A new parallel method for verified global optimization

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

am Fachbereich Mathematik der
Bergischen Universität Gesamthochschule Wuppertal
genehmigte

Dissertation

von

Suiunbek Ibraev

aus Bischkek (Kirgisien)

Tag der mündlichen Prüfung:	17. Juli 2001
Referent:	Prof. Dr. A. Frommer
Korreferent:	Prof. Dr. W. Krämer

Contents

List of Tables

Introduction

Optimization is used in a variety of areas. Often real systems are described through mathematical models. For those systems parameters are sought which optimize the system in some ways. And the search of these parameters leads usually to a problem of global optimization.

In optimization one distinguishes local and global optimization. For local optimization lots of efficient methods already exist. Global optimization is more complicated. Unlike local optimization it is not enough to only rely on local information. The smallest of all local minima should be determined. Real problems usually have many local minima. Therefore methods for global optimization are very computation consuming.

Usually, classical methods compute results which are not verified, i.e. there is no information on how closely the computed results approximate the "true" solution. This can be due to rounding errors on a computer or due to the fact that the method in use cannot provide such information by its very construction. For some problems from the industry it is not allowable to have a non-quantified error in results. This resulted in a new approach in global optimization, so called verified global optimization, which became popular in recent years. Verified methods involve interval arithmetic which guarantees verification of results.

In this work we consider verified global optimization. We analyze the standard method for verified global optimization and its variations occurring in the literature. And we design a new adaptive serial method for verified global optimization. Here we introduce two new strategies: a strategy for box processing after an interval method and a strategy for applying an interval method to boxes. We show that a strategy for determining the success of the interval method is not the best one and give an alternative strategy. This new strategy is an adaptive one. All existing methods either use statically set threshold parameters in the process of handling boxes or use a dynamically changing parameter tying the behavior of the function over unrelated parts of the search area. The new strategy takes into account the different behavior of the given function over different boxes. In the final serial method we incorporate both a new strategy for box processing and a strategy for applying an interval method to boxes. The resulting method is more efficient than existing methods, as our numerical results show.

Through the development of efficient methods for global optimization and through advances in computer technology, problems which were left out of consideration some years ago due to the very long solution time, became today solvable in acceptable time. For the solution of complex problems parallel computers are available. They offer more memory and power than by serial

computers. Most of the complex problems were not possible to solve since they require too much time and a lot of memory. With p processors one expects to solve a problem about p times faster than on one computer. But because processors are not always effectively utilized this speedup is not always achieved.

In this work we present a new approach for the efficient use of processors, memory and communications. This method does not have any constraints on the number of processors of the parallel computer. It runs only one process on each processor unlike some existing methods which need two processes - a scheduler and a worker - running on the same processor, thus loosing time for the context switch between them. The new method introduces a **send a box on demand** idea which seems to be new for parallel methods for verified global optimization. The underlying goal is to minimize unnecessary transference of boxes between processors. The new method also features a centralized management, which reduces the number of communications and simplifies the finalization of the method. The combination of the new idea with a centralized management results in a very efficient parallel method. The efficiency of the new method is confirmed by numerical results and by comparison with an existing method.

We now shortly describe the content of each chapter.
In the first chapter we give the background which is required for understanding the rest of the work. Here notations of terms used throughout the work and notations for the description of algorithms are given. We describe the problem of global optimization and briefly cover classical methods. We explain why interval arithmetic should be used instead of the usual floating point arithmetic on a computer. Then we give basics of interval arithmetic (definitions and rules). After this we introduce the concept of an inclusion function and give several possible ways to compute such functions. We give the definition of the order of inclusion functions, a measure for comparing the "accuracy" of inclusion functions. Then we briefly explain the implementation of computational differentiation which we used in our work. At the end we describe which software we used for the implementation of interval arithmetic and for the implementation of communication in the parallel method.

In Chapter 2 there is an in-depth introduction to verified global optimization. First we give a basic serial method. Then we show several strategies for the box subdivision and the selection of the next box. These strategies have already been well analyzed in the literature and therefore we give them only at an introductory level. We also consider accelerating devices, again at an introductory level. We also introduce two new strategies: a strategy for box processing and a strategy for applying an interval method to boxes. Both strategies improve an existing serial method. The efficiency of these strategies is shown by numerical results. Finalizing this chapter we present a new serial method for verified global optimization, which will later be used in the parallel method. This new serial method uses both of the new strategies introduced.

In Chapter 3 we further consider the interval Krawczyk method, which appeared already as an accelerating device for verified global optimization in Chapter 2. We additionally consider the interval Newton method. These methods are used not only in methods for verified global optimization. They are mainly applied for solving systems of nonlinear equations. We also consider modifications of these methods. We analyze properties of the original and the modified methods and give sufficient conditions for their convergence or divergence. The convergence properties were already considered in the literature. But divergence properties are newly analyzed in our work. We discuss advantages and drawbacks of modified methods, and we give numerical results for comparison.

In Chapter 4 we introduce the concept of parallel computing. First we give basics of parallel computing: Flynn's hardware taxonomy, taxonomy of topologies, new trends in parallel computing, and measure of performance. After that we turn to parallel methods for verified

global optimization. We consider existing parallel approaches for verified global optimization from: Dixon and Jha; Henriksen and Madsen; Eriksson; Moore, Hansen and Leclerc; Berner; Wiethoff. We also give a short analysis of these approaches based on numerical results from their papers. Then we introduce our new method for verified global optimization with the new communication model. After that we go through all routines of the new method in-depth giving a complete algorithmic description. Finally we present numerical results. The efficiency of the new method is shown by comparison with the centralized mediator approach of Berner which is the most efficient of the prior existing methods.

In Appendix A test problems used in this work are presented along with their solution sets. They are subdivided into three groups depending on the time required for their solution: simple, medium and hard problems. For some problems we have plotted graphs of the corresponding objective functions.

Acknowledgement

Here I would like to thank all, who made contribution to this work.

Special thanks go to my supervisor Prof. Dr. Andreas Frommer, the department of mathematics at the university of Wuppertal, who made possible my study at the university, supported me throughout my work by numerous suggestions and advises, and was always ready for discussions.

My colleagues at the institute for scientific computing have been particularly supportive and tolerant of my incessant requests during my stay in Wuppertal. I specifically thank Holger Arndt, Benedikt Grosser, Stefanie Krivsky, Bjoern Medeke, Brigitte Schultz, Andre Weinberg.

Finally, I thank my father and grand mother for their love and support during my study.

Wuppertal, July 2001 Suiunbek Ibraev

Chapter 1

Background

1.1 Terms and Algorithm Notations

In this section we shall explain shortly all the terms that are often used in this work. We also give the notation for expressing algorithms.

Terms

We will take intervals, interval vectors and interval matrices into brackets. Small letters will be used for integers and real numbers, for intervals and for real and interval vectors. Matrices will as usually be written in capital letters. For an interval vector $[x]$ by $\check{x}$ we denote an arbitrary point vector from $[x]$.

We denote the set of all intervals by $\mathbb{IR}$, the set of all n-dimensional interval vectors by $\mathbb{IR}^n$ and the set of all $n \times m$-interval matrices by $\mathbb{IR}^{n\times m}$. We denote the lower bound of an interval $[x]$ by $\inf([x])$ or x^1, the upper bound by $\sup([x])$ or x^2.

For a vector $x \in \mathbb{R}^n$ we denote by $x_i,\ i = 1, \ldots, n$, the i-th component of this vector. For a matrix $A \in \mathbb{R}^{n\times m}$ we denote by $a_{ij},\ i = 1, \ldots, n,\ j = 1,\ \ldots, m$, the j-th element in the i-th row. The same applies to interval vectors and interval matrices.

We use an upper index to denote enumeration, for instance, x^i means i-th real number or i-th vector. For a function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ the following notations apply:

f'	-	first derivative (gradient) of f, $f' : D \to \mathbb{R}^n$
f''	-	second derivative (Hessian) of f, $f'' : D \to \mathbb{R}^{n\times n}$
$\diamond f(\tilde{D})$	-	range of the function f over $\tilde{D} \subseteq D$, i.e. $\diamond f = \{f(x) :\ x \in \tilde{D}\}$
$\square f(\tilde{D})$	-	smallest enclosure of the range $\diamond f(\tilde{D})$ into an interval
$f([x])$	-	natural interval extension of f over the box $[x] \subseteq D$
$F([x])$	-	value of an inclusion function F for f over the box $[x] \subseteq D$

For continuous functions f the values $\diamond f([x])$ and $\square f([x])$ are everywhere equal and we will prefer the notation $\square f([x])$. The concepts of "natural interval extension" and "inclusion function" will be explained later.

Algorithm Notations

Algorithms are expressed in a high-level language in common use. Each algorithm begins with a description of its input and its output, followed by statements (which themselves consist of a sequence of one or more statements). We next give a list of the statements most frequently used in our algorithms.

1. Assignment statement:
   variable = expression
   The expression on the right is evaluated and assigned to the variable on the left.

2. Conditional statement:

   if condition **then**
   Statement_1;
   Statement_2;
   ...
   else
   Statement_1';
   Statement_2';
   ...
   end if

   The condition is evaluated, and the statements following **then** are executed if the value of the condition is **true**. The **else** part is optional; it is executed if the condition is **false**. In the case of nested conditional statements, we use braces to indicate the **if** statement associated with each **else** statement.

3. Loops:
   We use one of the following three formats:

   for *variable* = *start* **to** *end* **do**
   Statement_1;
   Statement_2;
   ...
   end for

   while condition **do**
   Statement_1;
   Statement_2;
   ...
   end while

   repeat
   Statement_1;
   Statement_2;
   ...
   until condition

   The interpretation of the **for** loop is as follows. If the initial value is less than or equal to the final value, the statement following **do** is executed, and the value of the variable is incremented by one. Otherwise, the execution of the loop terminates. The same process is repeated with the new value of the variable, until that value exceeds the final value, in which case the execution of the loop terminates.
   The **while** loop is similar, except that there is no initialization; the condition is tested before each execution of the statement, if the condition is **true**, the statement is executed,

otherwise, the execution of the loop terminates.
The **repeat until** is like **while**, but testing is done after each execution cycle.

4. Return statement:
   return
   return Value_1, ... , Value_n
   This statement causes the execution of the block where it is located to end. It may return a list of values.

5. Function calls:
   function_name(Argument_1, ... , Argument_n)
   (Value_1, ... , Value_m) = **function_name**(Argument_1, ... , Argument_n)
   A function is called by the name, passing the corresponding number of arguments. The number of arguments can be zero. A function may also return a list of values. Then they are assigned in the same order as given in the parenthesis.

