differences in computation time are caused by two main reasons: the different techniques in the solvers and the varying strength of the default settings for the individual test problems. As we already saw for SONIC, computation time relies heavily as well on the subdivision techniques as on the contractors. We also know that the performance of the solver for a single problem is strongly influenced by the parameters used for its solution. For example, when simply applying Taylor methods of second order for *Brent7* we can easily reduce the computation time of SONIC by over 60% and the advantage of Ibex over SONIC is reduced considerably. The timings of SONIC could further be reduced by running a parallel version (which is not yet possible in Ibex).

$\diamond$

There exist further programs for the reliable solution of nonlinear systems of equations and optimization problems. However, most of them are not freely available or the available versions are restricted in functionality and do not fulfill our requirements (as for variable numbers).

An informative comparison of existing global constrained optimization software can be found in [NSHV05].

Conclusion

Knowledge is an unending adventure at the edge of uncertainty.

Jacob Bronowski

We conclude our work with an outline of the accomplishments presented in the last chapters and give suggestions for future research.

Accomplishments

The author has developed general concepts as well as specific modifications of existing strategies for solving nonlinear systems of equations based on interval analysis. Most suggested techniques have been implemented for numerical tests and enhanced the functionality of the verified solver SONIC. However, the strategies are not bound to this implementation, but can also provide improvements for other nonlinear solvers. Our contributions mainly comprise the following concepts and achievements.

The available set of subdivision strategies has been augmented. For example, subdivision with a shifted subdivision point has been implemented and evaluated by a numerical study. Further, the handling of unbounded variables has been improved by an appropriate handling of multiple unbounded components, an adjusted subdivision and an additional stopping criterion for half-bounded boxes with large mignitude of at least one component. We further gave some test results concerning a trisection of intervals.

For applying extended systems, an adaptive strategy has been derived. Compared to the pre-existing box-intern hierarchy, this strategy has shortened the average computation time over our test sets significantly without evoking any negative outliers. The new strategy relies on data measured during the computation of a given nonlinear system of equations and allows the solver to adapt to individual problems as well as the underlying soft- and hardware structure. We

have shown that the modus operandi and the standard parameters of the strategy are sensibly chosen.

Considerable efforts were made to reduce the computation time spent on each single extended system. Two influential time factors in the application of extended systems are the considered variables and the computation time needed for optimal preconditioners. Thus the author scrutinized how to choose the considered variables and how to save computation time in the calculation of preconditioners. We further provided an implementation to choose the variables based on structural properties of the constraint system and revealed first test results.

Two verification tests have been implemented anew to make them usable in SONIC. The degree test has been inserted into SONIC for the first time. In addition, different variations of the available verification tests have been implemented and assessed.

For subdivision in the verification methods more sophisticated strategies have been harnessed. The author has further formulated a general verification scheme combining the available verification tests with stepwise epsilon-inflation. This scheme can even handle non-square systems. To verify them, the verification tests are invoked for a square subsystem. Subsequently, the information about existence and uniqueness of solutions is adjusted automatically to fit the originally given non-square system. For underdetermined systems, several options for fixing variables were provided and studied with respect to their mathematical statements.

An algorithm for finding and handling intersecting boxes that contain unique solutions has been proposed and implemented. In addition, the computation and verification of solutions can now be executed independently. Thus, if needed, solution boxes can be verified with the help of different verification methods or combinations of verification methods.

Concerning the general implementation of SONIC, the author has removed some bugs and enhanced the functionality, e.g., by a corrected power function. Further the program has been made more user-friendly, e.g., by appropriate outputs to support the user with the correct mathematical interpretation of the results.

We have set default values for all studied parameters. Those have been determined to ensure a good performance of SONIC for various problem types. The most influential parameters have been incorporated into the Controls-File for modification by the user. Thus the solver can easily be customized to special problems or problem classes.

Perspectives

In addition to earlier remarks concerning possible improvements in the corresponding sections of this work, the most promising directions for further development should be summarized at this point. The following suggestions comprise ideas to enhance the theory of the reliable solution of nonlinear systems of equations as well as pointers on how to augment the implementation of SONIC.

First of all, further methods for the calculation of function enclosures could provide sharper or faster evaluations of the true function range. This is a common goal in interval analysis and would contribute to an overall reduction of the computation time of the solver. The savings could be achieved by reducing the number of boxes needed in the branch-and-bound by sharper function enclosure or directly by reducing the time needed to analyze a single box.

Candidates for function enclosures of inverse functions are inverse Taylor models as presented in [Hoe01] [BH01] [HB02]. Although these are very costly, an implementation of inverse Taylor models is envisaged, associated with a study whether our solver can benefit from their utilization.

An ansatz related to the concept of extended systems lies in the design of customized extended systems with respect to the structural properties of a given problem or problem class. However, designing those systems is complicated due to the influence of preconditioning and linearizations in the contractors. Which variables should be considered on extended systems is another question worth further investigation—especially with respect to the usage of time-consuming optimal preconditioners. It may also be possible to achieve better results just by improving the algorithm for finding common subterms in the construction of the CST system.

Width-optimal preconditioner can be made even more appealing by reducing the computational costs for their computation—especially if the preconditioners have to be computed as many times as when applying extended systems. It might be possible to save computation time if one could "guess" an optimal or nearly optimal preconditioner. According to the authors opinion, this could work particularly well for extended systems with their simple constraints. Furthermore, computation time could be improved by utilizing the special structure of the optimization problem for the optimal preconditioner, sparse data structures or faster routines for vector and matrix operations. One option to achieve this goal is to feed the optimization problem to an appropriate external solver. Another approach could be the adjustment of a given algorithm to the given optimization problem. A third promising approach to reduce the computation time for preconditioning is to re-utilize preconditioner rows, as proposed in this work.

Up to now the optimization part of SONIC is rather a byproduct of the nonlinear solver. However, Beelitz showed that it works fairly well [Bee06]. Since the author has only changed details in the internal processing of optimization prob-

lems, there is a wide field for further improvements of the optimization part of SONIC. For example, the appropriate utilization of the subdivision strategies of the nonlinear solver could already provide considerable benefits for the optimizer. We could further think of specialized strategies as subdividing boxes near to points in which the monotonicity or curvature of a function changes.

To promote the common usage of SONIC, a web interface is in process of planning. Furthermore, the graphical output of the solution boxes and an appropriate representation of the calculation steps in the branch-and-bound algorithm are intended. Evaluated in combination with the yet available statistics and an analysis of the counters for the invocations and success of the implemented techniques, these graphical auxiliaries may provide further helpful insights for both, users and researchers.

Appendix A

List of symbols

$\exists$	existential quantifier ("exists")
$\mathcal{P}$	power set
$\mathbb{N}$	set of the natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$
m, n	if not stated otherwise $m, n \in \mathbb{N}$ (usually used for a function with m constraints and n variables)
$\mathbb{R}$	set of the real numbers
$\mathbb{R}_+$	set of the positive real numbers $\mathbb{R}_+ = \{r \in \mathbb{R} \mid r > 0\}$
$\mathbb{R}_{\geq 0}$	set of the nonnegative real numbers $\mathbb{R}_{\geq 0} = \{r \in \mathbb{R} \mid r \geq 0\}$
$\mathbb{R}^n$	set of all vectors of real numbers with length n
$\mathbb{R}^{m \times n}$	set of all $m \times n$ matrices with elements from $\mathbb{R}$
${}^*\mathbb{R}$	set of the extended real numbers ${}^*\mathbb{R} = \mathbb{R} \cup \{-\infty, +\infty\}$
$\mathbb{IR}$	set of all non-empty, bounded intervals in $\mathbb{R}$
$\mathbb{IR}^n$	set of all vectors of length n with elements from $\mathbb{IR}$
$\mathbb{IR}^{m \times n}$	set of all $m \times n$ matrices with elements from $\mathbb{IR}$
${}^*\mathbb{IR}$	set of all non-empty intervals in ${}^*\mathbb{R}$
${}^*\mathbb{IR}^n$	set of all vectors of length n with elements from ${}^*\mathbb{IR}$
${}^*\mathbb{IR}^{m \times n}$	set of all $m \times n$ matrices with elements from ${}^*\mathbb{IR}$
$\lfloor x \rfloor$	largest natural number not greater than $x \in \mathbb{R}$
$x = (x_1, \ldots, x_n)^T$	a vector of length n
$A = (a_{i,j})_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$	an $m \times n$ matrix
$\det A$	determinant of a square matrix A
$U, V \in \mathcal{P}({}^*\mathbb{R})$	arbitrary sets in the extended real numbers
$\inf(U)$	infimum of set U
$\sup(U)$	supremum of set U
$U \cup V$	union of sets U and V
$U \cap V$	intersection of sets U and V
$U \setminus V$	relative complement of V with respect to set U
$U \subseteq V$	U is a subset of V
$U \subset V$ or $U \subsetneq V$	U is a proper subset of V
$\square(U)$	the box operator $\square$ yields the smallest interval enclosing an arbitrary set of real numbers $U \in \mathcal{P}({}^*\mathbb{R})$, also interpreted componentwise for the smallest box comprising a set of vectors

$\mathrm{cl}(U)$	closure of set U
$\boldsymbol{x} = [\underline{x}, \overline{x}]$	interval or box (vector of intervals)
$\boldsymbol{x} = [x] = [x, x]$	point interval or vector of point intervals
$\inf(\boldsymbol{x}) = \underline{x}$	lower bound / infimum of $\boldsymbol{x}$
$\sup(\boldsymbol{x}) = \overline{x}$	upper bound / supremum of $\boldsymbol{x}$
$\boldsymbol{x}_i$	ith component of a vector $\boldsymbol{x}$ of intervals
$\widetilde{x}$	an inner point of $\boldsymbol{x}$
$\mathrm{mid}\,(\boldsymbol{x})$	midpoint of $\boldsymbol{x}$ (defined componentwise for vectors)
$\mathrm{width}\,(\boldsymbol{x})$	width (or diameter) of $\boldsymbol{x}$ (always nonnegative)
$\mathrm{rad}\,(\boldsymbol{x})$	radius of $\boldsymbol{x}$
$\mathrm{mag}(\boldsymbol{x}) = \lvert\boldsymbol{x}\rvert$	magnitude of $\boldsymbol{x}$
$\mathrm{mig}(\boldsymbol{x})$	mignitude of $\boldsymbol{x}$
$\mathrm{int}\,(\boldsymbol{x})$	interior of $\boldsymbol{x}$
$\mathrm{vol}(\boldsymbol{x})$	volume of a box $\boldsymbol{x}$
$\mathrm{RelVol}(\boldsymbol{x})$	relative volume of a box $\boldsymbol{x}$
$\mathrm{ThVol}(\boldsymbol{x})$	thresholded volume of a box $\boldsymbol{x}$
$\oslash$	relational division
$\partial\boldsymbol{x}$	topological boundary of $\boldsymbol{x}$
$\boldsymbol{x}^{i,+}$, $\boldsymbol{x}^{i,-}$	facets of $\boldsymbol{x}$ in direction i
$\boldsymbol{x}^{i,+,k}$, $\boldsymbol{x}^{i,-,k}$	subfacets of $\boldsymbol{x}$ in direction i
f_i	ith component of a function
Df	Jacobian matrix of a differentiable function f
$D^2 f$	Hessian matrix of a differentiable function f
$\boldsymbol{f}$ or $[f]$	interval-valued function, mostly function enclosure for some function f
$\lvert\lvert\cdot\rvert\rvert_\infty$	maximum norm
sign	sign function

Appendix B

Test problems

We conducted our tests with three different sets of test problems: test set $\mathcal{T}_A$, test set $\mathcal{T}_B$ and test set $\mathcal{T}_V$. Herein test set $\mathcal{T}_A$ is contained in test set $\mathcal{T}_B$. The problems have been chosen to represent a wide range of nonlinear problems and to cover a diversified spectrum of computation time. Each problem has a computation time of more than 10 seconds (in the initial version of SONIC). Test set $\mathcal{T}_V$ also is subset of test set $\mathcal{T}_B$ and is used for tests concerning verification methods.

Many problems can be found in the collection on the Coprin homepage [COP]. The chemistry systems are derived from modeling different singularities in process design, cf. [MMB$^+$07]. Since these problems contain sensitive data, we only display approximate values for these problems.

The *min-t-N* systems represent N-point spherical t-designs [HS96,SW09] formulated in polar coordinates. An N-point spherical t-design is a set of N points $z^1, \ldots, z^N$ on the unit sphere $S_d = \{x = (x_1, \ldots, x_d) \in \mathbb{R}^d \mid \|x\|_2 = 1\}$ in d dimensions. For all polynomials $p(x_1, \ldots, x_d)$ with degree up to t a spherical t-design fulfills the property

$$\frac{1}{|S_d|} \int_{S_d} p(x) d\mu = \frac{1}{N} \sum_{i=1}^{N} p(z^i)$$

where μ denotes the unit measure and $|S_d|$ is the surface measure of the sphere. Thus spherical t-designs provide quadrature rules that are exact for polynomials up to t-th degree over the sphere.