1.2 The Problem of Global Optimization

The problem of global nonlinear optimization, that will be considered here, is defined as follows:
Given a continuously differentiable function $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^n$, D open.
Sought is the global minimum

$$f^* = \min_{x \in [x]} f(x) \tag{1.1}$$

for an interval vector $[x] \subseteq D$. Sought is also the set of global minimizers

$$x^* = \{x \in [x] : f(x) = f^*\}$$

or at least one such minimizer. The continuous differentiability of the function f is not required at all places. But we will assume it in general for the sake of simplicity.

The global maximum of the function f is the global minimum of the function $-f$. Therefore global maximization is equivalent to global minimization (1.1). Throughout this work by the optimization we will understand the minimization of function f.

The problems considered in global optimization in general can not be solved by applying local optimization methods, since there can be many minimizers of the given function in the considered domain. Under these conditions the global minimizers can be found only with considerable computational expenses.

Finding the global minima is much more expensive than the local minima. There is no local criterion qualifying a found minimum as local or global. Lots of techniques exist for solving the problem of global optimization. We give an overview over the classical methods in the next section. An overview of interval methods, which is the subject of this work, starts in Section 1.2.2.

1.2.1 Classical Methods

The first collections of classical techniques for global optimization can be found in [11, 12]. Since that time there is a goal to develop more efficient methods. New developments in this field are described in [24, 49, 50, 54], for example. Usually one distinguishes stochastic and deterministic methods in classical methods for global optimization.

Stochastic methods, that are well described in [49] and [50], randomly select points from $[x]^0$ in order to find the global minimum. The simplest form of a stochastic method (pure random search) selects a certain number of points. The point with the smallest function value is taken as the global minimizer. An advanced modification of this method is the multi start-method. In this method a local optimization is performed from every selected point. For efficiency reasons one tries to apply the local optimization only for certain of the selected points. In this method the starting points will be determined from the analysis of the considered points and their function values. In this way the application of the method of local optimization is limited to the promising points. Often after completion of the method, one can compute the probability, that the real global minimizer was actually found.

The deterministic methods, which are well described in [24], do not use probability quantities. Using exhaustive search these methods try to guarantee that an approximation to the global minimizer is found. This is possible only for a limited class of functions. For instance, there exist deterministic methods for concave functions over a convex set $[x]$ and for Lipschitz-continuous functions, whose Lipschitz-constants are known or can be estimated. The methods relying on the branch and bound principle also belong to deterministic methods (see [24], chapter 4). Interval methods also belong to this class. Therefore the branch and bound principle will be explained in the next section.

1.2.2 Interval Methods

For interval methods we reformulate the problem of global optimization as follows:
sought is the global minimum

$$f^* = \min_{x \in [x]^0} f(x)$$

and the set

$$x^* = \{x \in [x]^0 : f(x) = f^*\},$$

where $[x]^0 = [x_1^1, x_1^2] \times \ldots \times [x_n^1, x_n^2] \subseteq D$. In the sequel we will not be limited to the search of only one global minimizer, but all such points from $[x]^0$ should be found.

Interval methods belong to the group of deterministic methods (see [20, 45]). They are based on the branch and bound principle. One consecutively decomposes the given problem into small subproblems (branching). The tree of subproblems, resulting from the process, will be cut wherever possible (bounding). In other words, some subproblems will not be further decomposed and will not be handled, because they surely do not lead to the solution. The bounding is important, in order to reduce the expense of the solution of the problem.

Applied to the global optimization problem, the branch and bound principle means that the starting box $[x]^0$, where the global minimizer is sought, will be subsequently divided. Subboxes, that definitely do not contain the global minimizer, should not be handled further. In Figure 1.1 the tree, that represents the branch and bound method, is shown. Among others the branch and bound principle is also applied in interval methods that seek the solution of a system of nonlinear equations (see [20, 34]). The crucial property of the method for verified global optimization is that the function is evaluated not only at discrete points, but an enclosure of the function values of f over the whole subbox is always computed: for a subbox $[x]$ of the starting box $[x]^0$ an interval $F([x])$ such that

$$\diamond f([x]) = \{f(x) : \ x \in [x]\} \subseteq F([x]) \tag{1.2}$$

will be computed. The upper bound $\tilde{f}$, $\tilde{f} \geq f^*$ for the global minimum, that is obtained through evaluating the function f at some point $x \in [x]$, will serve as a criterion for bounding. Boxes $[x]$ with $\inf(F([x])) > \tilde{f}$ definitely do not contain the global minimizer, hence they can be discarded. Should $[x]$ contain the global minimizer x^*, then from (1.2) and the definition of $\tilde{f}$ it follows that

$$\inf(F([x])) \leq f(x^*) = f^* \leq \tilde{f}$$

is valid. This is clarified in Figure 1.2. In this example the upper bound $\tilde{f}$ is set equal to a local minimum. The interval method returns always the enclosure of the global Minimum f^* and subboxes, whose union contains all the global minimizers of f in $[x]^0$.

In many classical methods only the information about the function in discrete points is used. One can not be sure, that the solution returned by such a method is really a global minimum. This is caused by the fact that during the execution of the method only a finite number of points will be considered and the global minimizer may leave out this set. An example of such a problem, where the global minimum is left unconsidered by a classical method for global optimization, is a function plotted in Figure 1.3:

$$f(x) = 3 \cdot x^2 - \frac{3 \cdot e^{-(200 \cdot (-x - 0.0675))^2}}{100} + 0.03.$$

A class of such functions is given in [17]. In this example, the enclosures for the range of a function over a given box, which are used in verified method, could be easily computed with the aid of interval arithmetic. Therefore we give a short introduction to interval arithmetic in the next section. In Section 1.2.6 we show how interval arithmetic can be used in order to compute the enclosure of the range of a function.

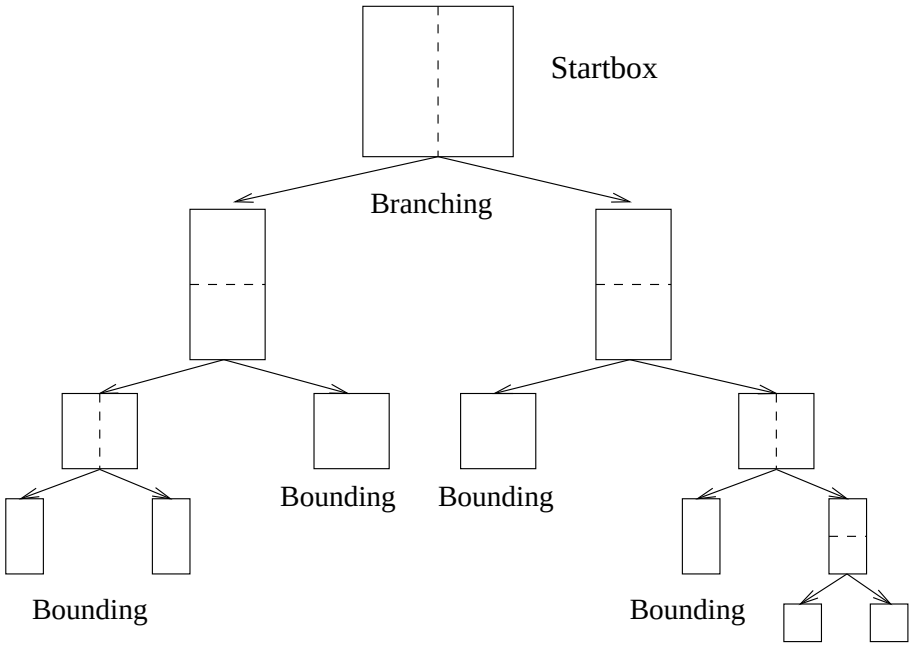


Figure 1.1: Like interval methods for global optimization branch and bound methods construct a search tree through subsequent decomposition of the given problem. The search tree is cut by the bounding strategy

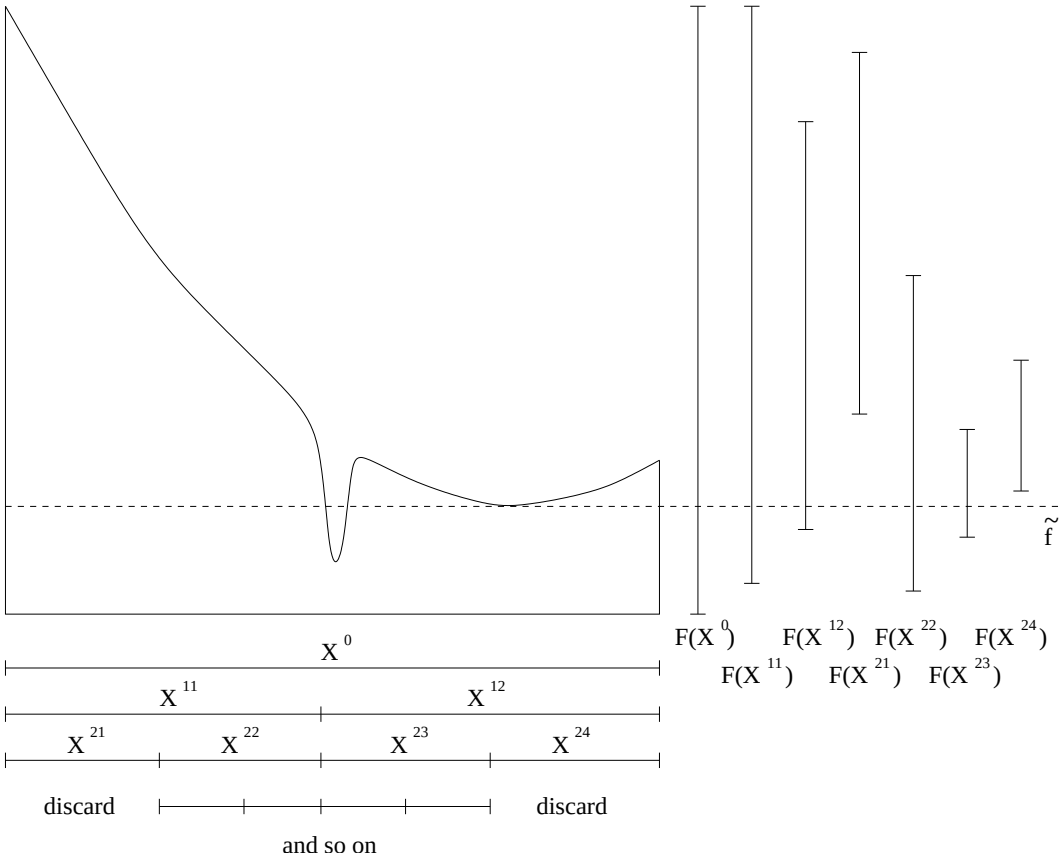


Figure 1.2: Schematic representation of the execution of an interval method for global optimization. Here $\tilde{f}$ does not change, which is not the case in general

1.2.3 Interval Arithmetic

Definitions and statements from interval computations can be found in [4, 39, 41].

Below we give a summary of the main concepts and properties.

Definition 1. *A subset of* $\mathbb{R}$ *of the form*

$$[a^1,\ a^2] = \{t \mid a^1 \leq t \leq a^2,\ a^1, a^2 \in \mathbb{R}\}$$

is called an interval.