All *min-t-N* problems can be reformulated as nonlinear systems of equations and feed to SONIC. The details on the reformulation can be found in [BLUW09]. The same paper gives information about two computational proofs, based on SONIC, stating that there exists no spherical t-design for $d = 3$, $t = 3$ and $N = 7$ or $N = 9$. With the help of these computational proofs, all designs for $t = 3$ are found or an existence is ruled out.

Test set $\mathcal{T}_A$

Our test set $\mathcal{T}_A$ comprises five problems, *Brent7*, *Eco9*, *Trigonometric*, *7erSystem* and *min-04-07*. In the following, we list the startbox, precision vector ψ and the constraints for each test problem.

Brent7 The formulation of *BrentN* for general N can be found in [COP], we only consider $N = 7$.

$$\boldsymbol{x}_i = [-10^8, 10^8], \quad \psi_i = [10^{-8}] \quad \text{for } i = 1, \ldots, 7$$

$$\begin{aligned}
f_1(x) &= 3 \cdot x_1 \cdot (x_2 - 2 \cdot x_1) + x_2^2/4 \\
f_i(x) &= 3 \cdot x_i \cdot (x_{i+1} - 2 \cdot x_i + x_{i-1}) + (x_{i+1} - x_{i-1})^2/4 \\
&\qquad \text{for } i \in i = 2, \ldots, 6 \text{ and} \\
f_7(x) &= 3 \cdot x_7 \cdot (20 - 2 \cdot x_7 + x_6) + (20 - x_6)^2/4
\end{aligned}$$

Eco9 [COP] [Bee06]

$$\boldsymbol{x}_i = [-100, 100], \quad \psi_i = [10^{-6}] \quad \text{for } i = 1, \ldots, 8$$

$$\begin{aligned}
f_1(x) &= x_1 + x_2 \cdot (x_1 + x_3) + x_4 \cdot (x_3 + x_5) + x_6 \cdot (x_5 + x_7) - \left(x_8 \cdot \left(\frac{1}{8} - x_7\right)\right) \\
f_2(x) &= x_2 + x_3 \cdot (x_1 + x_5) + x_4 \cdot (x_2 + x_6) + x_5 \cdot x_7 - \left(x_8 \cdot \left(\frac{2}{8} - x_6\right)\right) \\
f_3(x) &= x_3 \cdot (1 + x_6) + x_4 \cdot (x_1 + x_7) + x_2 \cdot x_5 - \left(x_8 \cdot \left(\frac{3}{8} - x_5\right)\right) \\
f_4(x) &= x_4 + x_1 \cdot x_5 + x_2 \cdot x_6 + x_3 \cdot x_7 - \left(x_8 \cdot \left(\frac{4}{8} - x_4\right)\right) \\
f_5(x) &= x_5 + x_1 \cdot x_6 + x_2 \cdot x_7 - \left(x_8 \cdot \left(\frac{5}{8} - x_3\right)\right) \\
f_6(x) &= x_6 + x_1 \cdot x_7 - \left(x_8 \cdot \left(\frac{6}{8} - x_2\right)\right) \\
f_7(x) &= x_7 - \left(x_8 \cdot \left(\frac{7}{8} - x_1\right)\right) \\
f_8(x) &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + 1
\end{aligned}$$

Trigonometric [COP] $n = 10$

$$\boldsymbol{x}_i = [0, 2\pi - 0.001], \quad \psi_i = [10^{-6}] \quad \text{for } i = 1, \ldots, 10$$

$$f_i(x) = 5 - (l+1)(1 + \cos(x_i)) - \sin(x_i) - \sum_{j=5l+1}^{5l+5} \cos(x_j) \quad \text{with} \quad l = \lfloor (i-1)/5 \rfloor \text{ for } i = 1, \ldots, 10$$

7erSystem [Bee06]

$$\boldsymbol{x}_i = [-2, 2], \quad \psi_i = [10^{-6}] \quad \text{for } i = 1, \ldots, 7$$

$$f_1(x) = x_1 + x_2 + x_3 - 1$$

$$
\begin{aligned}
f_2(x) = & -0.6675958407 \\
& + 0.2055525566 \cdot x_1 + 0.6625749516 \cdot 10^{-1} \cdot x_2 + 0.2135733917 \cdot 10^{-1} \cdot x_3 \\
& - 0.3294350663 \cdot x_4 - 0.2878424112 \cdot x_5 \\
& - 0.2515010154 \cdot x_6 - 0.2197478839 \cdot x_7 \\
& + 0.2003078577 \cdot 10^{-2} \cdot x_1 \cdot x_2 + 0.6456692700 \cdot 10^{-3} \cdot x_1 \cdot x_3 \\
& + 0.2081240402 \cdot 10^{-3} \cdot x_2 \cdot x_3 + 0.2099295270 \cdot 10^{-1} \cdot x_1 \cdot x_4 \\
& + 0.6766836107 \cdot 10^{-2} \cdot x_2 \cdot x_4 + 0.2181211551 \cdot 10^{-2} \cdot x_3 \cdot x_4 \\
& + 0.1834249824 \cdot 10^{-1} \cdot x_1 \cdot x_5 + 0.5912492690 \cdot 10^{-2} \cdot x_2 \cdot x_5 \\
& + 0.1905823807 \cdot 10^{-2} \cdot x_3 \cdot x_5 + 0.5954522808 \cdot 10^{-1} \cdot x_4 \cdot x_5 \\
& + 0.1602667555 \cdot 10^{-1} \cdot x_1 \cdot x_6 + 0.5166013963 \cdot 10^{-2} \cdot x_2 \cdot x_6 \\
& + 0.1665205002 \cdot 10^{-2} \cdot x_3 \cdot x_6 + 0.5202737589 \cdot 10^{-1} \cdot x_4 \cdot x_6 \\
& + 0.4545868627 \cdot 10^{-1} \cdot x_5 \cdot x_6 + 0.1400323585 \cdot 10^{-1} \cdot x_1 \cdot x_7 \\
& + 0.4513781523 \cdot 10^{-2} \cdot x_2 \cdot x_7 + 0.1454965400 \cdot 10^{-2} \cdot x_3 \cdot x_7 \\
& + 0.4545868627 \cdot 10^{-1} \cdot x_4 \cdot x_7 + 0.3971932318 \cdot 10^{-1} \cdot x_5 \cdot x_7 \\
& + 0.3470458044 \cdot 10^{-1} \cdot x_6 \cdot x_7 \\
& + 0.3107104497 \cdot 10^{-2} \cdot x_1^2 + 0.3228346350 \cdot 10^{-3} \cdot x_2^2 \\
& + 0.3354319164 \cdot 10^{-4} \cdot x_3^2 + 0.3407469747 \cdot 10^{-1} \cdot x_4^2 \\
& + 0.2601368795 \cdot 10^{-1} \cdot x_5^2 + 0.1985966159 \cdot 10^{-1} \cdot x_6^2 \\
& + 0.1516148573 \cdot 10^{-1} \cdot x_7^2
\end{aligned}
$$

$$
\begin{aligned}
f_3(x) = & -0.5828529757 \\
& + 0.1228340113 \cdot x_1 + 0.2805508994 \cdot 10^{-1} \cdot x_2 + 0.6407737265 \cdot 10^{-2} \cdot x_3 \\
& - 0.3414023228 \cdot x_4 - 0.2406648259 \cdot x_5 \\
& - 0.1696519167 \cdot x_6 - 0.1195927686 \cdot x_7 \\
& + 0.1073433627 \cdot 10^{-2} \cdot x_1 \cdot x_2 + 0.2451705080 \cdot 10^{-3} \cdot x_1 \cdot x_3 \\
& + 0.5599654833 \cdot 10^{-4} \cdot x_2 \cdot x_3 + 0.1806580429 \cdot 10^{-1} \cdot x_1 \cdot x_4 \\
& + 0.4126200541 \cdot 10^{-2} \cdot x_2 \cdot x_4 + 0.9424175444 \cdot 10^{-3} \cdot x_3 \cdot x_4 \\
& + 0.1273513199 \cdot 10^{-1} \cdot x_1 \cdot x_5 + 0.2908683592 \cdot 10^{-2} \cdot x_2 \cdot x_5 \\
& + 0.6643386384 \cdot 10^{-3} \cdot x_3 \cdot x_5 + 0.4705176468 \cdot 10^{-1} \cdot x_4 \cdot x_5 \\
& + 0.8977379819 \cdot 10^{-2} \cdot x_1 \cdot x_6 + 0.2050419062 \cdot 10^{-2} \cdot x_2 \cdot x_6 \\
& + 0.4683124045 \cdot 10^{-3} \cdot x_3 \cdot x_6 + 0.3316821239 \cdot 10^{-1} \cdot x_4 \cdot x_6 \\
& + 0.2338127636 \cdot 10^{-1} \cdot x_5 \cdot x_6 + 0.6328426630 \cdot 10^{-2} \cdot x_1 \cdot x_7 \\
& + 0.1445402428 \cdot 10^{-2} \cdot x_2 \cdot x_7 + 0.3301275819 \cdot 10^{-3} \cdot x_3 \cdot x_7 \\
& + 0.2338127636 \cdot 10^{-1} \cdot x_4 \cdot x_7 + 0.1648216907 \cdot 10^{-1} \cdot x_5 \cdot x_7 \\
& + 0.1161877962 \cdot 10^{-1} \cdot x_6 \cdot x_7 \\
& + 0.2349915088 \cdot 10^{-2} \cdot x_1^2 + 0.1225852540 \cdot 10^{-3} \cdot x_2^2 \\
& + 0.6394760633 \cdot 10^{-5} \cdot x_3^2 + 0.3337334754 \cdot 10^{-1} \cdot x_4^2 \\
& + 0.1658410619 \cdot 10^{-1} \cdot x_5^2 + 0.8241084535 \cdot 10^{-2} \cdot x_6^2 \\
& + 0.4095214628 \cdot 10^{-2} \cdot x_7^2
\end{aligned}
$$

$$
\begin{aligned}
f_4(x) = & -0.5196091865 \\
& + 0.7952023194 \cdot 10^{-1} \cdot x_1 + 0.1369550717 \cdot 10^{-1} \cdot x_2 + 0.2358732012 \cdot 10^{-2} \cdot x_3 \\
& - 0.3231249836 \cdot x_4 - 0.1842648187 \cdot x_5 \\
& - 0.1050786078 \cdot x_6 - 0.5992198558 \cdot 10^{-1} \cdot x_7 \\
& + 0.5914814012 \cdot 10^{-3} \cdot x_1 \cdot x_2 + 0.1018688901 \cdot 10^{-3} \cdot x_1 \cdot x_3 \\
& + 0.1754454283 \cdot 10^{-4} \cdot x_2 \cdot x_3 + 0.1415543336 \cdot 10^{-1} \cdot x_1 \cdot x_4 \\
& + 0.2437943581 \cdot 10^{-2} \cdot x_2 \cdot x_4 + 0.4198789790 \cdot 10^{-3} \cdot x_3 \cdot x_4 \\
& + 0.8072258394 \cdot 10^{-2} \cdot x_1 \cdot x_5 + 0.1390258429 \cdot 10^{-2} \cdot x_2 \cdot x_5 \\
& + 0.2394396220 \cdot 10^{-3} \cdot x_3 \cdot x_5 + 0.3198484963 \cdot 10^{-1} \cdot x_4 \cdot x_5 \\
& + 0.4603275220 \cdot 10^{-2} \cdot x_1 \cdot x_6 + 0.7928069027 \cdot 10^{-3} \cdot x_2 \cdot x_6 \\
& + 0.1365425170 \cdot 10^{-3} \cdot x_3 \cdot x_6 + 0.1823963734 \cdot 10^{-1} \cdot x_4 \cdot x_6 \\
& + 0.1040131107 \cdot 10^{-1} \cdot x_5 \cdot x_6 + 0.2625057537 \cdot 10^{-2} \cdot x_1 \cdot x_7 \\
& + 0.4521049982 \cdot 10^{-3} \cdot x_2 \cdot x_7 + 0.7786455212 \cdot 10^{-4} \cdot x_3 \cdot x_7 \\
& + 0.1040131107 \cdot 10^{-1} \cdot x_4 \cdot x_7 + 0.5931437666 \cdot 10^{-2} \cdot x_5 \cdot x_7 \\
& + 0.3382453669 \cdot 10^{-2} \cdot x_6 \cdot x_7 \\
& + 0.1717159416 \cdot 10^{-2} \cdot x_1^2 + 0.5093444506 \cdot 10^{-4} \cdot x_2^2 \\
& + 0.1510819362 \cdot 10^{-5} \cdot x_3^2 + 0.2804415971 \cdot 10^{-1} \cdot x_4^2 \\
& + 0.9119818669 \cdot 10^{-2} \cdot x_5^2 + 0.2965718833 \cdot 10^{-2} \cdot x_6^2 \\
& + 0.9644367408 \cdot 10^{-3} \cdot x_7^2
\end{aligned}
$$

$$
\begin{aligned}
f_5(x) = & -0.9708221316 \\
& + 0.8863865849 \cdot x_1 + 0.8095263737 \cdot x_2 + 0.7393308528 \cdot x_3 \\
& - 0.9374687356 \cdot 10^{-2} \cdot x_4 - 0.2969634573 \cdot 10^{-2} \cdot x_5 \\
& - 0.9406958507 \cdot 10^{-3} \cdot x_6 - 0.2979857157 \cdot 10^{-3} \cdot x_7 \\
& + 0.4233432703 \cdot 10^{-3} \cdot x_1 \cdot x_2 + 0.3866343967 \cdot 10^{-3} \cdot x_1 \cdot x_3 \\
& + 0.3531086170 \cdot 10^{-3} \cdot x_2 \cdot x_3 + 0.2009129356 \cdot 10^{-3} \cdot x_1 \cdot x_4 \\
& + 0.1834914054 \cdot 10^{-3} \cdot x_2 \cdot x_4 + 0.1675805281 \cdot 10^{-3} \cdot x_3 \cdot x_4 \\
& + 0.6364350907 \cdot 10^{-4} \cdot x_1 \cdot x_5 + 0.5812486333 \cdot 10^{-4} \cdot x_2 \cdot x_5 \\
& + 0.5308474951 \cdot 10^{-4} \cdot x_3 \cdot x_5 + 0.2648440225 \cdot 10^{-4} \cdot x_4 \cdot x_5 \\
& + 0.2016045525 \cdot 10^{-4} \cdot x_1 \cdot x_6 + 0.1841230509 \cdot 10^{-4} \cdot x_2 \cdot x_6 \\
& + 0.1681574025 \cdot 10^{-4} \cdot x_3 \cdot x_6 + 0.8389506080 \cdot 10^{-5} \cdot x_4 \cdot x_6 \\
& + 0.2657557139 \cdot 10^{-5} \cdot x_5 \cdot x_6 + 0.6386259364 \cdot 10^{-5} \cdot x_1 \cdot x_7 \\
& + 0.5832495068 \cdot 10^{-5} \cdot x_2 \cdot x_7 + 0.5326748692 \cdot 10^{-5} \cdot x_3 \cdot x_7 \\
& + 0.2657557139 \cdot 10^{-5} \cdot x_4 \cdot x_7 + 0.8418385875 \cdot 10^{-6} \cdot x_5 \cdot x_7 \\
& + 0.2666705437 \cdot 10^{-6} \cdot x_6 \cdot x_7 \\
& + 0.2317687279 \cdot 10^{-3} \cdot x_1^2 + 0.1933171983 \cdot 10^{-3} \cdot x_2^2 \\
& + 0.1612449597 \cdot 10^{-3} \cdot x_3^2 + 0.4180362680 \cdot 10^{-4} \cdot x_4^2 \\
& + 0.4194753040 \cdot 10^{-5} \cdot x_5^2 + 0.4209192938 \cdot 10^{-6} \cdot x_6^2 \\
& + 0.4223682540 \cdot 10^{-7} \cdot x_7^2
\end{aligned}
$$

$$
\begin{aligned}
f_6(x) = & -0.9802721137 \\
& + 0.9219460300 \cdot x_1 + 0.8672034774 \cdot x_2 + 0.8157113830 \cdot x_3 \\
& - 0.4450172272 \cdot 10^{-2} \cdot x_4 - 0.9941956844 \cdot 10^{-3} \cdot x_5 \\
& - 0.2221093924 \cdot 10^{-3} \cdot x_6 - 0.4962059574 \cdot 10^{-4} \cdot x_7 \\
& + 0.2128632096 \cdot 10^{-3} \cdot x_1 \cdot x_2 + 0.2002239931 \cdot 10^{-3} \cdot x_1 \cdot x_3 \\
& + 0.1883352576 \cdot 10^{-3} \cdot x_2 \cdot x_3 + 0.6717220595 \cdot 10^{-4} \cdot x_1 \cdot x_4 \\
& + 0.6318370998 \cdot 10^{-4} \cdot x_2 \cdot x_4 + 0.5943203966 \cdot 10^{-4} \cdot x_3 \cdot x_4 \\
& + 0.1500668136 \cdot 10^{-4} \cdot x_1 \cdot x_5 + 0.1411562698 \cdot 10^{-4} \cdot x_2 \cdot x_5 \\
& + 0.1327748090 \cdot 10^{-4} \cdot x_3 \cdot x_5 + 0.4276502662 \cdot 10^{-5} \cdot x_4 \cdot x_5 \\
& + 0.3352584336 \cdot 10^{-5} \cdot x_1 \cdot x_6 + 0.3153517342 \cdot 10^{-5} \cdot x_2 \cdot x_6 \\
& + 0.2966270385 \cdot 10^{-5} \cdot x_3 \cdot x_6 + 0.9553968318 \cdot 10^{-6} \cdot x_4 \cdot x_6 \\
& + 0.2134414915 \cdot 10^{-6} \cdot x_5 \cdot x_6 + 0.7489878308 \cdot 10^{-6} \cdot x_1 \cdot x_7 \\
& + 0.7045150477 \cdot 10^{-6} \cdot x_2 \cdot x_7 + 0.6626829330 \cdot 10^{-6} \cdot x_3 \cdot x_7 \\
& + 0.2134414915 \cdot 10^{-6} \cdot x_4 \cdot x_7 + 0.4768413372 \cdot 10^{-7} \cdot x_5 \cdot x_7 \\
& + 0.1065292692 \cdot 10^{-7} \cdot x_6 \cdot x_7 \\
& + 0.1131501407 \cdot 10^{-3} \cdot x_1^2 + 0.1001119965 \cdot 10^{-3} \cdot x_2^2 \\
& + 0.8857622082 \cdot 10^{-4} \cdot x_3^2 + 0.9571140702 \cdot 10^{-5} \cdot x_4^2 \\
& + 0.4776984159 \cdot 10^{-6} \cdot x_5^2 + 0.2384206686 \cdot 10^{-7} \cdot x_6^2 \\
& + 0.1189964491 \cdot 10^{-8} \cdot x_7^2
\end{aligned}
$$

$$
\begin{aligned}
f_7(x) = & -0.7736815316 \\
& + 0.3617367006 \cdot x_1 + 0.1724108153 \cdot x_2 + 0.8217438040 \cdot 10^{-1} \cdot x_3 \\
& - 0.2477761981 \cdot x_4 - 0.2472344192 \cdot x_5 - 0.2466938251 \cdot x_6 \\
& - 0.2461544129 \cdot x_7 \\
& + 0.3187193360 \cdot 10^{-2} \cdot x_1 \cdot x_2 + 0.1519078946 \cdot 10^{-2} \cdot x_1 \cdot x_3 \\
& + 0.7240228578 \cdot 10^{-3} \cdot x_2 \cdot x_3 + 0.1746280745 \cdot 10^{-1} \cdot x_1 \cdot x_4 \\
& + 0.8323116969 \cdot 10^{-2} \cdot x_2 \cdot x_4 + 0.3966960997 \cdot 10^{-2} \cdot x_3 \cdot x_4 \\
& + 0.1742462389 \cdot 10^{-1} \cdot x_1 \cdot x_5 + 0.8304917930 \cdot 10^{-2} \cdot x_2 \cdot x_5 \\
& + 0.3958286979 \cdot 10^{-2} \cdot x_3 \cdot x_5 + 0.4371338635 \cdot 10^{-1} \cdot x_4 \cdot x_5 \\
& + 0.1738652381 \cdot 10^{-1} \cdot x_1 \cdot x_6 + 0.8286758688 \cdot 10^{-2} \cdot x_2 \cdot x_6 \\
& + 0.3949631927 \cdot 10^{-2} \cdot x_3 \cdot x_6 + 0.4361780417 \cdot 10^{-1} \cdot x_4 \cdot x_6 \\
& + 0.4352243100 \cdot 10^{-1} \cdot x_5 \cdot x_6 + 0.1734850704 \cdot 10^{-1} \cdot x_1 \cdot x_7 \\
& + 0.8268639147 \cdot 10^{-2} \cdot x_2 \cdot x_7 + 0.3940995796 \cdot 10^{-2} \cdot x_3 \cdot x_7 \\
& + 0.4352243100 \cdot 10^{-1} \cdot x_4 \cdot x_7 + 0.4342726636 \cdot 10^{-1} \cdot x_5 \cdot x_7 \\
& + 0.4333230981 \cdot 10^{-1} \cdot x_6 \cdot x_7 \\
& + 0.3343539697 \cdot 10^{-2} \cdot x_1^2 + 0.7595394732 \cdot 10^{-3} \cdot x_2^2 \\
& + 0.1725417563 \cdot 10^{-3} \cdot x_3^2 + 0.2190458898 \cdot 10^{-1} \cdot x_4^2 \\
& + 0.2180890209 \cdot 10^{-1} \cdot x_5^2 + 0.2171363318 \cdot 10^{-1} \cdot x_6^2 \\
& + 0.2161878045 \cdot 10^{-1} \cdot x_7^2
\end{aligned}
$$

min-04-07 [BLUW09]

$$
\begin{aligned}
&\varphi_3 = [0, \pi], && \psi_{\varphi_3} = [10^{-8}] && \\
&\varphi_i = [0, 2\pi], && \psi_{\varphi_i} = [10^{-8}] && \text{for } i = 4, \ldots, 7 \\
&\theta_i = [0, \pi], && \psi_{\theta_i} = [10^{-8}] && \text{for } i = 2, \ldots, 7
\end{aligned}
$$

$$
\begin{aligned}
f_1(x) = {} & \sin(\theta_2) \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3)) + (\sin(\theta_4) \cdot \cos(\varphi_4)) + (\sin(\theta_5) \cdot \cos(\varphi_5)) \\
& + (\sin(\theta_6) \cdot \cos(\varphi_6)) + (\sin(\theta_7) \cdot \cos(\varphi_7))
\end{aligned}
$$

$$
\begin{aligned}
f_2(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3)) + (\sin(\theta_4) \cdot \sin(\varphi_4)) + (\sin(\theta_5) \cdot \sin(\varphi_5)) \\
& + (\sin(\theta_6) \cdot \sin(\varphi_6)) + (\sin(\theta_7) \cdot \sin(\varphi_7))
\end{aligned}
$$

$$
f_3(x) = 1 + \cos(\theta_2) + \cos(\theta_3) + \cos(\theta_4) + \cos(\theta_5) + \cos(\theta_6) + \cos(\theta_7)
$$

$$
\begin{aligned}
f_4(x) = {} & \sin(\theta_2)^2 \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3))^2 + (\sin(\theta_4) \cdot \cos(\varphi_4))^2 + (\sin(\theta_5) \cdot \cos(\varphi_5))^2 \\
& + (\sin(\theta_6) \cdot \cos(\varphi_6))^2 + (\sin(\theta_7) \cdot \cos(\varphi_7))^2 - 7/3
\end{aligned}
$$

$$
\begin{aligned}
f_5(x) = {} & (\sin(\theta_3) \cdot \cos(\varphi_3)) \cdot (\sin(\theta_3) \cdot \sin(\varphi_3)) \\
& + (\sin(\theta_4) \cdot \cos(\varphi_4)) \cdot (\sin(\theta_4) \cdot \sin(\varphi_4)) \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5)) \cdot (\sin(\theta_5) \cdot \sin(\varphi_5)) \\
& + (\sin(\theta_6) \cdot \cos(\varphi_6)) \cdot (\sin(\theta_6) \cdot \sin(\varphi_6)) \\
& + (\sin(\theta_7) \cdot \cos(\varphi_7)) \cdot (\sin(\theta_7) \cdot \sin(\varphi_7))
\end{aligned}
$$

$$
\begin{aligned}
f_6(x) = {} & \sin(\theta_2) \cdot \cos(\theta_2) \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3)) \cdot \cos(\theta_3) + (\sin(\theta_4) \cdot \cos(\varphi_4)) \cdot \cos(\theta_4) \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5)) \cdot \cos(\theta_5) + (\sin(\theta_6) \cdot \cos(\varphi_6)) \cdot \cos(\theta_6) \\
& + (\sin(\theta_7) \cdot \cos(\varphi_7)) \cdot \cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_7(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3))^2 + (\sin(\theta_4) \cdot \sin(\varphi_4))^2 + (\sin(\theta_5) \cdot \sin(\varphi_5))^2 \\
& + (\sin(\theta_6) \cdot \sin(\varphi_6))^2 + (\sin(\theta_7) \cdot \sin(\varphi_7))^2 - 7/3
\end{aligned}
$$