Real numbers $a \in \mathbb{R}$ may be considered as special members $[a, a]$ of $\mathbb{IR}$, and they will generally be called point intervals. Interval vectors will also be called n-dimensional boxes.

Definition 2. *Let* $\circ \in \{+, -, \cdot, :\}$ *be a binary operation on the set of real numbers* $\mathbb{R}$*. If* $[a],\ [b] \in \mathbb{IR}$*, then*

$$[a] \circ [b] = \{a \circ b \mid a \in [a],\ b \in [b]\}$$

defines a binary operation on $\mathbb{IR}$*.*

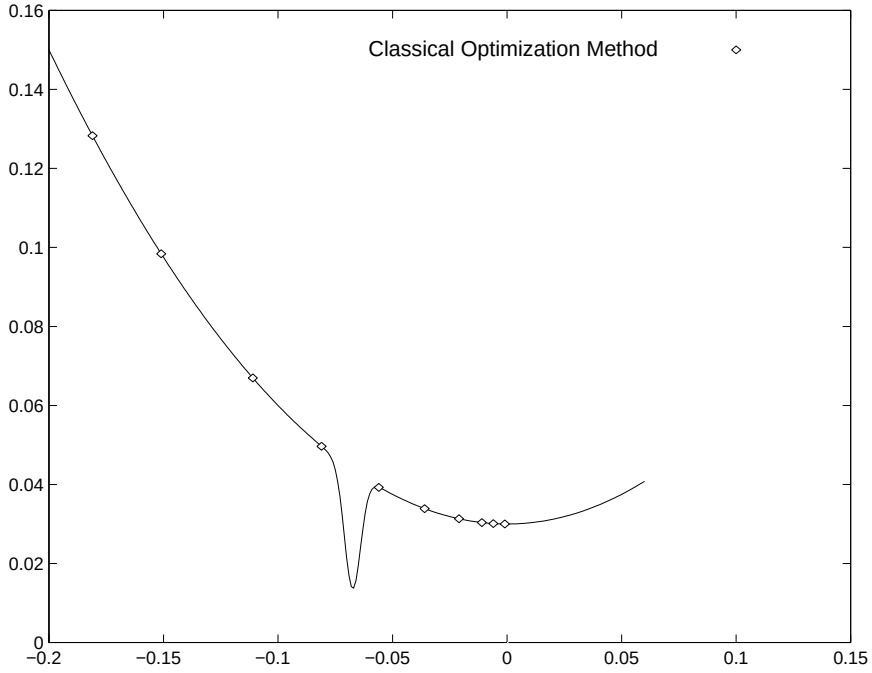


Figure 1.3: An example of an optimization problem, where the classical method will only find a local minimizer

It is assumed that $0 \notin [b]$ in the case of division, and this will not be explicitly mentioned in the sequel. The operations on intervals $[a] = [a^1, a^2]$ and $[b] = [b^1, b^2]$ may be calculated explicitly as

$[a] + [b] = [a^1 + b^1, a^2 + b^2]$,
$[a] - [b] = [a^1 - b^2, a^2 - b^1]$,
$[a] \cdot [b] = [\min\{a^1 \cdot b^1, a^1 \cdot b^2, a^2 \cdot b^1, a^2 \cdot b^2\}, \max\{a^1 \cdot b^1, a^1 \cdot b^2, a^2 \cdot b^1, a^2 \cdot b^2\}]$,
$[a] : [b] = [a^1, a^2] \cdot [1/b^2, 1/b^1]$.

This follows from the fact that $f(x, y)$ with $f(x, y) = x \circ y$, $\circ \in \{+, -, \cdot, :\}$, is a continuous function on a compact set. The function $f(x, y)$ therefore takes a largest and a smallest value as well as all values in between. $[a] \circ [b]$ is therefore again an interval. Notice that subtraction and division are no more the inverse operations for addition and multiplication. This becomes clear from the following examples:

$$[0, 1] - [0, 1] = [-1, 1] \neq 0,$$
$$[1, 2]/[1, 2] = [1/2, 2] \neq 1.$$

Furthermore, for an interval $[a]$ the equality $[a] \cdot [a] = [a]^2$ is no more valid; for $[a] = [-1, 3]$ we have:

$$[a] \cdot [a] = [-1, 3] \cdot [-1, 3] = [-3, 9],$$
$$[a]^2 = [-1, 3]^2 = [0, 9].$$

The explanation to this is that in the interval arithmetic evaluation of an expression several occurrences of the same variable are treated as being independent. For example, for the

computation of $[a]-[a]$ we use

$$[a]-[a]=\{a-b\ :\ a,b\in[a]\}.$$

For multiplication and addition in interval arithmetic the following rules hold

$$\begin{aligned}[a]+[b]&=[b]+[a],\\ [a]+([b]+[c])&=([a]+[b])+[c],\\ [a]\cdot[b]&=[b]\cdot[a],\\ [a]\cdot([b]\cdot[c])&=([a]\cdot[b])\cdot[c].\end{aligned}$$

The distributivity rule is no more valid. It is replaced by the subdistributivity

$$[a]\cdot([b]+[c])\subseteq[a]\cdot[b]+[a]\cdot[c].$$

Proves of all rules given above can be found in [4]. One significant property of interval arithmetic is the inclusion monotonicity of standard operations and elementary functions s (like sin, cos, exp, etc), i.e.

$$\begin{gathered}[a]\subseteq[c],\ [b]\subseteq[d]\ \Rightarrow[a]\circ[b]\subseteq[c]\circ[d],\\ [a]\subseteq[c]\Rightarrow s([a])\subseteq s([c])\end{gathered}$$

are valid, where $s([a])$ is the range of the function s over an interval $[a]$.
The intersection, union and set relations for intervals are defined as for sets:

$$\begin{aligned}[a]\cap[b]&:=\{c\ :\ c\in[a]\wedge c\in[b]\}\\ [a]\cup[b]&:=\{c\ :\ c\in[a]\vee c\in[b]\}\\ [a]=[b]&\Leftrightarrow a^1=b^1\wedge a^2=b^2\\ [a]\subseteq[b]&\Leftrightarrow a^1\geq b^1\wedge a^2\leq b^2.\end{aligned}$$

An intersection is said to be empty if the intersection is empty at least for one component. The union of intervals is not necessarily an interval.
Further functions on intervals, that will be used later, are

$$\begin{array}{rll}m([a]) & :=(a^1+a^2)/2 & \text{(the midpoint of [a]),}\\ d([a]) & :=a^2-a^1 & \text{(the diameter of [a]),}\\ |[a]| & :=\max\{|a|\ :\ a\in[a]\} & \text{(the absolute value of [a]),}\\ q([a],[b]) & :=\max\{|a^1-b^1|,|a^2-b^2|\} & \text{(the distance between } [a] \text{ and } [b]).\end{array}$$

For interval vectors and interval matrices these definitions are applied componentwise. In this work the following rules will be used. One can find them in [4, 41]. Here $[a],[b]\in\mathbb{IR}$.

$$\begin{array}{ll}R1 & d([a]+[b])=d([a])+d([b]);\\ R2 & |[a]\cdot[b]|=|[a]|\cdot|[b]|;\\ R3 & [b] \text{ symmetric, i.e. } b^1=-b^2\Rightarrow d([a]\cdot[b])=|[a]|\cdot d([b]);\\ R4 & 0\in[a],0\in[b]\Rightarrow d([a]\cdot[b])\leq d([a])\cdot d([b]).\end{array}$$

The intersection, union and set relations of interval vectors and interval matrices are defined componentwise.
For an arbitrary set D by $I(D)$ we denote the smallest interval enclosing the set D.

1.2.4 Machine Arithmetic

We now turn to the realization of interval operations on a digital computer. As is well known, computers have only a finite set of numbers that are often represented in a semilogarithmic manner as fixed length floating point numbers (see [4])

$$x = m \cdot b^e.$$

Here m is the mantissa, b the base, and e the exponent. The numbers are normally represented internally with a base $b = 2$ and a normalized mantissa, that is $\frac{1}{2} \leq |m| < 1$. The set of machine numbers of the above type is denoted by $\mathbb{R}_{\mathbb{M}}$. In actual numerical computation using digital computers the user is bedeviled by errors which arise in various fashions including the following:

Errors in data. In the context of formula evaluation and differentiation, this means that the values of the constants, parameters, and variables may not be known or represented exactly, so that one has to compute with approximations to them.

Roundoff error. Computer arithmetic is not carried out exactly, even on exactly known arguments, so that most calculations of the results of arithmetic operations and library functions will be mathematically inaccurate, since all results are required to be expressed as a number in $\mathbb{R}_{\mathbb{M}}$. The actual computation thus results in an approximation to the true result of the transformation or operation being performed, and the difference between the result obtained and the true result is called the roundoff error of the computation.

Truncation error. Many mathematical quantities, such as integrals, derivatives, and various algebraic and transcendental functions, are actually defined only as the limits of infinite sequences of operations. Unless, as in the case of differentiation of simple functions, rules are available which give the values of these limits explicitly, their computation can only be carried out to a finite point of the sequence defining their values. The resulting error of the approximate result obtained by actual computation is called the truncation error.

Ordinarily, a result produced by execution of a computer program will be contaminated by one or more of the above types of error. Furthermore, it is not possible to draw a clear line of distinction between these types of error, nor to eliminate them completely. For example, numbers as $1/3$ or π which appear in a formula may be impossible to be represented exactly as numbers in the system used by a given type of computer. The resulting error in the computation of $\sin(x)$ by a library subroutine in the computation of the transcendental sine function is due to the approximation of the function by a polynomial. Whatever the source, error is pervasive in digital computation, and the accuracy of the result of hundreds or thousands of inexact computations on inaccurate data is always questionable without some additional investigation. The problem of the analysis of the error of numerical results is one which is fundamental to statements about the reliability of the computation.

We denote by $[a]_M \in \mathbb{R}_M$ the smallest machine interval that encloses $[a] \in \mathbb{IR}$. Let $[a], [b] \in \mathbb{IR}$ and $[a]_M, [b]_M \in \mathbb{R}_M$ the corresponding machine intervals, $\circ \in \{+, -, \cdot, /\}$. Then

$$[a] \circ [b] \subseteq [a]_M \circ [b]_M$$

is valid due to the inclusion isotony. The interval $[a]_M \circ [b]_M$ is not in general a machine interval. On a computer it will be approximated by a new machine interval $([a]_M \circ [b]_M)_M$. This is done

by the outer rounding in the computation of $[a]_M \circ [b]_M$, i.e. the lower bound of the resulting interval is rounded down and the upper bound is rounded up to the closest machine number. The consequence of the outer rounding is that after several cosequitive standard operations and an application of elementary functions an interval can become relatively large. Hence we cannot use the theory as it is on a computer when working with small quantities. Dispite this drawback we should use machine interval arithmetic. Because this way we always enclose the real result.