$$
\begin{aligned}
f_8(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3)) \cdot \cos(\theta_3) + (\sin(\theta_4) \cdot \sin(\varphi_4)) \cdot \cos(\theta_4) \\
& + (\sin(\theta_5) \cdot \sin(\varphi_5)) \cdot \cos(\theta_5) + (\sin(\theta_6) \cdot \sin(\varphi_6)) \cdot \cos(\theta_6) \\
& + (\sin(\theta_7) \cdot \sin(\varphi_7)) \cdot \cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_9(x) = {} & 1^2 + \cos(\theta_2)^2 + \cos(\theta_3)^2 + \cos(\theta_4)^2 \\
& + \cos(\theta_5)^2 + \cos(\theta_6)^2 + \cos(\theta_7)^2 - 7/3
\end{aligned}
$$

$$
\begin{aligned}
f_{10}(x) = {} & \sin(\theta_2)^3 + (\sin(\theta_3) \cdot \cos(\varphi_3))^3 + (\sin(\theta_4) \cdot \cos(\varphi_4))^3 \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5))^3 + (\sin(\theta_6) \cdot \cos(\varphi_6))^3 + (\sin(\theta_7) \cdot \cos(\varphi_7))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{11}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot(\sin(\theta_3)\cdot\sin(\varphi_3)) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot(\sin(\theta_4)\cdot\sin(\varphi_4)) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot(\sin(\theta_5)\cdot\sin(\varphi_5)) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot(\sin(\theta_6)\cdot\sin(\varphi_6)) \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^2\cdot(\sin(\theta_7)\cdot\sin(\varphi_7))
\end{aligned}
$$

$$
\begin{aligned}
f_{12}(x) = {} & \sin(\theta_2)^2\cdot\cos(\theta_2) + (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot\cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot\cos(\theta_4) + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot\cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot\cos(\theta_6) + (\sin(\theta_7)\cdot\cos(\varphi_7))^2\cdot\cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{13}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^2 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^2 \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot(\sin(\theta_7)\cdot\sin(\varphi_7))^2
\end{aligned}
$$

$$
\begin{aligned}
f_{14}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6) \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot(\sin(\theta_7)\cdot\sin(\varphi_7))\cdot\cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{15}(x) = {} & \sin(\theta_2)\cdot\cos(\theta_2)^2 + (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot\cos(\theta_3)^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot\cos(\theta_4)^2 + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot\cos(\theta_5)^2 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot\cos(\theta_6)^2 + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot\cos(\theta_7)^2
\end{aligned}
$$

$$
\begin{aligned}
f_{16}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^3 + (\sin(\theta_4)\cdot\sin(\varphi_4))^3 + (\sin(\theta_5)\cdot\sin(\varphi_5))^3 \\
& + (\sin(\theta_6)\cdot\sin(\varphi_6))^3 + (\sin(\theta_7)\cdot\sin(\varphi_7))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{17}(x) = {} & \cos(\theta_2) \\
& + (\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6) \\
& + (\sin(\theta_7)\cdot\sin(\varphi_7))^2\cdot\cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{18}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^2 + (\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^2 \\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^2 + (\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^2 \\
& + (\sin(\theta_7)\cdot\sin(\varphi_7))\cdot\cos(\theta_7)^2
\end{aligned}
$$

$$
f_{19}(x) = 1^3 + \cos(\theta_2)^3 + \cos(\theta_3)^3 + \cos(\theta_4)^3 + \cos(\theta_5)^3 + \cos(\theta_6)^3 + \cos(\theta_7)^3
$$

$$
\begin{aligned}
f_{20}(x) = {} & \sin(\theta_2)^4 \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^4 + (\sin(\theta_4)\cdot\cos(\varphi_4))^4 + (\sin(\theta_5)\cdot\cos(\varphi_5))^4 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^4 + (\sin(\theta_7)\cdot\cos(\varphi_7))^4 - 7/5
\end{aligned}
$$

$$
\begin{aligned}
f_{21}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^3 \cdot (\sin(\theta_3)\cdot\sin(\varphi_3)) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^3 \cdot (\sin(\theta_4)\cdot\sin(\varphi_4)) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^3 \cdot (\sin(\theta_5)\cdot\sin(\varphi_5)) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^3 \cdot (\sin(\theta_6)\cdot\sin(\varphi_6)) \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^3 \cdot (\sin(\theta_7)\cdot\sin(\varphi_7))
\end{aligned}
$$

$$
\begin{aligned}
f_{22}(x) = {} & \sin(\theta_2)^3 \cdot \cos(\theta_2) \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^3 \cdot \cos(\theta_3) + (\sin(\theta_4)\cdot\cos(\varphi_4))^3 \cdot \cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^3 \cdot \cos(\theta_5) + (\sin(\theta_6)\cdot\cos(\varphi_6))^3 \cdot \cos(\theta_6) \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^3 \cdot \cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{23}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2 \cdot (\sin(\theta_3)\cdot\sin(\varphi_3))^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2 \cdot (\sin(\theta_4)\cdot\sin(\varphi_4))^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2 \cdot (\sin(\theta_5)\cdot\sin(\varphi_5))^2 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2 \cdot (\sin(\theta_6)\cdot\sin(\varphi_6))^2 \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^2 \cdot (\sin(\theta_7)\cdot\sin(\varphi_7))^2 - 7/15
\end{aligned}
$$

$$
\begin{aligned}
f_{24}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2 \cdot (\sin(\theta_3)\cdot\sin(\varphi_3)) \cdot \cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2 \cdot (\sin(\theta_4)\cdot\sin(\varphi_4)) \cdot \cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2 \cdot (\sin(\theta_5)\cdot\sin(\varphi_5)) \cdot \cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2 \cdot (\sin(\theta_6)\cdot\sin(\varphi_6)) \cdot \cos(\theta_6) \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^2 \cdot (\sin(\theta_7)\cdot\sin(\varphi_7)) \cdot \cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{25}(x) = {} & \sin(\theta_2)^2 \cdot \cos(\theta_2)^2 \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^2 \cdot \cos(\theta_3)^2 + (\sin(\theta_4)\cdot\cos(\varphi_4))^2 \cdot \cos(\theta_4)^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2 \cdot \cos(\theta_5)^2 + (\sin(\theta_6)\cdot\cos(\varphi_6))^2 \cdot \cos(\theta_6)^2 \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))^2 \cdot \cos(\theta_7)^2 - 7/15
\end{aligned}
$$

$$
\begin{aligned}
f_{26}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3)) \cdot (\sin(\theta_3)\cdot\sin(\varphi_3))^3 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4)) \cdot (\sin(\theta_4)\cdot\sin(\varphi_4))^3 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5)) \cdot (\sin(\theta_5)\cdot\sin(\varphi_5))^3 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6)) \cdot (\sin(\theta_6)\cdot\sin(\varphi_6))^3 \\
& + (\sin(\theta_7)\cdot\cos(\varphi_7)) \cdot (\sin(\theta_7)\cdot\sin(\varphi_7))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{27}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3)\\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4)\\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5)\\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6)\\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot(\sin(\theta_7)\cdot\sin(\varphi_7))^2\cdot\cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{28}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^2\\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^2\\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^2\\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^2\\
& + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot(\sin(\theta_7)\cdot\sin(\varphi_7))\cdot\cos(\theta_7)^2
\end{aligned}
$$

$$
\begin{aligned}
f_{29}(x) = {} & \sin(\theta_2)\cdot\cos(\theta_2)^3 + (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot\cos(\theta_3)^3\\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot\cos(\theta_4)^3 + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot\cos(\theta_5)^3\\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot\cos(\theta_6)^3 + (\sin(\theta_7)\cdot\cos(\varphi_7))\cdot\cos(\theta_7)^3
\end{aligned}
$$

$$
\begin{aligned}
f_{30}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^4 + (\sin(\theta_4)\cdot\sin(\varphi_4))^4 + (\sin(\theta_5)\cdot\sin(\varphi_5))^4\\
& + (\sin(\theta_6)\cdot\sin(\varphi_6))^4 + (\sin(\theta_7)\cdot\sin(\varphi_7))^4 - 7/5
\end{aligned}
$$

$$
\begin{aligned}
f_{31}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^3\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\sin(\varphi_4))^3\cdot\cos(\theta_4)\\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))^3\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\sin(\varphi_6))^3\cdot\cos(\theta_6)\\
& + (\sin(\theta_7)\cdot\sin(\varphi_7))^3\cdot\cos(\theta_7)
\end{aligned}
$$

$$
\begin{aligned}
f_{32}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3)^2 + (\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4)^2\\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5)^2 + (\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6)^2\\
& + (\sin(\theta_7)\cdot\sin(\varphi_7))^2\cdot\cos(\theta_7)^2 - 7/15
\end{aligned}
$$

$$
\begin{aligned}
f_{32}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^3 + (\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^3\\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^3 + (\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^3\\
& + (\sin(\theta_7)\cdot\sin(\varphi_7))\cdot\cos(\theta_7)^3
\end{aligned}
$$

$$
\begin{aligned}
f_{33}(x) = {} & 1 + \cos(\theta_2)^4 + \cos(\theta_3)^4 + \cos(\theta_4)^4 + \cos(\theta_5)^4\\
& + \cos(\theta_6)^4 + \cos(\theta_7)^4 - 7/5
\end{aligned}
$$

$$
\begin{array}{lll}
\varphi_4 \geq \varphi_3 & \varphi_5 \geq \varphi_3 & \varphi_5 \geq \varphi_4\\
\varphi_6 \geq \varphi_3 & \varphi_6 \geq \varphi_4 & \varphi_6 \geq \varphi_5\\
\varphi_7 \geq \varphi_3 & \varphi_7 \geq \varphi_4 & \varphi_7 \geq \varphi_5\\
\varphi_7 \geq \varphi_6 & &
\end{array}
$$

Test set $\mathcal{T}_B$

Test set $\mathcal{T}_B$ comprises 14 test problems, including all problems from test set $\mathcal{T}_A$. The other nine are given as follows.

min-04-06 [BLUW09]

$$
\begin{aligned}
&\varphi_3 = [0, \pi], && \psi_{\varphi_3} = [10^{-8}] && \\
&\varphi_i = [0, 2\pi], && \psi_{\varphi_i} = [10^{-8}] && \text{for } i = 4, \ldots, 6 \\
&\theta_i = [0, \pi], && \psi_{\theta_i} = [10^{-8}] && \text{for } i = 2, \ldots, 6
\end{aligned}
$$

$$
\begin{aligned}
f_1(x) = {} & \sin(\theta_2) \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3)) + (\sin(\theta_4) \cdot \cos(\varphi_4)) \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5)) + (\sin(\theta_6) \cdot \cos(\varphi_6))
\end{aligned}
$$

$$
\begin{aligned}
f_2(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3)) + (\sin(\theta_4) \cdot \sin(\varphi_4)) \\
& + (\sin(\theta_5) \cdot \sin(\varphi_5)) + (\sin(\theta_6) \cdot \sin(\varphi_6))
\end{aligned}
$$

$$
f_3(x) = 1 + \cos(\theta_2) + \cos(\theta_3) + \cos(\theta_4) + \cos(\theta_5) + \cos(\theta_6)
$$

$$
\begin{aligned}
f_4(x) = {} & \sin(\theta_2)^2 \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3))^2 + (\sin(\theta_4) \cdot \cos(\varphi_4))^2 \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5))^2 + (\sin(\theta_6) \cdot \cos(\varphi_6))^2 - 2
\end{aligned}
$$

$$
\begin{aligned}
f_5(x) = {} & (\sin(\theta_3) \cdot \cos(\varphi_3)) \cdot (\sin(\theta_3) \cdot \sin(\varphi_3)) \\
& + (\sin(\theta_4) \cdot \cos(\varphi_4)) \cdot (\sin(\theta_4) \cdot \sin(\varphi_4)) \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5)) \cdot (\sin(\theta_5) \cdot \sin(\varphi_5)) \\
& + (\sin(\theta_6) \cdot \cos(\varphi_6)) \cdot (\sin(\theta_6) \cdot \sin(\varphi_6))
\end{aligned}
$$

$$
\begin{aligned}
f_6(x) = {} & \sin(\theta_2) \cdot \cos(\theta_2) \\
& + (\sin(\theta_3) \cdot \cos(\varphi_3)) \cdot \cos(\theta_3) + (\sin(\theta_4) \cdot \cos(\varphi_4)) \cdot \cos(\theta_4) \\
& + (\sin(\theta_5) \cdot \cos(\varphi_5)) \cdot \cos(\theta_5) + (\sin(\theta_6) \cdot \cos(\varphi_6)) \cdot \cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_7(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3))^2 + (\sin(\theta_4) \cdot \sin(\varphi_4))^2 \\
& + (\sin(\theta_5) \cdot \sin(\varphi_5))^2 + (\sin(\theta_6) \cdot \sin(\varphi_6))^2 - 2
\end{aligned}
$$

$$
\begin{aligned}
f_8(x) = {} & (\sin(\theta_3) \cdot \sin(\varphi_3)) \cdot \cos(\theta_3) + (\sin(\theta_4) \cdot \sin(\varphi_4)) \cdot \cos(\theta_4) \\
& + (\sin(\theta_5) \cdot \sin(\varphi_5)) \cdot \cos(\theta_5) + (\sin(\theta_6) \cdot \sin(\varphi_6)) \cdot \cos(\theta_6)
\end{aligned}
$$

$$
f_9(x) = 1 + \cos(\theta_2)^2 + \cos(\theta_3)^2 + \cos(\theta_4)^2 + \cos(\theta_5)^2 + \cos(\theta_6)^2 - 2
$$

$$
\begin{aligned}
f_{10}(x) = {} & \sin(\theta_2)^3 + (\sin(\theta_3)\cdot\cos(\varphi_3))^3 + (\sin(\theta_4)\cdot\cos(\varphi_4))^3 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^3 + (\sin(\theta_6)\cdot\cos(\varphi_6))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{11}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot(\sin(\theta_3)\cdot\sin(\varphi_3)) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot(\sin(\theta_4)\cdot\sin(\varphi_4)) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot(\sin(\theta_5)\cdot\sin(\varphi_5)) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))
\end{aligned}
$$

$$
\begin{aligned}
f_{12}(x) = {} & \sin(\theta_2)^2\cdot\cos(\theta_2) \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{13}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^2 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^2
\end{aligned}
$$

$$
\begin{aligned}
f_{14}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{15}(x) = {} & \sin(\theta_2)\cdot\cos(\theta_2)^2 \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot\cos(\theta_3)^2 + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot\cos(\theta_4)^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot\cos(\theta_5)^2 + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot\cos(\theta_6)^2
\end{aligned}
$$

$$
\begin{aligned}
f_{16}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^3 + (\sin(\theta_4)\cdot\sin(\varphi_4))^3 \\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))^3 + (\sin(\theta_6)\cdot\sin(\varphi_6))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{17}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{18}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^2 \\
& + (\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^2 \\
& + (\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^2 \\
& + (\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^2
\end{aligned}
$$

$$
f_{19}(x) = 1^3 + \cos(\theta_2)^3 + \cos(\theta_3)^3 + \cos(\theta_4)^3 + \cos(\theta_5)^3 + \cos(\theta_6)^3
$$

$$
\begin{aligned}
f_{20}(x) = {} & \sin(\theta_2)^4 \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^4 + (\sin(\theta_4)\cdot\cos(\varphi_4))^4 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^4 + (\sin(\theta_6)\cdot\cos(\varphi_6))^4 - 6/5
\end{aligned}
$$

$$
\begin{aligned}
f_{21}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^3\cdot(\sin(\theta_3)\cdot\sin(\varphi_3)) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^3\cdot(\sin(\theta_4)\cdot\sin(\varphi_4)) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^3\cdot(\sin(\theta_5)\cdot\sin(\varphi_5)) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^3\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))
\end{aligned}
$$

$$
\begin{aligned}
f_{22}(x) = {} & \sin(\theta_2)^3\cdot\cos(\theta_2) \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^3\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\cos(\varphi_4))^3\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^3\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\cos(\varphi_6))^3\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{23}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^2 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^2 - 2/5
\end{aligned}
$$

$$
\begin{aligned}
f_{24}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{25}(x) = {} & \sin(\theta_2)^2\cdot\cos(\theta_2)^2 \\
& + (\sin(\theta_3)\cdot\cos(\varphi_3))^2\cdot\cos(\theta_3)^2 + (\sin(\theta_4)\cdot\cos(\varphi_4))^2\cdot\cos(\theta_4)^2 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))^2\cdot\cos(\theta_5)^2 + (\sin(\theta_6)\cdot\cos(\varphi_6))^2\cdot\cos(\theta_6)^2 - 2/5
\end{aligned}
$$

$$
\begin{aligned}
f_{26}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^3 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^3 \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^3 \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^3
\end{aligned}
$$

$$
\begin{aligned}
f_{27}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3) \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4) \\
& + (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5) \\
& + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{28}(x) = {} & (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot(\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^2 \\
& + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot(\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^2
\end{aligned}
$$

$$
\begin{aligned}
&+ (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot(\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^2\\
&+ (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot(\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^2
\end{aligned}
$$

$$
\begin{aligned}
f_{29}(x) = {} & \sin(\theta_2)\cdot\cos(\theta_2)^3\\
&+ (\sin(\theta_3)\cdot\cos(\varphi_3))\cdot\cos(\theta_3)^3 + (\sin(\theta_4)\cdot\cos(\varphi_4))\cdot\cos(\theta_4)^3\\
&+ (\sin(\theta_5)\cdot\cos(\varphi_5))\cdot\cos(\theta_5)^3 + (\sin(\theta_6)\cdot\cos(\varphi_6))\cdot\cos(\theta_6)^3
\end{aligned}
$$

$$
\begin{aligned}
f_{30}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^4 + (\sin(\theta_4)\cdot\sin(\varphi_4))^4\\
&+ (\sin(\theta_5)\cdot\sin(\varphi_5))^4 + (\sin(\theta_6)\cdot\sin(\varphi_6))^4 - 6/5
\end{aligned}
$$

$$
\begin{aligned}
f_{31}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^3\cdot\cos(\theta_3) + (\sin(\theta_4)\cdot\sin(\varphi_4))^3\cdot\cos(\theta_4)\\
&+ (\sin(\theta_5)\cdot\sin(\varphi_5))^3\cdot\cos(\theta_5) + (\sin(\theta_6)\cdot\sin(\varphi_6))^3\cdot\cos(\theta_6)
\end{aligned}
$$

$$
\begin{aligned}
f_{32}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))^2\cdot\cos(\theta_3)^2 + (\sin(\theta_4)\cdot\sin(\varphi_4))^2\cdot\cos(\theta_4)^2\\
&+ (\sin(\theta_5)\cdot\sin(\varphi_5))^2\cdot\cos(\theta_5)^2 + (\sin(\theta_6)\cdot\sin(\varphi_6))^2\cdot\cos(\theta_6)^2 - 2/5
\end{aligned}
$$

$$
\begin{aligned}
f_{32}(x) = {} & (\sin(\theta_3)\cdot\sin(\varphi_3))\cdot\cos(\theta_3)^3 + (\sin(\theta_4)\cdot\sin(\varphi_4))\cdot\cos(\theta_4)^3\\
&+ (\sin(\theta_5)\cdot\sin(\varphi_5))\cdot\cos(\theta_5)^3 + (\sin(\theta_6)\cdot\sin(\varphi_6))\cdot\cos(\theta_6)^3
\end{aligned}
$$

$$
f_{33}(x) = 1 + \cos(\theta_2)^4 + \cos(\theta_3)^4 + \cos(\theta_4)^4 + \cos(\theta_5)^4 + \cos(\theta_6)^4 - 6/5
$$

$$
\begin{array}{lll}
\varphi_4 \geq \varphi_3 & \varphi_5 \geq \varphi_3 & \varphi_5 \geq \varphi_4\\
\varphi_6 \geq \varphi_3 & \varphi_6 \geq \varphi_4 & \varphi_6 \geq \varphi_5
\end{array}
$$

G7_gradientsystem Is equal to problem "Griewank" in [Bee06].

$$
\boldsymbol{x}_i = [-3, 3], \quad \psi_i = [10^{-2}] \quad \text{for } i = 1, \ldots, 7
$$

For shorter notation we introduce $\breve{x}_i := \frac{x_i}{\sqrt{i}}$ for $i = 1, \ldots, 7$. (These were *not* part of the system as used for calculations.)

$$
f_1(x) = \frac{x_1}{2000} + \frac{1}{\sqrt{1}}\sin(\breve{x}_1)\cos(\breve{x}_2)\cos(\breve{x}_3)\cos(\breve{x}_4)\cos(\breve{x}_5)\cos(\breve{x}_6)\cos(\breve{x}_7)
$$

$$
f_2(x) = \frac{x_2}{2000} + \frac{1}{\sqrt{2}}\cos(\breve{x}_1)\sin(\breve{x}_2)\cos(\breve{x}_3)\cos(\breve{x}_4)\cos(\breve{x}_5)\cos(\breve{x}_6)\cos(\breve{x}_7)
$$

$$
f_3(x) = \frac{x_3}{2000} + \frac{1}{\sqrt{3}}\cos(\breve{x}_1)\cos(\breve{x}_2)\sin(\breve{x}_3)\cos(\breve{x}_4)\cos(\breve{x}_5)\cos(\breve{x}_6)\cos(\breve{x}_7)
$$

$$
f_4(x) = \frac{x_4}{2000} + \frac{1}{\sqrt{4}}\cos(\breve{x}_1)\cos(\breve{x}_2)\cos(\breve{x}_3)\sin(\breve{x}_4)\cos(\breve{x}_5)\cos(\breve{x}_6)\cos(\breve{x}_7)
$$

$$
f_5(x) = \frac{x_5}{2000} + \frac{1}{\sqrt{5}}\cos(\breve{x}_1)\cos(\breve{x}_2)\cos(\breve{x}_3)\cos(\breve{x}_4)\sin(\breve{x}_5)\cos(\breve{x}_6)\cos(\breve{x}_7)
$$

$$f_6(x) = \frac{x_6}{2000} + \frac{1}{\sqrt{6}} \cos(\breve{x}_1) \cos(\breve{x}_2) \cos(\breve{x}_3) \cos(\breve{x}_4) \cos(\breve{x}_5) \sin(\breve{x}_6) \cos(\breve{x}_7)$$

$$f_7(x) = \frac{x_7}{2000} + \frac{1}{\sqrt{7}} \cos(\breve{x}_1) \cos(\breve{x}_2) \cos(\breve{x}_3) \cos(\breve{x}_4) \cos(\breve{x}_5) \cos(\breve{x}_6) \sin(\breve{x}_7)$$

Reactor We introduce the following parameters $p_1, \ldots, p_{21}$.

$p_1 \approx 2000$	$p_2 \approx 0.02$	$p_3 \approx 0.1$
$p_4 \approx 20000$	$p_5 \approx 0.3$	$p_6 \approx 0.1$
$p_7 \approx 0.3$	$p_8 \approx 0.2$	$p_9 \approx 0.2$
$p_{10} \approx 40$	$p_{11} \approx 2$	$p_{12} \approx 500$
$p_{13} \approx 1$	$p_{14} \approx 200$	$p_{15} \approx 0.5$
$p_{16} \approx 0.7$	$p_{17} \approx 0.4$	$p_{18} \approx 0.3$
$p_{19} \approx 1.0$	$p_{20} \approx 0.6$	$p_{21} \approx 0.7$