1.2.5 Inclusion Functions

An inclusion function is defined in [32] as

Definition 3. *Let $f : D \to \mathbb{R}$ be a given function, $[x]^0 \in \mathbb{IR}^n$ with $[x]^0 \subseteq D$, where $D \subseteq \mathbb{R}^n$ is open. Then a function F that maps every box $[x] \in [x]^0$ to an interval $F([x]) \in \mathbb{IR}$ so that $F([x])$ contains $\diamond f([x])$, the range of f over $[x]$, will be called an inclusion function of f on $[x]^0$.*

It follows that

$$\diamond f([x]) \subseteq F([x]) \quad \text{for all } [x] \subseteq I([x]^0).$$

Definition 4. *The inclusion function F of f over $[x]^0$ is called inclusion monotone if for $[x], [y] \subseteq I([x]^0)$:*

$$[x] \subseteq [y] \quad \Rightarrow \quad F([x]) \subseteq F([y]).$$

In this work we only consider inclusion functions which are inclusion monotone. One way to obtain an inclusion function is through the natural interval evaluation as we explain now.

1.2.6 The Natural Interval Evaluation

The easiest way to get an inclusion function F is to use interval quantities instead of real quantities in variables that appear in an expression of the function f. The inclusion function F obtained this way is called the natural interval extension of f, i.e. we set $F([x]) := f([x])$. The fact that the resulting function F is really an inclusion function and that F is inclusion monotone follows from the inclusion isotony of interval arithmetic.

For clarity we show this in an example:
Let

$$f(x_1, x_2) = x_1^2 - x_1 + x_2,$$

then

$$F([x]_1, [x]_2) = [x]_1^2 - [x]_1 + [x]_2$$

is the natural interval extension for f. If we choose, for instance, $[x] = ([1,2],[3,4])^T$, then we have

$$\begin{aligned} \diamond f([1,2],[3,4]) &= \{f(x_1,x_2) \ : \ x_1 \in [1,2], x_2 \in [3,4]\} \\ &= [3,6], \\ F([1,2],[3,4]) &= [1,2]^2 - [1,2] + [3,4] \\ &= [2,7] \\ &\supseteq [3,6]. \end{aligned} \tag{1.3}$$

Notice that there could be many mathematically equivalent expressions for the function f. The corresponding interval extensions usually give different results. For the considered function an alternative expression is

$$f(x_1, x_2) = x_1^2 - x_1 + x_2 = x_1 \cdot (x_1 - 1) + x_2.$$

If the interval vector $([-1, 2], [0, 0])^T$ is chosen, one can see that the two interval extensions return different results:

$$\begin{aligned} [x]_1^2 - [x]_1 + [x]_2 &= [-1, 2]^2 - [-1, 2] + [0, 0] \\ &= [0, 4] - [-1, 2] + [0, 0] = [-2, 5], \\ [x]_1 \cdot ([x]_1 - 1) + [x]_2 &= [-1, 2] \cdot ([-1, 2] - 1) + [0, 0] \\ &= [-1, 2] \cdot [-2, 1] = [-4, 2]. \end{aligned}$$

One should choose the expression of a function in such a way that the interval extension encloses the range as closely as possible. If in the expression every variable appears only once, then the enclosure is equal to the range (see [4, 39, 41]). We do not know any unified method allowing to determine which expression should be preferred to another.

1.2.7 The Mean Value and Taylor Forms

As we can see from (1.3) the real range of a function over an interval is usually overestimated by the enclosure returned by the natural interval extension. This overestimation is often substantial. There exist possibilities, other than the natural interval extension, to construct an inclusion function for f. For instance, one possibility is through the use of Taylor series.

Let $[x] \subseteq I(D)$, $c \in [x]$, and f be differentiable. Then for all $x \in [x]$

$$f(x) = f(c) + (x - c)^T \cdot f'(\xi) \qquad \text{for one } \xi \in [x]$$

and, if f is twice differentiable,

$$f(x) = f(c) + (x - c)^T \cdot f'(c) + \frac{1}{2}(x - c)^T \cdot f''(\xi) \cdot (x - c) \quad \text{for some } \xi \in [x].$$

If we denote by F' and F'' inclusion functions for the gradient f' and the Hessian f'' of f, respectively, that could be, for instance, obtained from the natural interval extension, then from the inclusion isotony it follows that

$$\begin{aligned} f([x]) &\subseteq T_1([x], c) := f(c) + (x - c)^T \cdot F'([x]), \\ f([x]) &\subseteq T_2([x], c) := f(c) + (x - c)^T \cdot f'(c) + \frac{1}{2}([x] - c)^T \cdot F''([x]) \cdot ([x] - c) \end{aligned}$$

for all $[x] \subseteq I(D)$, $c \in [x]$, i.e. T_1 and T_2 are inclusion functions for f. We will call T_1 the mean value form, since the expression is constructed from the mean value formula, and T_2 the second order Taylor form for f. The point c can be chosen as an arbitrary point from the box $[x]$. Usually one takes the midpoint $c = m([x])$.

The application of the inclusion functions introduced above is more expensive per evaluation than the application of the natural interval extension. Therefore it is useful only if it leads to narrower enclosures. Below we show that one can expect narrower enclosures for the range using the mean value or the Taylor form instead of the natural interval extension, at least for narrow intervals.

1.2.8 The Order of the Inclusion Function

In order to be able to compare inclusion functions the concept of the order is introduced:

Definition 5. *Let F denote an inclusion function of $f : \mathbb{R}^n \to \mathbb{R}$. If there is a constant K independent of the box $[x] \in \mathbb{R}^n$ such that*

$$d(F([x])) - d(\square f([x])) \leq K \cdot d([x])^\alpha$$

for all boxes $[x]$ with $d([x])$ sufficiently small and fixed $\alpha > 0$, then we say that F is an order α inclusion function for f. When α is 1 or 2, we call the inclusion first order or second order, respectively.

Because of $\square f([x]) \subseteq F([x])$, the order gives the rate of how fast the inclusion function converges to the smallest enclosure of the range of the given function. One can show:

- If f has an expression and F is the natural interval extension of this expression, then F is of order 1.
- If f is differentiable and the inclusion function F' for f' is Lipschitz continuous, i.e. there exists a positive $c \in \mathbb{R}$ such that

  $$d(F'([y])) \leq c \cdot d([y]) \quad \text{for all } [y] \in I(D),$$

  then the mean value form is of order 2 (see [30]).
- If f is twice differentiable and the inclusion function F'' for f'' is bounded, then the second order Taylor form is of order 2 (see [44]);

An introductory treatment of different inclusion functions can be found in [44]. Our above statements make it clear that especially for small boxes one can expect better enclosures from the mean value form or the Taylor form than from the natural interval extension, because they have higher order. But their computation is expensive due to the computation of the gradient or the gradient and the Hessian.

1.3 Differentiation

As one can see from the mean value or the Taylor form, in some cases we need to compute the gradient and the Hessian of f or their enclosures. Therefore we should find methods to compute derivatives and their enclosures.

There are at least three possibilities to realize the computation of derivatives. In numerical differentiation an approximation to the derivative is computed through finite differences. In symbolic differentiation, through the application of differentiation rules to a function's expression, one obtains an expression for the derivative. This is then used for the computation of the derivative at different points. In automatic differentiation, the two steps of the symbolic differentiation are combined. One uses differentiation rules to compute the value of the derivative. To this purpose, one substitutes variables by the corresponding values, so that one does not get a symbolic expression, but the value for the derivative. During the computations one works with numbers, not symbols.

Numerical differentiation is not suitable in our context, since it does not yield a possibility to compute an enclosure of the derivatives over an interval vector.

With the use of symbolic differentiation one can enclose the derivative over a box $[x] \in \mathbb{IR}^n$, by substituting the point $x \in \mathbb{R}^n$ by the box $[x]$ in the expression resulting from symbolic differentiation (natural interval extension). However, one may get very complicated expressions. Those are not efficient to use and may lead to large overestimations of the range.

Within this work symbolic differentiation has been used. First the code list for the given function was generated. Then code lists for derivatives were generated using a special package (we have implemented the package similar to the one described in [32]). Finally all code lists were translated to C-XSC standard expressions. One can find an in depth introduction to the technique we used in [32]. Our symbolic differentiation is about twice faster than the automatic differentiation of the CToolbox (see [13]). Therefore in the implementation of our algorithms we use this symbolic differentiation. For all objective functions used in our work we obtained the same enclosures over starting boxes as by automatic differentiation of the CToolbox.

1.4 The Programming Language in Use

For implementation of the following methods the programming language C++ was used together with the C-XSC library (see [33]), which is developed especially for scientific computations. The library C-XSC has been chosen for implementation since there exists a data type for intervals with corresponding machine arithmetic. This library also allows good handling of real and interval constants, since they can be directly substituted by the smallest enclosing machine interval. This is important, for instance, if constants like 0.1 appear, that have no binary code representation as a machine number. For the implementation of the communication in the parallel method we used MPI (see [36]).

Chapter 2

The Serial Method for Verified Global Optimization

For interval vectors $[x] \in \mathbb{IR}^n$ we define a new diameter $d_m([x])$ as

$$d_m([x]) := \max_{i=1}^{n} d([x]_i).$$

In other words it is the length of the longest edge of the box $[x]$.

As a rule all classical methods for global optimization are very computation consuming. Additionally a problem arises, whether a global or a relative minimum is found as a solution. The absence of guarantee for the correctness of the solution implies that a method for verified global optimization is a right alternative even though it might be even more time consuming.

The development of interval methods for global optimization began in 1966 with an algorithm of Moore [37], that was later improved by Skelboe [51]. These algorithms aim only on determining the global minimum f^*. Methods that also determine global minimizers, were developed by Ichida and Fujii [25] and by Hansen [18, 19]. One can find the description of these methods in [45]. In [20] interval methods for global optimization are described at an introductory level.

As before we always assume that $f : D \to \mathbb{R}$, $D \subseteq \mathbb{R}^n$ open, is a continuously differentiable function. And F will be an inclusion function for f over $[x]^0$. We point out that when the evaluation of the function f at a point $x \in \mathbb{R}^n$ is required, we will write $F(x)$ and consider x as a point interval vector. We do this because by using machine arithmetic instead of exact arithmetic we obtain that $F(x)$ contains the correct result $f(x)$, thus guaranteeing the correctness of the method. The usual evaluation of $f(x)$ could lead to wrong results due to rounding errors.