$\boldsymbol{x}_1 = [10^{-6}, 100]$,	$\psi_1 = [10^{-8}]$	
$\boldsymbol{x}_i = [10^{-6}, 1]$,	$\psi_i = [10^{-7}]$	for $i = 2, 6, 9$
$\boldsymbol{x}_i = [10^{-6}, 1]$,	$\psi_i = [10^{-9}]$	for $i = 3, 4, 5, 14$
$\boldsymbol{x}_i = [10^{-6}, 10]$,	$\psi_i = [10^{-6}]$	for $i = 7, 8, 10, 11, 12$
$\boldsymbol{x}_{13} = [10^{-6}, 1000]$,	$\psi_{13} = [10^{-4}]$	
$\boldsymbol{x}_{15} = [0, 1]$,	$\psi_{15} = [10^{-7}]$	
$\boldsymbol{x}_i = [-1, 1]$,	$\psi_i = [10^{-9}]$	for $i = 16, 17, 18, 23, 27, 28$
$\boldsymbol{x}_i = [-1, 1]$,	$\psi_i = [10^{-8}]$	for $i = 19, 20, 22, 24, 25$
$\boldsymbol{x}_i = [-1, 1]$,	$\psi_i = [10^{-7}]$	for $i = 21, 26$
$\boldsymbol{x}_{29} = [0, 10]$,	$\psi_{29} = [10^{-6}]$	

$$f_1(x) = x_{29} \cdot p_3 - 0.01 \cdot x_1 \cdot x_2 - 1/6 \cdot x_{10}$$

$$f_2(x) = p_{16} \cdot 10^{-1} \cdot x_{29} - 100 \cdot x_1 \cdot x_3 + 1 \cdot x_{10}/6$$

$$f_3(x) = p_{17} \cdot x_{29} - 100 \cdot x_1 \cdot x_4$$

$$f_4(x) = -100 \cdot x_1 \cdot x_5 + 2 \cdot x_{10}/3 + x_{29} \cdot x_{14}$$

$$f_5(x) = -x_1 \cdot x_6 + x_7 - x_8 + x_9$$

$$f_6(x) = -1 + 10^{-4} \cdot x_2 + x_3 + x_4 + x_5 + 0.01 \cdot x_6$$

$$f_7(x) = p_5 \cdot 0.1 \cdot x_8 - p_5 \cdot 0.1 \cdot x_9 - p_5 \cdot 0.1 \cdot x_{10}$$

$$f_8(x) = -p_{10} \cdot p_1 \cdot (p_6 \cdot 10^{-3} - p_7 \cdot 10^{-4} \cdot x_4 - p_8 \cdot 10^{-4} \cdot x_5 - p_9 \cdot 10^{-3} \cdot x_6) + x_7$$

$$f_9(x) = x_8 - p_4 \cdot p_2 \cdot x_6 \cdot (1 - 0.1 \cdot x_{11} - 0.1 \cdot x_{12})$$

$$f_{10}(x) = x_9 - p_{11} \cdot p_2 \cdot x_{11}$$

$$f_{11}(x) = x_{10} - p_{12} \cdot p_2 \cdot x_2 \cdot x_{11}$$

$$f_{12}(x) = -p_9 \cdot 10^{-5} \cdot x_2/(p_{13} \cdot 10^{-7} + p_9 \cdot 10^{-5} \cdot x_2) + 0.1 \cdot x_{12}$$

$$f_{13}(x) = -p_{14} \cdot x_{29} \cdot p_3 + 35 \cdot p_2 \cdot x_{13}$$

$$f_{14}(x) = -p_{15} + p_3 + x_{14}$$

$$f_{15}(x) = -0.01 \cdot x_{15} \cdot x_2 - 100 \cdot x_{16} \cdot x_3 - 100 \cdot x_{17} \cdot x_4 - 100 \cdot x_{18} \cdot x_5 - x_{19} \cdot x_6$$

$$\begin{aligned} f_{16}(x) = & -0.01 \cdot x_{15} \cdot x_1 + 0.1 \cdot 10^{-3} \cdot x_{20} - p_{12} \cdot x_{25} \cdot p_2 \cdot x_{11} \\ & + x_{26} \cdot (-p_9 \cdot 10^{-5}/(p_{13} \cdot 10^{-7} + p_9 \cdot 10^{-5} \cdot x_2) \\ & + p_{18} \cdot 10^{-11} \cdot x_2/(p_{13} \cdot 10^{-7} + p_9 \cdot 10^{-5} \cdot x_2)^2) \end{aligned}$$

$$f_{17}(x) = -100 \cdot x_{16} \cdot x_1 + x_{20}$$

$$f_{18}(x) = -100 \cdot x_{17} \cdot x_1 + x_{20} + p_{19} \cdot 10^{-3} \cdot x_{22} \cdot p_1$$

$$f_{19}(x) = -100 \cdot x_{18} \cdot x_1 + x_{20} + p_{20} \cdot 10^{-3} \cdot x_{22} \cdot p_1$$

$$\begin{aligned} f_{20}(x) = & -x_{19} \cdot x_1 + 0.1 \cdot 0.1 \cdot x_{20} + p_{21} \cdot 10^{-2} \cdot x_{22} \cdot p_1 \\ & - p_4 \cdot x_{23} \cdot p_2 \cdot (1 - 0.1 \cdot x_{11} - 0.1 \cdot x_{12}) \end{aligned}$$

$$f_{21}(x) = x_{19} + x_{22}$$

$$f_{22}(x) = -x_{19} + p_5 \cdot 0.1 \cdot x_{21} + x_{23}$$

$$f_{23}(x) = x_{19} - p_5 \cdot 0.1 \cdot x_{21} + x_{24}$$

$$f_{24}(x) = -1/6 \cdot x_{15} + 1/6 \cdot x_{16} + 2/3 \cdot x_{18} - p_5 \cdot 0.1 \cdot x_{21} + x_{25}$$

$$f_{25}(x) = 0.1 \cdot p_4 \cdot x_{23} \cdot p_2 \cdot x_6 - p_{11} \cdot x_{24} \cdot p_2 - p_{12} \cdot x_{25} \cdot p_2 \cdot x_2$$

$$f_{26}(x) = 0.1 \cdot p_4 \cdot x_{23} \cdot p_2 \cdot x_6 + 0.1 \cdot x_{26}$$

$$f_{27}(x) = 35 \cdot x_{27} \cdot p_2$$

$$f_{28}(x) = x_{18} \cdot x_{29} + x_{28}$$

$$\begin{aligned} f_{29}(x) = {} & x_{15}^2 + x_{16}^2 + x_{17}^2 + x_{18}^2 + x_{19}^2 + x_{20}^2 + x_{21}^2 + x_{22}^2 + x_{23}^2 \\ & + x_{24}^2 + x_{25}^2 + x_{26}^2 + x_{27}^2 + x_{28}^2 \end{aligned}$$

Chemistry1

$$\psi_i = [10^{-8}] \quad \text{for } i = 1, \ldots, 12$$

$$\begin{aligned} x_1 &\approx [0.6, 0.8] \\ x_2 &= [0, 1] \\ x_3 &= [0, 1] \\ x_4 &\approx [-1, 9] \\ x_5 &\approx [-10, 20] \\ x_6 &\approx [0, 100] \\ x_7 &\approx [-2000, 5000] \\ x_8 &= [-1, 1] \\ x_9 &= [-1, 1] \\ x_{10} &= [-1, 1] \\ x_{11} &= [-1, 1] \\ x_{12} &= [-\infty, \infty] \\ p &\approx [0.2, 0.4] \end{aligned}$$

$$f_1(x) = \ -x_2 - (x_3 - 1) \cdot \exp(25/3 \cdot p \cdot x_3) \cdot x_1 \cdot x_2$$

$$f_2(x) = \ -x_3 - (x_3 - 1) \cdot (5.4 + 180 \cdot p) \cdot \exp(25/3 \cdot p \cdot x_3)/(5.4 - 180 \cdot p \cdot x_3 + 180 \cdot p) \cdot x_1 \cdot x_2$$

$$f_3(x) = x_4 + 1 + (x_3 - 1) \cdot \exp(25/3 \cdot p \cdot x_3) \cdot x_1$$

$$f_4(x) = x_5 + \exp(25/3 \cdot p \cdot x_3) \cdot x_1 \cdot x_2 + 25/3 \cdot (x_3 - 1) \cdot p \cdot \exp(25/3 \cdot p \cdot x_3) \cdot x_1 \cdot x_2$$

$$f_5(x) = x_6 + (x_3 - 1) \cdot (5.4 + 180 \cdot p) \cdot \exp(25/3 \cdot p \cdot x_3)/(5.4 - 180 \cdot p \cdot x_3 + 180 \cdot p) \cdot x_1$$

$$\begin{aligned} f_6(x) = {} & x_7 + 1 + (5.4 + 180 \cdot p) \cdot \exp(25/3 \cdot p \cdot x_3)/(5.4 - 180 \cdot p \cdot x_3 + 180 \cdot p) \\ & \cdot x_1 \cdot x_2 + 25/3 \cdot (x_3 - 1) \cdot (5.4 + 180 \cdot p) \cdot p \cdot \exp(25/3 \cdot p \cdot x_3) \\ & /(5.4 - 180 \cdot p \cdot x_3 + 180 \cdot p) \cdot x_1 \cdot x_2 + 180 \cdot (x_3 - 1) \cdot (5.4 + 180 \cdot p) \\ & \cdot \exp(25/3 \cdot p \cdot x_3)/(5.4 - 180 \cdot p \cdot x_3 + 180 \cdot p)^2 \cdot x_1 \cdot x_2 \cdot p \end{aligned}$$

$$f_7(x) = x_4 \cdot x_8 + x_5 \cdot x_9 + x_{12} \cdot x_{10}$$

$$f_8(x) = x_6 \cdot x_8 + x_7 \cdot x_9 + x_{12} \cdot x_{11}$$

$$f_9(x) = x_4 \cdot x_{10} + x_5 \cdot x_{11} - x_{12} \cdot x_8$$

$$f_{10}(x) = x_6 \cdot x_{10} + x_7 \cdot x_{11} - x_{12} \cdot x_9$$

$$f_{11}(x) = x_8^2 + x_9^2 + x_{10}^2 + x_{11}^2 - 1$$

$$f_{12}(x) = x_8 \cdot x_{10} + x_9 \cdot x_{11}$$

Trigexp1 [COP] with $n = 50$
(Be aware of typos in [COP] itself. The system is given as in reference cited by [COP].)

$$\boldsymbol{x}_i = [-100, 100], \quad \psi_i = [10^{-6}] \quad \text{for } i = 1, \ldots, n$$

$$\begin{aligned} f_1(x) &= 3x_1^3 + 2x_2 - 5 + \sin(x_1 - x_2) \cdot \sin(x_1 + x_2) \\ f_i(x) &= 3x_i^3 + 2x_{i+1} - 5 + \sin(x_i - x_{i+1}) \cdot \sin(x_i + x_{i+1}) + 4x_i - x_{i-1} \exp(x_{i-1} - x_i) - 3 \\ &\quad \text{for } i = 2, \ldots, n-1 \\ f_n(x) &= 4 \cdot x_n - x_{n-1} \cdot \exp(x_{n-1} - x_n) - 3 \end{aligned}$$

Chemistry2

$$\begin{aligned} x_1 &\approx [0.5, 0.6] \\ x_2 &= [0, 1] \\ x_3 &= [0, 1] \\ x_4 &= [-1, 1] \\ x_5 &= [-1, 1] \\ x_6 &\approx [0.2, 0.4] \end{aligned}$$

$$\psi_i = [10^{-5}] \quad \text{for } i = 1, \ldots, 6$$

For shorter denotation, additional variables to display the problem. (These were *not* part of the system as used for calculations.)

$$\begin{aligned} y_1 &= 25/3 \cdot x_6 & y_2 &= \exp(y_1 \cdot x_3) & y_3 &= 5.4 + 180 \cdot x_6 \\ y_4 &= 5.4 - 180 \cdot x_6 \cdot x_3 & y_5 &= y_4 + 180 \cdot x_6 & y_6 &= (x_3 - 1) \cdot y_2 \cdot y_3 \end{aligned}$$

$$f_1(x) = -x_2 - y_6 \cdot x_1 \cdot x_2$$

$$f_2(x) = -x_3 - y_6 \cdot y_3/(y_4 + 180 \cdot x_6) \cdot x_1 \cdot x_2$$

$$f_3(x) = (-1 - y_6 \cdot x_1) \cdot x_4 - (y_2 + y_6 \cdot y_1) \cdot x_1 \cdot x_2 \cdot x_5$$

$$f_4(x) = -y_6/y_5 \cdot x_1 \cdot x_4$$

$$+ x_5 \cdot ((-1) - ((((y_3 \cdot y_2 + y_6 \cdot y_1) \cdot y_5 - y_6 \cdot (-180 \cdot x_6))/y_5^2) \cdot x_1) \cdot x_2)$$

$$f_5(x) = 1 - x_4^2 - x_5^2$$

DirectKinematics [COP], [Bee06]

We use the following parameters.

$$\begin{aligned}
p_1 &= 12.80624847 & p_2 &= 1.570796327 & p_3 &= 304.0192 \\
p_4 &= 7.011678 & p_5 &= 4.065716 & p_6 &= 25.46010 \\
p_7 &= 28.04672 & p_8 &= 420.7008 & p_9 &= 243.9430 \\
p_{10} &= 16.26286 & p_{11} &= 196.3270 & p_{12} &= 113.8400 \\
p_{13} &= 1.54924278922385
\end{aligned}$$

The variables for this problem are given as follows.

$$\begin{aligned}
x_c &= [-p_1, p_1] & y_c &= [-p_1, p_1] & z_c &= [0, p_1] \\
p &= [-p_2, p_2] & t &= [-p_2, p_2] & x_1 &= [-12.0, -2.0] \\
y_1 &= [10.0, 20.0] & z_1 &= [-5.0, 5.0] & x_2 &= [2.0, 12.0] \\
y_2 &= [10.0, 20.0] & z_2 &= [-5.0, 5.0]
\end{aligned}$$

The demanded precision ψ_i is 10^{-6} for all variables.

$$f_1(x) = x_c^2 + y_c^2 + z_c^2 - 164$$

$$\begin{aligned}
f_2(x) = p_3 &- 20 \cdot x_c - 300 \cdot \cos(p) + 100 \cdot \sin(p) \cdot \cos(t) - 10 \cdot y_c \\
&- 150 \cdot \sin(p) - 50 \cdot \cos(p) \cdot \cos(t) + 30 \cdot x_c \cdot \cos(p) \\
&- 10 \cdot x_c \cdot \sin(p) \cdot \cos(t) + 30 \cdot y_c \cdot \sin(p) \\
&+ 10 \cdot y_c \cdot \cos(p) \cdot \cos(t) + 10 \cdot z_c \cdot \sin(t)
\end{aligned}$$

$$\begin{aligned}
f_3(x) = p_3 &+ 20 \cdot x_c - 300 \cdot \cos(p) - 100 \cdot \sin(p) \cdot \cos(t) - 10 \cdot y_c \\
&+ 150 \cdot \sin(p) - 50 \cdot \cos(p) \cdot \cos(t) - 30 \cdot x_c \cdot \cos(p) \\
&- 10 \cdot x_c \cdot \sin(p) \cdot \cos(t) - 30 \cdot y_c \cdot \sin(p) \\
&+ 10 \cdot y_c \cdot \cos(p) \cdot \cos(t) + 10 \cdot z_c \cdot \sin(t)
\end{aligned}$$

$$f_4(x) = (x_1 + 7)^2 + (y_1 - 15)^2 + z_1^2 - 25$$

$$\begin{aligned}
f_5(x) = &(\cos(p) \cdot (x_1 - x_c) + \sin(p) \cdot (y_1 - y_c) + 7)^2 \\
&+ (-\sin(p) \cdot \cos(t) \cdot (x_1 - x_c) + \cos(p) \cdot \cos(t) \cdot (y_1 - y_c) \\
&+ \sin(t) \cdot (z_1 - z_c) - p_4)^2 \\
&+ (\sin(p) \cdot \sin(t) \cdot (x_1 - x_c) - \cos(p) \cdot \sin(t) \cdot (y_1 - y_c) \\
&+ \cos(t) \cdot (z_1 - z_c) + p_5)^2 - p_6
\end{aligned}$$

$$f_6(x) = (x_2 - 7)^2 + (y_2 - 15)^2 + z_2^2 - 25$$

$$f_7(x) = (\cos(p) \cdot (x_2 - x_c) + \sin(p) \cdot (y_2 - y_c) - 7)^2$$

$$
\begin{aligned}
&+ (-\sin(p) \cdot \cos(t) \cdot (x_2 - x_c) + \cos(p) \cdot \cos(t) \cdot (y_2 - y_c) \\
&+ \sin(t) \cdot (z_2 - z_c) - p_4)^2 \\
&+ (\sin(p) \cdot \sin(t) \cdot (x_2 - x_c) - \cos(p) \cdot \sin(t) \cdot (y_2 - y_c) \\
&+ \cos(t) \cdot (z_2 - z_c) + p_5)^2 - p_6
\end{aligned}
$$

$$
\begin{aligned}
f_8(x) = {} & 60 \cdot z_1 - 4 \cdot z_1 \cdot y_c + (p_7 \cdot y_1 - p_8) \cdot \sin(t) + (p_9 - p_{10} \cdot y_1) \cdot \cos(t) \\
&+ 28 \cdot z_1 \cdot \sin(p) + (-p_{10} \cdot \sin(t) \cdot z_1 - p_7 \cdot \cos(t) \cdot z_1) \cdot \cos(p) \\
&+ (-60 + 4 \cdot y_1) \cdot z_c
\end{aligned}
$$

$$
\begin{aligned}
f_9(x) = {} & 28 \cdot z_1 + 4 \cdot z_1 \cdot x_c + (-p_{11} - p_7 \cdot x_1) \cdot \sin(t) + (p_{12} + p_{10} \cdot x_1) \cdot \cos(t) \\
&+ (-p_{10} \cdot \sin(t) \cdot z_1 - p_7 \cdot \cos(t) \cdot z_1) \cdot \sin(p) - 28 \cdot z_1 \cdot \cos(p) \\
&+ (-4 \cdot x_1 - 28.) \cdot z_c
\end{aligned}
$$

$$
\begin{aligned}
f_{10}(x) = {} & 60 \cdot z_2 - 4 \cdot z_2 \cdot y_c + (-p_8 + p_7 \cdot y_2) \cdot \sin(t) + (p_9 - p_{10} \cdot y_2) \cdot \cos(t) \\
&- 28 \cdot z_2 \cdot \sin(p) + (-p_7 \cdot \cos(t) \cdot z_2 - p_{10} \cdot \sin(t) \cdot z_2) \cdot \cos(p) \\
&+ (4 \cdot y_2 - 60) \cdot z_c
\end{aligned}
$$

$$
\begin{aligned}
f_{11}(x) = {} & - 28 \cdot z_2 + 4 \cdot z_2 \cdot x_c + (p_{11} - p_7 \cdot x_2) \cdot \sin(t) + (-p_{12} + p_{10} \cdot x_2) \cdot \cos(t) \\
&+ (-p_7 \cdot \cos(t) \cdot z_2 - p_{10} \cdot \sin(t) \cdot z_2) \cdot \sin(p) + 28 \cdot z_2 \cdot \cos(p) \\
&+ (-4 \cdot x_2 + 28) \cdot z_c
\end{aligned}
$$

DesignProblem9 [Bee06] , [COP]

$$\boldsymbol{x}_i = [0, 10], \quad \psi_i = [10^{-4}] \quad \text{for } i = 1, \ldots, 9$$

Additionally, we introduce the following parameters.

$$\begin{aligned} p_1 &= 28.5132 & p_2 &= 23.3037 \\ p_3 &= 111.8467 & p_4 &= 101.779 \\ p_5 &= 134.3884 & p_6 &= 111.461 \\ p_7 &= 211.4823 & p_8 &= 191.267 \\ p_9 &= 0.0052095 & p_{10} &= 0.0100677 \\ p_{11} &= 0.0229274 & p_{12} &= 0.0202153 \end{aligned}$$

$$\begin{aligned} f_1(x) = & -p_1 + p_2 \cdot x_2 \\ & + (1 - x_1 \cdot x_2) \cdot x_3 \cdot (\exp(x_5 \cdot (0.485 - p_9 \cdot x_7 - 0.0285132 \cdot x_8)) - 1) \end{aligned}$$

$$\begin{aligned} f_2(x) = & \, p_2 - p_1 \cdot x_1 \\ & + (1 - x_1 \cdot x_2) \cdot x_4 \cdot (\exp(x_6 \cdot (0.116 - p_9 \cdot x_7 + 0.0233037 \cdot x_9)) - 1) \end{aligned}$$

$$\begin{aligned} f_3(x) = & -p_3 + p_4 \cdot x_2 \\ & + (1 - x_1 \cdot x_2) \cdot x_3 \cdot (\exp(x_5 \cdot (0.752 - p_{10} \cdot x_7 - 0.1118467 \cdot x_8)) - 1) \end{aligned}$$

$$\begin{aligned} f_4(x) = & \, p_4 - p_3 \cdot x_1 \\ & + (1 - x_1 \cdot x_2) \cdot x_4 \cdot (\exp(x_6 \cdot (-0.502 - p_{10} \cdot x_7 + 0.101779 \cdot x_9)) - 1) \end{aligned}$$

$$\begin{aligned} f_5(x) = & -p_5 + p_6 \cdot x_2 \\ & + (1 - x_1 \cdot x_2) \cdot x_3 \cdot (\exp(x_5 \cdot (0.869 - p_{11} \cdot x_7 - 0.1343884 \cdot x_8)) - 1) \end{aligned}$$

$$\begin{aligned} f_6(x) = & \, p_6 - p_5 \cdot x_1 \\ & + (1 - x_1 \cdot x_2) \cdot x_4 \cdot (\exp(x_6 \cdot (0.166 - p_{11} \cdot x_7 + 0.111461 \cdot x_9)) - 1) \end{aligned}$$

$$\begin{aligned} f_7(x) = & -p_7 + p_8 \cdot x_2 \\ & + (1 - x_1 \cdot x_2) \cdot x_3 \cdot (\exp(x_5 \cdot (0.982 - p_{12} \cdot x_7 - 0.2114823 \cdot x_8)) - 1) \end{aligned}$$

$$\begin{aligned} f_8(x) = & \, p_8 - p_7 \cdot x_1 \\ & + (1 - x_1 \cdot x_2) \cdot x_4 \cdot (\exp(x_6 \cdot (-0.473 - p_{12} \cdot x_7 + 0.191267 \cdot x_9)) - 1) \end{aligned}$$

$$f_9(x) = x_1 \cdot x_3 - x_2 \cdot x_4$$

Chemistry3 The system is the same as for *Chemistry1*, only the demanded precision and the startbox differ.

$$\psi_i = [10^{-5}] \quad \text{for } i = 1, \ldots, 12$$

$$\begin{aligned} x_1 &\approx [0.5, 0.6] \\ x_2 &= [0, 1] \\ x_3 &= [0, 1] \\ x_4 &\approx [-1, 7] \\ x_5 &\approx [-10, 20] \\ x_6 &\approx [0, 100] \\ x_7 &\approx [-1000, 4000] \\ x_8 &= [-1, 1] \\ x_9 &= [-1, 1] \\ x_{10} &= [-1, 1] \\ x_{11} &= [-1, 1] \\ x_{12} &= [-10, 10] \\ p &\approx [0.2, 0.4] \end{aligned}$$

Test set $\mathcal{T}_V$

Test set $\mathcal{T}_V$ is a subset of test set $\mathcal{T}_B$ utilized for verification. It comprises problems for which at least one solution box was computed, but that do not contain a manifold of solutions. Namely those are *G7_gradientsystem*, *Reactor*, *Brent7*, *Eco9*, *Trigexp1*, *Trigonometric*, *DesignProblem9* and *7erSystem*.

Bibliography

[AFKL04] René Alt, Andreas Frommer, R. Baker Kearfott, and Wolfram Luther, editors. *Numerical Software with Result Verification, International Dagstuhl Seminar, Dagstuhl Castle, Germany, January 19-24, 2003, Revised Papers*, volume 2991 of *Lecture Notes in Computer Science*. Springer, 2004.

[AH83] G. Alefeld and J. Herzberger. *Introduction to interval computations*. Computer science and applied mathematics. Academic Press, 1983.

[AMFH02] Götz Alefeld, Jan Mayer, Andreas Frommer, and Gerhard Heindl. On the existence theorems of Kantorovich, Miranda and Borsuk, 2002.

[ANT12] Ignacio Araya, Bertrand Neveu, and Gilles Trombettoni. An interval extension based on occurrence grouping. *Computing*, 94(2-4):173–188, March 2012.

[Bal10] Lars Balzer. Evaluation und Anpassung von Software zur linearen Optimierung im Kontext des nichtlinearen Lösers SONIC. Bachelor-thesis, Bergische Universität Wuppertal, 2010.

[Bal13] Lars Balzer. Untersuchung des Einsatzes von Taylor-Modellen bei der Lösung nichtlinearer Gleichungssysteme. Master-thesis, Bergische Universität Wuppertal, 2013.

[Bau88] Eckart Baumann. Optimal centered forms. *BIT Numerical Mathematics*, 28:80–87, 1988. 10.1007/BF01934696.

[BBLSA04] Thomas Beelitz, Christian Bischof, Bruno Lang, and Klaus Schulte Althoff. Result-verifying solution of nonlinear systems in the analysis of chemical processes. In René Alt, Andreas Frommer, R. Baker Kearfott, and Wolfram Luther, editors, *Numerical Software with Result Verification*, volume 2991 of *Lecture Notes in Computer Science*, pages 198–205. Springer Berlin Heidelberg, 2004.

[BBLW04] Thomas Beelitz, Christian H. Bischof, Bruno Lang, and Paul Willems. SONIC - a framework for the rigorous solution of nonlinear problems. *Tech. Rep. BUW-SC 2004/7, University of Wuppertal*, 2004.

[Bee06] Thomas Beelitz. *Effiziente Methoden zum verifizierten Lösen von Optimierungsaufgaben und nichtlinearen Gleichungssystemen*. PhD thesis, Bergische Universität Wuppertal, 2006.

[Ber] Webpage of the work group of Martin Berz and Kyoko Makino. `http://www.bt.pa.msu.edu/pub/`.

[Ber95] Sonja Berner. *Ein paralleles Verfahren zur verifizierten globalen Optimierung*. Dissertation, Bergische Universität Wuppertal, Deutschland, August 1995.