Existing methods are implemented upon a basic principle, that is described in Algorithm 2.1. As input the algorithm takes a box $[x]^0$ wherein global minimizers are sought, an inclusion monotonic inclusion function F for f over $[x]^0$. The algorithm returns an enclosure for the global minimum and a list of boxes which contain all the global minimizers. The algorithm works by the branch and bound principle. Branching is the subdivision of a box into several

Algorithm 2.1 The Basic Serial Method for Verified Global Optimization

Input: the starting box $[x]^0 \in \mathbb{IR}^n$, the inclusion function F;
Output: an enclosure $[f] \in \mathbb{IR}$ for the global minimum, the list of boxes $\tilde{L}$ which contain all the minimizers;

```
f̃ = sup(F(m([x]^0)));
Initialize the working list L := {([x]^0, inf(F([x]^0)))};
Initialize the solution list L̃ := ∅;
while L ≠ ∅ do
  choose ([x], inf(F([x]))) ∈ L;
  subdivide the box [x] into boxes [x]^1, ..., [x]^k, k ≥ 2; {Branching}
  for i = 1 to k do
    if inf(F([x]^i)) ≤ f̃ then {Bounding}
      f̃ := min{f̃, sup(F([x]^i))};
      if d(F([x]^i)) < ε then
        put ([x]^i, inf(F([x]^i))) into L̃;
      else
        put ([x]^i, inf(F([x]^i))) into L;
      end if
    end if
  end for
  L = cut-off(L, f̃); {Bounding}
end while
L̃ = cut-off(L̃, f̃);
select the pair ([x], inf(F([x]))) ∈ L̃ with inf(F([x])) minimal;
[f] = [inf(F([x])), f̃];
return [f], L̃;
```

subboxes and the search of global minimizers there. Bounding is the deletion of boxes which so far surely cannot contain a global minimizer. This is done by comparison of the lower bound $\underline{f}$ of the range of the function over that box with the upper bound $\tilde{f}$. If $\underline{f} > \tilde{f}$ then the box definitely does not contain any global minimizer and can be discarded, since the value of the function at any point from this box is already greater than the verified upper bound for the global minimum. In the algorithm first we initialize the verified upper bound for the global minimum by $\sup(F([x]^0)$. Then we initialize the working list L with a pair $([x]^0, \inf(F([x]^0)))$ and the solution list S with an empty list. Then we take a pair $([x], \inf(F([x])))$ from L. We will give strategies for the selection of a pair from the list in subsection 2.1. We subdivide the box $[x]$ into k subboxes $[x]^1, \ldots, [x]^k$. Strategies for the subdivision we will give in subsection 2.3. For each $[x]^i$ we compute $F([x]^i)$. Now if $\inf(F([x]^i)) < \tilde{f}$, then we insert the pair $([x]^i, \inf(F([x]^i)))$ either into the solution list $\tilde{L}$ or the working list L, depending on a criterion (see below), else we can discard this pair. As a criterion for the insertion of the pair $([x], \inf(F([x])))$ into the solution list $\tilde{L}$ we use $d(F([x])) < \varepsilon$, where ε is a given accuracy. If $\sup(F([x]^i))$ is less than $\tilde{f}$, then we assign it to $\tilde{f}$. After all boxes $[x]^1, \ldots, [x]^k$ are handled we perform bounding, i.e. the cut-off test. The cut-off test merely discards all elements $([x], \inf(F([x])))$ with $\inf(F([x])) > \tilde{f}$ from the list. The description of the cut-off test is given in Algorithm 2.2. As long as the list L is not empty we take the next pair and do the same what we have done with the pair $([x], \inf(F([x])))$. At the end before returning the solution list $\tilde{L}$ we perform cut-off test for $\tilde{f}$ with the latest value of $\tilde{f}$.

Algorithm 2.2 cut-off

Input: the list L and the verified upper bound $\tilde{f}$;
Output: the list L without elements $([x], \inf(F([x])))$ for which $\inf(F([x])) > \tilde{f}$;

```
for i = 1 to #L do
  if inf(F[x]^i) > f̃ then
    discard pair ([x]^i, inf(F([x]^i))) from the list L;
  end if
end for
return L;
```

2.1 Strategies for the Box Selection

The choice of the box to be handled next influences the order of the execution of Algorithm 2.1 definitely. In the literature one distinguishes the following criteria:

- *oldest-first*-strategy: to choose the oldest box, i.e. L is handled as a queue;
- *best-first*-strategy: to choose the box with the smallest corresponding lower bound, i.e. L is handled as a sorted list;
- *depth-first*-strategy: to choose one of the recently created boxes, i.e. L is handled as a stack.

The *best-first*-strategy was used in [8, 21] as *breadth-first*-strategy. The *oldest-first*-strategy was used in [18, 19]. Examples of algorithms with *best-first*-strategy can be found in [20, 25, 38, 51]. In [8, 21] the *depth-first*-strategy was used. In [27, 28, 29] one can see a combination of *oldest-first*- and *best-first*-strategies. A short introduction to selection strategies can be found in [46].

2.1.1 *oldest-first*-Strategy

In the *oldest-first*-strategy the box, that is longest in the list, is selected. One can consider it as a FIFO (first in first out) structure, i.e. L is a queue. Always the box from the head is selected in order to be handled. Subboxes, resulting from the handling, are inserted to the tail.

The advantage of this strategy is that all boxes are regularly subdivided. Therefore one can expect that the range of the function f over all boxes is always well enclosed. If $\tilde{f} \neq f^*$ is valid then this strategy tends to subdivide those boxes again, which with other list management would have been discarded without further subdivision through the cut-off test when $\tilde{f}$ was sufficiently decreased. In other words during other list managements the best upper bound $\tilde{f}$ tends to f^* faster than by this one.

2.1.2 *best-first*-Strategy

If one chooses as the next box the element $([x], lbf)$, that has the minimal lower bound lbf for the range of f over the box $[x]$ then the strategy is called *best-first*-strategy. In this way the most promising box will always be selected as the next box. The important point in this strategy is that basically no box is subdivided unnecessarily as shown in the statement

below. Here we assume that the criterion for insertion of a box $[x]$ into the solution list $\tilde{L}$ is $d(F([x])) \leq \varepsilon$.

Statement 1. *Using Algorithm 2.1 with the best-first-strategy no pair* $([x], lbf)$ *with* $lbf > f^* + \varepsilon$ *will be chosen for subdivision.*

Proof. See Statement 2.1 in [5]. □

Here, ε is the parameter which was chosen for the criterion to insert boxes into the solution list $\tilde{L}$. It is important to note that any strategy for choosing the next box will have to investigate pairs $([x], lbf)$ with $lbf \leq f^*$ and $[x]$ not fulfilling the termination criterion $d(F([x])) \leq \varepsilon$. For these boxes a decision, if they contain global minimum points or not is not possible yet, even if the global minimum f^* is already known. Only pairs $([x], lbf)$ with $f^* < lbf < f^* + \varepsilon$, therefore, might be considered unnecessarily with the *best-first*-strategy, but the number of these boxes is small in practice. Using the other two strategies, normally many pairs $([x], lbf)$ with $lbf > f^* + \varepsilon$ are investigated, so that it is favorable to use the *best-first*-strategy. This was always our choice in our algorithms.

2.1.3 *depth-first*-Strategy

In the *depth-first*-strategy one manages boxes in a LIFO (last in first out) structure. The natural structure for box storage is thus a stack. For proceeding one always takes the first box from the stack and places new subboxes into the stack. In this way the box will be subdivided further and further as long as it does not fulfill the termination criterion.

One advantage of this strategy is that the maximal length of the list of boxes is usually less than in other strategies. There are always a few small boxes in the list.

In this strategy boxes could be subdivided unnecessarily, though. This happens, for instance, if we insert always the left part of the box (by bisection, i.e. $k = 2$) first, and the right part second, and the global minimizer is in the right part of the initial box. All the boxes in the left part would be subdivided unnecessarily.

2.2 Comparison of Box Selection Strategies

For the comparison of the selection strategies it is important to emphasize that all handled boxes come from the tree resulting from subdividing the initial box $[x]^0$. The tree does not depend on the selection strategy. It depends only on how the subdivision is done. If the box is divided along the longest edge, then one gets the tree presented in Figure 2.1. For the leaves of this tree the termination criterion $d(F([x]))$ is fulfilled.

For all selection strategies boxes from this tree will be handled. Only the set of subboxes to be handled and the order of handling differ. As an example, in Figure 2.1 we used gray boxes to denote those which are handled using the *depth-first*-strategy. The rest of the boxes was discarded through the cut-off test and won't be handled. Using another selection strategy usually the order of boxes to be handled is different, yielding different values for $\tilde{f}$, so that a different set of boxes will be handled.

The *best-first*-strategy is the only strategy out of the three considered here, that will handle only a small amount of boxes with $\inf(F([x])) \leq f^* + \varepsilon$, that were unnecessary. For all strategies this is only so if $\tilde{f}$ is very close to f^* from the beginning, see [5], statement 2.2.

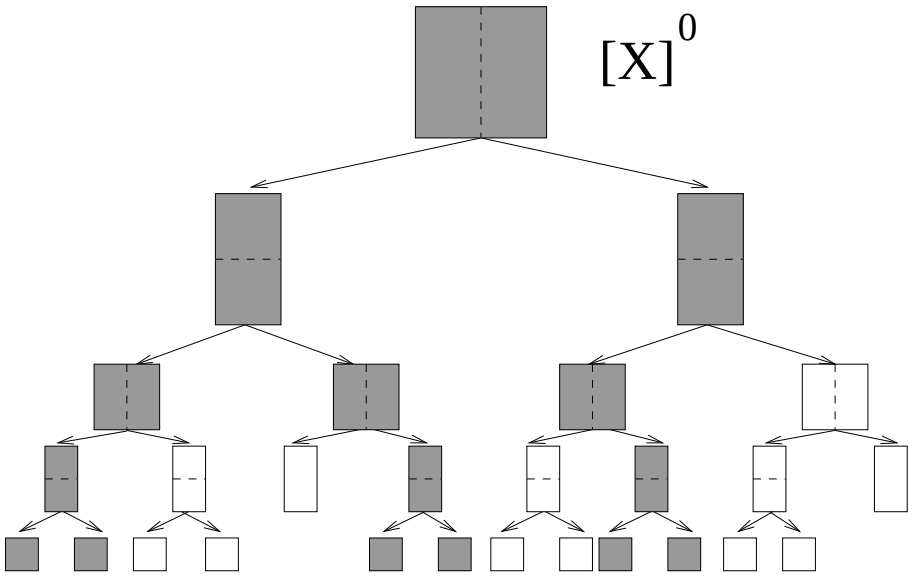


Figure 2.1: Tree resulting from the successive subdivision of the starting box $[x]^0$

Along with the number of boxes that are handled during the execution of the method, also the cost for the box management should be taken into account. They both are decisive for the efficiency of the method.

In the *best-first*-strategy one can store boxes, for instance, in a sorted list. In this case the insertion of new boxes is relatively expensive, but the cut-off test is efficient. It is not necessary to go through the whole list. It is sufficient to find the point from where the rest of the list can be discarded. If one does not store boxes in a sorted list, then the cost of insertion is small. But new expenses appear: in each step the whole list must be searched in order to perform the cut-off test, and also to determine the element with the smallest lower bound. Therefore, in the implementation of the method we use the sorted list.