[BFLW05] Thomas Beelitz, Andreas Frommer, Bruno Lang, and Paul Willems. Symbolic–numeric techniques for solving nonlinear systems. *PAMM*, 5(1):705–708, 2005.

[BFLW09] Thomas Beelitz, Andreas Frommer, Bruno Lang, and Paul Willems. A framework for existence tests based on the topological degree and homotopy. *Numer. Math.*, 111(4):493–507, 2009.

[BH01] Martin Berz and Jens Hoefkens. Verified High-Order Inversion of Functional Depedencies and Interval Newton Methods. *Reliab. Comput.*, 7(5):379–398, 2001.

[BLB03] Thomas Beelitz, Bruno Lang, and Christian H. Bischof. Intervals and OpenMP : towards an efficient parallel result-verifying nonlinear solver. In *Proceedings of the 5th European Workshop on OpenMP, Aachen, Germany, September 22-26, 2003*, pages 119–125, Aachen, Germany, 2003.

[BLB06] Thomas Beelitz, Bruno Lang, and Christian H. Bischof. Efficient task scheduling in the parallel result-verifying solution of nonlinear systems. *Reliab. Comput.*, 12(2):141–151, 2006.

[BLMM01] Christian H. Bischof, Bruno Lang, Wolfgang Marquardt, and Martin Mönnigmann. Verified determination of singularities in chemical processes. In Walter Krämer and Jürgen Wolff von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 305–316, New York, 2001. Kluwer Academic/Plenum Publishers.

[BLUW09] Thomas Beelitz, Bruno Lang, Peer Ueberholz, and Paul Willems. Closing the Case $t = 3$ for 3D Spherical t-Designs Using a Result-Verifying Nonlinear Solver, 2009. Preprint, `http://www-ai.math.uni-wuppertal.de/SciComp/preprints/SC0903.ps.gz`.

[BM09] Martin Berz and Kyoko Makino. Rigorous global search using Taylor models. In *Proceedings of the 2009 conference on Symbolic numeric computation*, SNC '09, pages 11–20, New York, NY, USA, 2009. ACM.

[Bor33] Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 21:177–190, 1933.

[CFL09] Xiaojun Chen, Andreas Frommer, and Bruno Lang. Computational Existence Proofs for Spherical t-Designs, 2009. Preprint, `http://www-ai.math.uni-wuppertal.de/SciComp/preprints/SC0903.ps.gz`.

[COP] COPRIN Homepage. `http://www-sop.inria.fr/coprin/index_english.html`.

[Cos] Cosy Infinity Homepage. `http://bt.pa.msu.edu/index_cosy.htm`.

[cxs] C-XSC Homepage. `http://www2.math.uni-wuppertal.de/~xsc/xsc/cxsc.html`.

[Dei85] Klaus Deimling. *Nonlinear Functional Analysis*. Springer, Berlin, Heidelberg, New York, 1985.

[Ebl07] Ingo Eble. *Über Taylor-Modelle*. PhD thesis, Universität Karlsruhe, 2007.

[FHL07] Andreas Frommer, Fatmir Hoxha, and Bruno Lang. Proving the existence of zeros using the topological degree and interval arithmetic. *J. Comput. Appl. Math.*, 199(2):397–402, 2007.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[fil] filib++ Homepage. `http://www2.math.uni-wuppertal.de/~xsc/software/filib.html`.

[FL04] Andreas Frommer and Bruno Lang. On preconditioners for the Borsuk existence test. *PAMM*, 4(1):638–639, 2004.

[FL05] Andreas Frommer and Bruno Lang. Existence tests for solutions of nonlinear equations using Borsuk's theorem. *SIAM Journal on Numerical Analysis*, 43(3):1348–1361, jun 2005.

[FLS04] Andreas Frommer, Bruno Lang, and Marco Schnurr. A Comparison of the Moore and Miranda Existence Tests. *Computing*, 72:349–354, 2004. 10.1007/s00607-004-0064-4.

[Fro01] Andreas Frommer. Proving conjectures by use of interval arithmetic. In Ulrich Kulisch, Rudolf Lohner, and Axel Facius, editors, *Perspectives on Enclosure Methods*, pages 1–12. Springer Vienna, 2001.

[GVL89] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. John Hopkins University Press, Baltimore, MD, UDA, 2nd edition, 1989.

[Han92] Eldon Hanson. *Global Optimization Using Interval Analysis*. Marcel Dekker, 1992.

[HB02] Jens Hoefkens and Martin Berz. Verification of invertibility of complicated functions over large domains. *Reliab. Comput.*, 8(1):67–82, 2002.

[HJvE99] Timothy J. Hickey, Q. Ju, and M.H. van Emden. Interval arithmetic: from principles to implementation. Technical report, Brandeis University CS, April 1999.

[HK03] Werner Hofschuster and Walter Krämer. C-XSC 2.0: A C++ library for extended scientific computing. In Alt et al. [AFKL04], pages 15–35.

[Hoe01] Jens Hoefkens. *Rigorous numerical analysis with high-order Taylor Models*. PhD thesis, Michigan State University, 2001. Dissertation, Michigan State University.

[HS96] R. H. Hardin and N. J. A. Sloane. McLaren's improved snub cube and other new spherical designs in three dimensions. *Discrete & Computational Geometry*, 15(4):429–441, 1996.

[HvEW98] Timothy J. Hickey, M.H. van Emden, and H. Wu. A unified framework for interval constraints and interval arithmetic. In M. Maher and J-F. Puget, editors, *Principles and Practice of Constraint Programming*, volume 1520 of *LNCS*, pages 250–264. Springer-Verlag, 1998.

[ibe] Ibex Homepage. `http://www.emn.fr/z-info/ibex/`.

[iee08] IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, New York, August 2008.

[JKDW01] Luc Jaulin, Michel Kieffer, Oliver Didrit, and Éric Walter. *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, London, 2001.

[JL12] Elke Just and Bruno Lang. Success-guided selection of expanded systems for result-verifying nonlinear solvers. *Reliab. Comput.*, 16:73–83, 2012.

[Kea90] R. Baker Kearfott. Preconditioners for the interval Gauss-Seidel method. *SIAM Journal on Numerical Analysis*, 27(3):804–822, June 1990.

[Kea91] R. Baker Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. *Computing*, 47(2):169–191, July 1991.

[Kea96a] R. Baker Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2:95–112, 1996.

[Kea96b] R. Baker Kearfott. *Rigorous Global Search: Continuous Problems*. Nonconvex Optimization and Its Applications. Kluwer Academic Publishers, 1996.

[Kea09] R. Baker Kearfott. Globsol user guide. *Optimization Methods and Software 24 (4-5)*, pages 687–708, August, 2009.

[KHN91] R. Baker Kearfott, Chenyi Hu, and Manuel Novoa III. A review of preconditioners for the interval Gauss–Seidel method. *Interval Computations*, 1(1):59–85, 1991.

[Kie03] Andre Kienitz. Analysis of Taylor Models for the Verified Solution of Systems of Equations. Diploma thesis, RWTH Aachen University, 2003.

[KN90] R. Baker Kearfott and Manuel Novoa III. Algorithm 681: INTBIS, a portable interval Newton/bisection package. *ACM Trans. Math. Software*, 16(2):152–157, 1990.

[KNN+02] R. Baker Kearfott, Mitsuhiro T. Nakao, Arnold Neumaier, Siegfried M. Rump, Sergey P. Shary, and Pascal van Hentenryck. Standardized notation in interval analysis. In *In Proc. XIII Baikal International School-seminar "Optimization methods and their applications". Vol. 4 "Interval analysis". Irkutsk: Institute of Energy Systems SB RAS*, pages 106–113, 2002.

[KS96] R. Baker Kearfott and Xiaofa Shi. Optimal preconditioners for interval Gauss-Seidel methods. In Götz Alefeld and Andreas Frommer, editors, *Scientific Computing and Validated Numerics*, pages 173–178. Akademie Verlag, 1996.

[KW10] Bartłomiej Jacek Kubica and Adam Woźniak. An interval method for seeking the Nash equilibria of non-cooperative games. In *Proceedings of the 8th international conference on Parallel processing and applied mathematics: Part II*, PPAM'09, pages 446–455, Berlin, Heidelberg, 2010. Springer-Verlag.

[LS03] Youdong Lin and Mark A. Stadtherr. LP-based strategies for modeling and optimization using interval methods. In *Fourth International Conference on Foundations of Computer Aided Process Operations (FOCAPO 2003)*, Coral Springs, Florida, 2003.

[LTG+06] Michael Lerch, German Tischler, Jürgen Wolff von Gudenberg, Werner Hofschuster, and Walter Krämer. Filib++, a fast interval library supporting containment computations. *ACM Trans. Math. Softw.*, 32(2):299–324, 2006.

[Mak98] Kyoko Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators*. PhD thesis, Michigan State University, 1998.

[May95] Günter Mayer. Epsilon-inflation in verification algorithms. *Journal of Computational and Applied Mathematics*, 60(1–2):147–169, 1995. Proceedings of the International Meeting on Linear/Nonlinear Iterative Methods and Verification of Solution.

[Mir40] Carlo Miranda. Un' osservazione su un teorema di Brouwer. *Bolletino Unione Mathematica Italiana*, pages 5–7, 1940.

[MK80] Ramon E. Moore and J. B. Kioustelidis. A simple test for accuracy of approximate solutions to nonlinear (or linear) systems. *SIAM Journal on Numerical Analysis*, 17(4):521–529, 1980.

[MMB+07] Martin Mönnigmann, Wolfgang Marquardt, Christian H. Bischof, Thomas Beelitz, Bruno Lang, and Paul Willems. A hybrid approach for efficient robust design of dynamic systems. *SIAM Rev.*, 49(2):236–254, 2007.

[Moo66] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[Moo77] Ramon E. Moore. A test for existence of solutions to nonlinear systems. *SIAM Journal on Numerical Analysis*, 14:611–615, 1977.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, UK, 1990.

[Neu02] Arnold Neumaier. Taylor forms - use and limits. *Reliab. Comput.*, 9:43–79, 2002.

[Nov93] Manuel Novoa III. Theory of preconditioners for the interval Gauss–Seidel method and existence / uniqueness theory with interval Newton methods. Department of Mathematics, University of Southwestern Louisiana, U.S.L. Box 4-1010, Lafayette, La 70504, 1993.

[NSHV05] Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer, and Tamás Vinkó. A comparison of complete global optimization solvers. *Math. Program.*, 103(2):335–356, 2005.

[Ral80] L. B. Rall. A comparison of the existence theorems of Kantorovich and Moore. *SIAM Journal on Numerical Analysis*, 17(1):148–161, 1980.

[Rat96] Dietmar Ratz. On extended interval arithmetic and inclusion isotonicity. Technical report, Institut für Angewandte Mathematik, Universität Karlsruhe, 1996.

[RiO] RiOT. `http://www.math.kit.edu/ianm1/~ingo.eble/en`.

[Roh09] Jiri Rohn. Forty necessary and sufficient conditions for regularity of interval matrices: A survey. *ELA. The Electronic Journal of Linear Algebra [electronic only]*, 18:500–512, 2009.

[RR84] Helmut Ratschek and Jon Rokne. *Computer methods for the range of functions*. Ellis Horwood series in mathematics and its applications. E. Horwood, 1984.

[RR95] Georg Rex and Jiri Rohn. A note on checking regularity of interval matrices. *Linear and Multilinear Algebra*, 39:259–262, 1995.

[Rum10] Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[SA02] Klaus Schulte Althoff. Algorithmen zum verifizierten Lösen nichtlinearer Gleichungssysteme. Diploma thesis, RWTH Aachen University, 2002.

[SCB05] L. Ridgway Scott, Terry Clark, and Babak Bagheri. *Scientific Parallel Computing*. Princeton University Press, Princeton, NJ, 2005.

[SN04] Hermann Schichl and Arnold Neumaier. Exclusion regions for systems of equations. *SIAM J. Numerical Analysis*, 42(1):383–408, 2004.

[Sun] Sun Studio 11: C++ Interval Arithmetic Programming Reference. `http://docs.sun.com/app/docs/doc/819-3696`.

[SW09] Ian H. Sloan and Robert S. Womersley. A variational characterisation of spherical designs. *Journal of Approximation Theory*, 159(2):308–318, 2009.

[Ueb07] Peer Ueberholz. An efficient parallel result-verifying nonlinear solver for large systems. *Science and Supercomputing in Europe*, pages 441–446, 2007.

[weba] Webpage about interval computations. `http://www.cs.utep.edu/interval-comp/`.

[webb] Webpage of the *Reliable Computing* journal. `http://interval.louisiana.edu/reliable-computing-journal`.

[Wil04] Paul Willems. Symbolisch-numerische Techniken zur verifizierten Lösung nichtlinearer Gleichungssysteme. Diploma thesis, RWTH Aachen University, 2004.

Index