In the *oldest-first*-strategy boxes should be stored in a queue. In the *depth-first*-strategy boxes should be stored in a stack. In both, stack and queue the insertion and retrieval are performed without a large effort. But for the cut-off test the whole stack or the whole queue must be searched.

2.3 Strategies for the Box Subdivision

We now turn to discuss different strategies for the box subdivision in Algorithm 2.1. In this section we will present some possibilities for how boxes could be subdivided in order to obtain smaller enclosures for the range of the function and therefore for the global minima.

2.3.1 Bisection Strategies

In most of the literature on interval methods for global optimization the initial box is successively *bisected*. The direction orthogonal to which the box is to be bisected can be determined by different criteria. Untill now the most used criterion is the direction with the largest interval

diameter. In [46] it is shown that this is by far not the most efficient way and other subdivision strategies are introduced.

We introduce the new measure - *generalized diameter* - $d_g([x]) = (d_1([x]), \ldots, d_n([x]))^T$, where $d_i([x]) \in \mathbb{R}$, $d_i \geq 0$. Using the generalized diameter we bisect the box $[x]$ in the direction k for which $(d_g)_k = \max_i (d_g)_i$. Algorithm 2.3 describes how the bisection can be done. The

Algorithm 2.3 bisection

Input: the box $[x]$;
Output: subboxes $[x]^1$ and $[x]^2$;

$k = \min\{j \in \{1, \ldots, n\} \ : \ ((d_g)_j([x]) = \max_{i=1}^n (d_g)_i([x])\}$; {determine the direction k orthogonally to which the box is to be bisected}
$[x]^1 = [x]$; {bisect the box $[x]$}
$[x]^2 = [x]$;
$[x]^1_k = [\inf([x]_k), m([x]_k)]$;
$[x]^2_k = [m([x]_k), \sup([x]_k)]$;
return $[x]^1, [x]^2$;

evaluation of the direction, i.e. the evaluation of $(d_g)_i([x])$ could be performed in different ways. Below we give some bisection strategies. Here again F' is the inclusion function for the gradient of f.

Strategy A: This is the strategy proposed first, historically. It is until now mostly used in the literature, see [18, 19, 45]. One defines

$$(d_g)_i([x]) = d([x]_i).$$

In other words, one always bisects orthogonally to the direction of the longest edge (the longest interval diameter). This strategy is based on the idea that such a regular bisection leads to boxes looking like a cube and the diameter of the boxes possibly tends to zero fast. In this way one gets a good enclosure of the global minimizer and hopes to get a good enclosure for the range of the function over the box as well. For this strategy there exists a short theoretical investigation for its convergence and its convergence speed in the Moore-Skelboe-Algorithm in [5], page 27.

Strategy B: Hansen in [20] used another strategy. He sets the goal to bisect orthogonally to the direction in which the function varies most. One defines

$$f_i(t) = f(x_1, \ldots, x_{i-1}, t, x_{i+1}, \ldots, x_n), \ i = 1, \ldots, n,$$

where $x_j = m([x_j])$. Then

$$w_i([x]) = \max_{t \in [x]_i} f_i(t) - \min_{t \in [x]_i} f_i(t)$$

represents a measure of how the function varies in i-th variable. It would be desirable to bisect the box orthogonal to the direction i in which w_i is maximal. Since the evaluation of w_i would be expensive, one uses the estimate

$$w_i \leq d(F_i'([x])) \cdot d([x]_i),$$

which immediately follows from the mean value Theorem, see (3.7). Now one sets

$$(d_g)_i([x]) = w_i([x]).$$

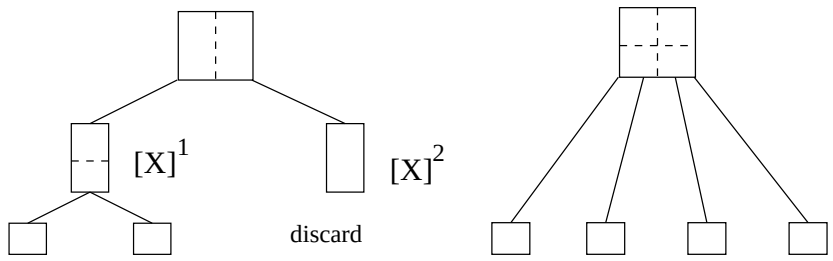
Strategy C: The bisection strategy C like the strategy B uses the gradient over the box $[x]$ in order to determine the direction. Like the strategy B it sets the goal to bisect orthogonally to the direction in which the diameter of $F([x])$ changes most. One now takes

$$(d_g)_i = |F_i'([x])| \cdot d([x]_i).$$

This strategy is already used in the method for enclosing the solution of a system of nonlinear equations in [31]. In the method for verified global optimization it was first used in [47]. This strategy differs from the strategy B in that it uses the absolute value instead of the diameter. And therefore we choose now the direction in which the given function is steepest.

2.3.2 Multisection Strategies

By multisection we understand the subdivision of a box into more than two parts. Again strategies similar to Strategies A - C can be used. The idea of multisection originates from the parallelization of methods for verified global optimization. During the execution of the method it happens that some processors handle boxes that contain no global minimizer and this becomes clear relatively early. Those processors may run out of boxes. At this point it is important to have them busy again as soon as possible. When a processor has boxes not just for itself but also to share, idle processors could be provided with new boxes fast. For this it is efficient to have boxes multisected into more than two parts in each step of the iteration. In [5] tests show that for an algorithm given there it is efficient to multisect boxes into four parts. Also for a serial method given in [9] it turned out to be efficient to multisect boxes into 3 parts. For multisection it is again important to find the appropriate strategy for computing the subdivision directions. If the box is subdivided into many parts, then one can expect to have better enclosures of the range of the function f over those subboxes. By the bisection it



(a) Bisection ($l = 1$) (b) Multisection ($l = 2$)

Figure 2.2: Strategies for the box subdivision

can happen that intermediate boxes, which do not contain a minimizer cannot be discarded because the lower bound for the range of the function over this box is still small. On the other hand, in multisection it can happen that we produce many subboxes (which all have to be worked upon), whereas bisection would already have given two subboxes which both would have been discarded subsequently. In the parallel method in [5] it is proven by numerical experiments that multisection into four parts is the most efficient. In [20] it is shown that for the algorithm for global optimization considered there, it is indeed efficient to subdivide the box into even more parts. There the number of subboxes was $2^3 = 8$ and strategy B for the

direction selection was used. Also in [15] for that parallel method multisection was efficient. There strategy A is used.

The multisection of the box $[x]$ can be done with Algorithm 2.4. One gets the bisection if one chooses $l = 1$.

Algorithm 2.4 multisection

Input: the box $[x]$ and the number of subboxes $k = 2^l$;
Output: subboxes $[x]^1, \ldots, [x]^k$;

compute $(d_g)_i([x])$, $i = 1, \ldots, n$, according to the multisection strategy;
for $i = 1$ to l **do** {determine the directions $r_1, \ldots, r_l$}
  $r_i = \min\{j \in \{1, \ldots, n\} : (d_g)_i([x]) = \max_{i=1}^{n}(d_g)_i([x])\}$;
  $(d_g)_{r_i}([x]) = (d_g)_{r_i}([x])/2$;
end for
list = multisect($[x]$, l, $r_1, \ldots, r_l$) {an auxiliary function}
return list;

Algorithm 2.5 multisect

Input: the box $[x]$, the number of subboxes $k = 2^l$ and directions $r_1, \ldots, r_l$;
Output: subboxes $[x]^1, \ldots, [x]^k$;

bisect $[y]$ orthogonally to the direction r_l:
$[y]^1 = [y]$;
$[y]^2 = [y]$;
$[y]^1_k = [\inf([y]_{r_l}), m([y]_l)]$;
$[y]^1_k = [m([y]_l), \sup([y]_{r_l})]$;
if $l = 1$ **then**
  list = $\{[y]^1 \cup [y]^2\}$;
else
  list = multisect($[y]^1$, $k-1$, $r_1, \ldots, r_{l-1}$) $\cup$ multisect($[y]^2$, $k-1$, $r_1, \ldots, r_{l-1}$);
end if
return list;

2.4 Accelerating Devices

Algorithm 2.1 is widely applicable, since it does not require the differentiability of the considered function. By considering differentiable functions, the process can be accelerated through the insertion of additional tests to discard boxes and of methods to shrink boxes other than by subdivision. As accelerating devices we consider here the monotonicity test, which requires continuous differentiability of a function, and the nonconvexity test as well as the interval Krawczyk method, which require two times continuous differentiability of a function.

All the accelerating devices guarantee that none of the boxes that contain the global minimizer is discarded (see [20, 45]).

2.4.1 The Monotonicity Test

For the monotonicity test, whose description can be found in [20, 45], it is assumed that the given function is continuously differentiable. Let $F' = (F'_1, \dots, F'_n)^T$ be an inclusion function of the gradient $g = (g_1, \dots, g_n)^T$ of the function f.

A global minimum, that is assumed to be in the interior of the starting box $[x]^0$, is always a local minimum at a certain point x^*. If the function f is strongly monotone over the given box $[x] \in I([x]^0)$, i.e. $g_i(x) < 0$ or $g_i(x) > 0$ is valid for some $i \in \{1, \dots, n\}$ and for all $x \in [x]$, then the box $[x]$ does not contain any stationary points and hence neither local nor global minima.

In the monotonicity test we therefore check whether for at least one $i \in \{1, \dots, n\}$

$$0 \notin F'_i([x]).$$

Since F' is the inclusion function for the gradient g it follows that $g_i(x) < 0$ or $g_i(x) > 0$ is valid for all $x \in [x]$. Since we assume that the global minimum is located in the interior of $[x]^0$, the box $[x]$ needs not be considered any more. If only $F'_i([x]) \leq 0$ or $F'_i([x]) \geq 0$ is valid then it is not sufficient to consider just edges, given by

$$([x]_1, \dots, [x]_{i-1}, \sup([x]_i), [x]_{i+1}, \dots, [x]_n)^T,$$
$$([x]_1, \dots, [x]_{i-1}, \inf([x]_i), [x]_{i+1}, \dots, [x]_n)^T,$$

at least, if all global minimizers are sought. The function surely achieves its minimal value at the edges. But the function could be constant parallel to the direction i in the interior of $[x]$ and there would thus be other global minimizers in the interior of $[x]$.

If the global minimum is definitely not in the interior of the starting box $[x]$, but can be located on an edge, then the monotonicity test should be modified. Instead of discarding the box in the case $0 \notin F'_i([x])$ we should check if $F'_i([x]) \leq 0$ or $F'_i([x]) \geq 0$ is valid for $([x]_1, \dots, [x]_{i-1}, \sup([x]_i), [x]_{i+1}, \dots, [x]_n)^T$ or $([x]_1, \dots, [x]_{i-1}, \inf([x]_i), [x]_{i+1}, \dots, [x]_n)^T$, respectively. If it is the case, then the box can be reduced to the corresponding edge, otherwise it can be discarded.

Throughout this work we assume that global minimizers are in the interior of the initial box $[x]^0$.

2.4.2 The Nonconvexity Test and the Interval Krawczyk Method

If the function f is twice differentiable, then the usage of an inclusion function F'' for the Hessian can accelerate the method as we shall explain below. But one should remember that it is usually relatively expensive to compute an enclosure for the Hessian. By application of forward automatic differentiation or symbolic differentiation the cost increases by factor $c \cdot n^2$ (here n is the number of variables, c is a constant depending on the function complexity) compared to the amount of time required to compute the original function value.

In the nonconvexity test and the interval Krawczyk methods, which are given below, it is again assumed that the global minimum is in the interior of the box $[x]^0$.

The Nonconvexity Test

If an enclosure for the Hessian over the box $[x]$ is computed, then it is useful to perform the nonconvexity test (also called concavity test), since it has very low additional cost. From this

test, that is quite well described in [20, 47], we can determine whether the given function is nowhere locally convex over the given box. For a local minimizer $x \in [x]$ the matrix $f''(x)$ is always positive semidefinite. Hence the function is locally convex at the minimum. Therefore a box $[x]$, where f is not convex at each point locally, cannot contain local minimizers and therefore it can be discarded.

We will denote $F''([x])$ by H. In the nonconvexity test we will consider the diagonal elements $H_{ii} = [\underline{h_{ii}}, \overline{h_{ii}}]$ of H. If $\overline{h_{ii}} < 0$ holds for at least one $i \in \{1, \ldots, n\}$, then none of the matrices $(h_{ii})_{i,j=1,\ldots,n} \in H$ with $h_{ij} \in H_{ij}$ is positive semidefinite. Since H is the enclosure for $\diamond f''([x])$, the matrix f'' is nowhere in $[x]$ positive semidefinite. The box $[x]$ needs not be considered further.

The Interval Krawczyk Method

A global minimizer in the interior of $[x]^0$ is always a stationary point x^*, i.e. a point where the gradient f' vanishes, i.e. $f'(x^*) = 0$. Therefore one can try to enclose zeros of f' in $[x]$ using the interval Krawczyk method.

If we denote the gradient by g, $g : [x]^0 \subseteq \mathbb{R}^n \to \mathbb{R}^n$, $g = (g_1, \ldots, g_n)^T$, $g_i : [x]^0 \to \mathbb{R}$, then points $x \in [x]$ with $g(x) = 0$ are sought. For this purpose we can use the interval Krawczyk method (which will be analyzed in detail in Section 3.5). We write down the algorithm for this method in Algorithm 2.6.

Algorithm 2.6 The Interval Krawczyk Method

Input: the box $[x]$, inclusion functions F', F'' for g and f'', respectively, m - number of maximum steps;

Output: the reduced box $[x]$, which contains a zero of g or an empty set;

1: set $[x]^1 = [x]$;
2: **for** $l = 1$ **to** m **do**
3: choose $c \in [x]^l$;
4: compute $Y = \left(m\left(F''\left([x]^l\right)\right)\right)^{-1}$;
5: compute $[y] = F'(c)$;
6: $[k] = c - Y \cdot [y] + \left(I - F''([x]^l)\right) \cdot ([x]^l - c)$;
7: set $[x]^{l+1} = [x]^l \cap [k]$;
8: **if** $[x]^{l+1} = \varnothing$ **then**
9: **return** $\varnothing$;
10: **end if**
11: **end for**
12: **return** $[x]^{m+1}$;

In Algorithm 2.6, line 6 we always get an interval vector, not a set of interval vectors like in the interval Newton method. If in Algorithm 2.6, line 7 we get an empty intersection we terminate the iteration and there is no zero of g in $[x]$.

2.5 A New Serial Method for Verified Global Optimization

In this section we introduce a new serial method for verified global optimization. We will build it up step by step introducing several new techniques. In the next subsection we give a modification of the interval Krawczyk method, which makes a little improvement over the interval Krawczyk method. Then we introduce two new techniques related to the interval methods along with numerical results. And finally in subsection 2.5.5 we give the complete description of the new method.

2.5.1 Modification of the Krawczyk Method

In the interval Krawczyk method, the interval vector $[x]^l$ is used for calculation of all the components of the intermediate interval vector $[k]$ (see Algorithm 2.6, line 6). We modify the interval Krawczyk method by substituting some terms in thhe calculation of the i-th component of the vector k by the updated values: $([x]^{l+1} - c)_j$ instead of $[x]^l - c)_j$ for $1 \leq j < i$. Using updated intervals (sometimes they get narrower) we try to get narrower intervals $[k]^l$ and consequently at the end a narrower resulting interval $[x]^{m+1}$. Algorithm 2.7 states this modification.

Algorithm 2.7 The Modification of the Interval Krawczyk Method

Input: the box $[x]$, inclusion functions F', F'' for g and f'', respectively;
Output: the reduced box $[x]$, which contains a zero of g or an empty set;

set $[x]^1 = [x]$;
for $l = 1$ **to** m **do**
  choose $c^l \in [x]^l$;
  compute $Y = \left(m \left(F''\left([x]^l\right)\right)\right)^{-1}$;
  compute $[y] = g(c)$;
  compute $M = I - Y \cdot F''([x]^l)$;
  for $i = 1$ **to** n **do**
    $[k]_i = c_i - (Y \cdot [y])_i + \left(M \cdot ([x]^l - c)\right)_i$;
    $[x]_i^l = [x]_i^l \cap [k]_i$;
    if $[x]_i^l = \varnothing$ **then**
      return $\varnothing$;
    end if
  end for
  set $[x]^{l+1} = [x]^l$;
end for
return $[x]^{m+1}$;

We measure the run time of the program implementing Algorithm 2.1 using the Krawczyk Interval Method as an accelerating device. After the modification we get time improvements for seven test problems, which we report in Table 2.1. For the rest of the test problems there is no change. The definition of the test functions are given in Appendix A. Although the improvement is not significant, we will use the modified method. From now on we will refer to this modification as the interval Krawczyk method. In order to perform the interval Krawczyk method we evaluate the Hessian of the function f. At this point, before the interval Krawczyk

method, we compute an enclosure of the range of the function using the Taylor form (see 1.2.7), which costs almost nothing, in hope to discard the box (comparing the lower bound of the range of the function with $\tilde{f}$) without performing the expensive interval Krawczyk method.

2.5.2 A New Strategy for Box Processing

After the introduction of the interval methods as accelerating devices the following question arises:
Should we perform multisection after the interval method.
In [5] Berner used both, the interval Newton method and multisection. In [55] Wiethoff used the interval Newton method as an accelerating device and performs multisection only if the Newton method leaves the box unchanged. There are two more possibilities:

- apply multisection if after the interval Krawczyk method at least one component of the box remains unchanged.
- never apply multisection.

The option of *never* processing a box by subdivision is not a viable one. Even when boxes are very small, it can happen that the interval Krawczyk method does not reduce the box in at least one direction, so that we can run into an infinite loop. So we have the following strategies for performing multisection after the interval method:

A if the box remains unchanged;

B if at least one component remains unchanged;

C always.

Table 2.2 represents the time measured using algorithms, that implement strategies A - C formulated above in the second, the third and the forth columns, respectively. In the third and the forth columns, the parenthesis indicate the speedup relatively to the second column. In the last row the average speedup for the third and the forth columns is given. As we see from the table, strategy B, i.e. processing a box by subdivision if at least one component remains unchanged, is the best on average. The slightly longer time for strategy B than strategy A when the numbers of iterations are the same is explained as the time spent for extra subdivisions. Because *the box remains unchanged* implies *at least one component remains unchanged*, all boxes processed by subdivision in the strategy A are also processed by subdivision in the strategy B. But not all boxes processed by subdivision in the strategy B are processed by subdivision in the strategy A.

In some test problems those boxes that are processed by subdivision in strategy B and not in strategy A, yield a smaller upper bound for the global minimum $\tilde{f}$. Owe to them the time for these problems are significantly decreased. But for the rest of the problems execution time increases slightly. After all we should emphasize that if strategy B increases the time, then not significantly, but it can decrease the time substantially (see problems SHCB, L8, L12, L18, H6, GEO3). We will use the strategy B in our algorithm. It is difficult to apply the strategy B to the Newton method, since it can return more than one boxes.

2.5.3 A New Strategy for Applying the Krawczyk Method

Usually interval methods used as accelerating devices are applied once the diameter of the box gets smaller than an a priori set parameter (see [5]). The disadvantage of this approaches is that we have a parameter not depending on the given function. This results in a non-optimal use of the interval method which is quite expensive. Wiethoff in [55] tried to adjust the parameter dynamically. His approach has disadvantages, too. He adjusts the parameter based on the result of the interval method applied to the current box. But the function can behave completely different over the next selected box. And therefore adjusted parameters could be of no use there. Here is a new approach to this problem. When the interval vector $[x]$ is too large and the given function is too complicated, the interval Krawczyk method fails, leaving all components of the interval vector $[x]$ unchanged. If this takes place, we save, together with the box, a counter t, that indicates the number of times the box should now be processed without the interval Krawczyk method. Each time the box is processed, the counter is decremented until it is zero. In this way we try to store the information about the behavior of the given function over the box. If the interval method failed to reduce the box then we postpone the application of the interval method next time when this box is selected as the next one to be handled. In Table 2.3 we give times measured with different values of t compared with the time of the algorithm which processes a box with the standard interval Krawczyk method ($t = 0$). In this table we see that in general the adaptive application of the interval method is better than the usual way of application of the interval method. For simple problems like S5, S7, S10 and RO it works very well with all values of t. A new adaptive approach is good not only for simple problems. We see that it is also good for medium and hard problems like R4, H6, S2.14, GEO1, GEO2, GEO3 and R8 for all three values. And for values 1 and 2 it is efficient for problem S2.14, HM3, HM4. For the case $t = 2$ we have the best average time. But in the final serial algorithm we rather set $t = 1$ because for medium and hard problems it gives better timing (see Table 2.4). To show the efficiency of this technique we also tried out this strategy in an algorithm using the interval Newton method instead of the interval Krawczyk method. In general we got a similar improvement. Results are presented in Table 2.5.

In Table 2.6 we compare algorithms with the new technique and the algorithm by Wiethoff. In perenthesis we give weighted percentage - the percentage of the correcponding method from the total time spent by all methods for a certain problem. The very small time for R8 spent by the Algorithm of Wiethoff is the result of use of the Boxing Method (see [55]). For R8, by chance one of global minimizers is isolated at the beginning and a very good approximation to the global minimum is obtained. But unfortunately for the rest of the problems this method does not work that well.

2.5.4 Dependence of the results on the differentiation technique

The decision of which interval method to use depends on the software for interval arithmetic and on a technique for differentiation. As we have mentioned in the introduction, automatic differentiation of the package C++ Toolbox, which is well explained in [13], it is about twice slower than the symbolic differentiation we have implemented. The interval Krawczyk method always returns at most one box, none in the best case. The interval Newton method can return up to n boxes, where n is the dimension of the box, since in every internal loop division by an interval containing zero can occur. If the dimension is large and the gradient of the function is expensive to evaluate then the insertion of boxes after the interval Newton method becomes expensive. In such cases it is preferable to use the interval Krawczyk method over the interval Newton method. In this subsection we perform all tests on the serial method, but this time we

will use automatic differentiation of the package C++ Toolbox (see [13]). We perform all tests to show that our new techniques work well also with automatic differentiation of the package C++ Toolbox.

In Table 2.7 we show that the modification of the interval Krawczyk method has a little impact to the execution time. Here we see that using the modified method for the problem H6 gets worse. It can be explained as follows. The interval Krawczyk method shrinks the box only a little and the succeeding multisection would be of good use. In Table 2.8 we tested three possible strategies for multisecting the box after the interval Krawzyk method. Like the version without the C++ Toolbox we get better times on average for strategy B - processing the box by subdivision if at least one component of a box remains unaffected.

In Table 2.9 we see times for different values of t. For $t = 1$ we have the best average time, unlike the algorithms which use our symbolic differentiation. But the important point is that the new technique accelerates the average time. The new technique also works for methods using the interval Newton Method. This fact is shown in Table 2.10.

2.5.5 A New Serial Method for Verified Global Optimization

Puttin all previous observations together, we end up with our new serial method for verified global optimization, which we discribe in detail now. The algorithm is divided into two parts: its main part - Algorithm 2.8 and the handling of an element - Algorithm 2.9. It is done in this way since the handleBox routine can also be used in a parallel method. The main part is simple. At the beginning the working list L is initialized with $([x]^0, \inf(F([x]^0), 0)$ and the solution list $\tilde{L}$ with an empty list. Then one element is taken from the list L and processed by handleBox. After that bounding is performed. This continues until L becomes empty. Then the cut-off test is performed for the solution list and an enclosure for the global minimum is set to $[\inf(F([x])), \tilde{f}]$, where $([x], \inf(F([x])))$ is an element in $\tilde{L}$ with $\inf(F([x]))$ minimal. At the end both, $[f]$ and $\tilde{L}$ are returned. The handleBox part is where everything happens.

Algorithm 2.8 A New Serial Method for Verified Global Optimization

Input: the starting box $[x]^0$, inclusion functions F, F' and F'', an upper bound $\tilde{f} = \sup(F(m[x]^0))$ for a global minimum;

Output: an enclosure for the global minimum f^* and the list of boxes $\tilde{L}$ which contain all the global minimizers;

```
initialize the working list L = {([x]^0, inf(F([x]^0))), 0};
initialize the solution list L̃ = ∅;
repeat
   retrieve element ([x], lbf, t) with lbf minimal from the list L;
   (L, L̃, f̃) = handleBox(L, L̃, f̃, ([x], lbf, t));
   L = cut-off(L, f̃);
until L = ∅
L̃ = cut-off(L̃, f̃);
select an element ([x], inf(F([x])), t) ∈ L̃ with inf(F([x])) minimal;
[f] = [inf(F([x])), f̃];
return [f], L̃;
```

In handleBox we perform tests in the increasing order of their computational cost. We compute $F'([x])$ and try to discard the box if the function is monotone over this box. If the function

is not monotone then we have two possibilities. If the counter t is positive or the diameter d_m of the box is greater than an a priori set static parameter ε_{int_meth} then we multisect the box into subboxes $[x]^1, \ldots, [x]^{2^l}$ (for subdivision strategies see Section 2.3). For each of these boxes we evaluate the range of the function and test if the lower bound of this range is greater than $\tilde{f}$, the validated upper bound for the minimum. If it happens that the lower bound of the range is greater than $\tilde{f}$ then the function f definitely has no minimizer in that box and the box can be discarded. Otherwise we insert the box $[x]^l$ either into L or $\tilde{L}$, depending on the termination criterion, with the counter set to $t-1$. As a termination criterion we use the condition $d(F([x])) < \varepsilon_{accept}$, i.e. when the range of the function over this box is small enough. We compare the upper bound of the range with $\tilde{f}$. If the upper bound is less than $\tilde{f}$ we set $\tilde{f}$ to this value. After the insertion of all subboxes we return from the procedure, probably with the updated value of $\tilde{f}$.
But if the counter is zero and the diameter of the box is smaller than the static parameter ε_{int_meth} then we do the following. We evaluate $F''([x])$ ($F([x])$ is evaluated automatically during the evaluation of $F''([x])$). This will be used for the nonconvexity test as well as for the evaluation of the range of the function using the Taylor form and for performing one step of the interval Newton method. Using $F''([x])$ we test if the function is concave over the box. If not we discard the box and exit from the procedure returning L and $\tilde{L}$.
If the box is not yet discarded we find the middle point of the box and compute function and gradient values at that point. Using the Taylor form we evaluate the range of the function and check if its lower bound is greater than $\tilde{f}$. If so we discard the box.
Now we perform one step of the interval Newton method. If the Newton method returns empty intersection we exit the procedure returning L and $\tilde{L}$. If the Newton method does not shrink the box at all, i.e. all components of the box remain unchanged, we multisect the box into subboxes $[x]^1, \ldots, [x]^{2^l}$ and insert them into the working list L with the counter set to t_{multi}. If the Newton method left unchanged at least one component (but not all) we multisect the box into subboxes and insert them into the working list L with the counter set to zero. At the end we return L and $\tilde{L}$ and exit the procedure. We choose the interval Newton method over the interval Krawczyk method bacause based on the comparison given in Table 2.6 it is better in the average.

Problem	Original		Modified	
	Iterations	Time	Iterations	Time
S7	22	1.01	21	**0.99**
S10	22	1.36	21	**1.28**
RO	47	0.29	45	**0.27**
H6	300	28.88	299	**28.24**
GEO1	352	14.51	351	**14.48**
GEO2	423	25.20	422	**23.09**
S2.7	94	32.29	91	**31.57**

Table 2.1: The use of Algorithm 2.7 instead of Algorithm 2.6 gives a little improvement

Algorithm 2.9 handleBox

Input: the working list L, the solution list $\tilde{L}$, the triple $([x], x, t)$, the best upper bound $\tilde{f}$;
Output: the working list L, the solution list $\tilde{L}$, the actual $\tilde{f}$;

perform monotonicity test;
if the box $[x]$ is to be discarded **then**
  return L, $\tilde{L}$, $\tilde{f}$;
end if
if $t = 0$ and $d_m([x]) \leq \varepsilon_{int\_meth}$ **then**
  perform nonconvexity test;
  if the box $[x]$ is to be discarded **then**
    return L, $\tilde{L}$, $\tilde{f}$;
  end if
  $c = m([x])$;
  $[w] = F([x]) \cap (F(c) + ([x] - c)^T F'(c)^T + \frac{1}{2}([x] - c)^T F''([x])([x] - c))$; {Taylor Form}
  if $\inf([w]) > \tilde{f}$ **then**
    return L, $\tilde{L}$, $\tilde{f}$;
  end if
  perform one step of the interval Newton method;
  if the box is $[x]$ discarded because of the empty intersection **then**
    return L, $\tilde{L}$, $\tilde{f}$;
  else if Newton method did not shrink the box $[x]$ at all **then**
    $([x]^1, \ldots, [x]^{2^l}) = \text{multisect}([x], l)$;
    for $i = 1$ **to** 2^l **do**
      $(L, \tilde{L}, \tilde{f}) = \text{insert}(L, \tilde{L}, [x]^i, t_{multi})$;
    end for
  else if Newton method left at least one component unchanged **then**
    $([x]^1, \ldots, [x]^{2^l}) = \text{multisect}([x], l)$;
    for $i = 1$ **to** 2^l **do**
      $(L, \tilde{L}, \tilde{f}) = \text{insert}(L, \tilde{L}, [x]^i, 0)$;
    end for
  end if
else
  $([x]^1, \ldots, [x]^{2^l}) = \text{multisect}([x], l)$;
  for $i = 1$ **to** 2^l **do**
    $(L, \tilde{L}, \tilde{f}) = \text{insert}(L, \tilde{L}, [x]^i, t - 1)$;
  end for
end if
return L, $\tilde{L}$, $\tilde{f}$;

Algorithm 2.10 *insert* - inserts the box into L or $\tilde{L}$ depending on the termination criterion

Input: the working list L, the solution list $\tilde{L}$, the box $[x]$, the counter t and the upper bound $\tilde{f}$;
Output: the working list L, the solution list $\tilde{L}$, the actual $\tilde{f}$;

if $\inf(F([x]) > \tilde{f}$ **then**
  return $L, \tilde{L}, \tilde{f}$;
end if
$\tilde{f} = \min\{\tilde{f}, \sup(F([x])\}$;
if $d(F([x]) \leq \varepsilon_{accept}$ **then**
  insert $([x], \inf(F([x])), t)$ into L;
else
  insert $([x], \inf(F([x])), t)$ into $\tilde{L}$;
end if
return $L, \tilde{L}, \tilde{f}$;

Problem	Whole box		One component			Always		
	Iters	Time	Iters	Time	(%)	Iters	Time	(%)
S5	17	**0.69**	17	**0.69**	**(100)**	23	0.75	(108)
S7	21	**0.99**	21	**0.99**	**(100)**	27	1.07	(108)
S10	21	**1.28**	21	**1.28**	**(100)**	27	1.44	(112)
SHCB	691	5.73	235	**1.52**	**(26)**	271	1.81	(31)
BR	38	**1.09**	38	**1.09**	**(100)**	42	1.44	(132)
RO	46	0.28	45	**0.27**	**(96)**	46	0.29	(103)
L8	35	8.27	12	2.16	(26)	12	**2.13**	**(25)**
L9	14	**2.96**	14	**2.96**	**(100)**	14	3.36	(113)
H3	37	**2.59**	41	2.67	(103)	63	3.60	(138)
G5	31	**2.85**	31	2.88	(101)	31	3.09	(108)
R4	265	11.97	261	**11.68**	**(97)**	293	12.88	(107)
L12	166	67.52	34	**11.81**	**(17)**	33	11.97	(17)
L18	90	25.20	29	**6.75**	**(26)**	28	6.99	(27)
G7	42	5.60	42	**5.57**	**(99)**	42	5.89	(105)
G10	44	**9.12**	44	**9.12**	**(100)**	44	9.87	(108)
GP	7600	90.05	7687	90.03	(99)	7726	**89.63**	**(99)**
H6	872	86.11	299	**28.24**	**(32)**	332	29.63	(34)
S2.14	10219	143.33	5146	**78.93**	**(55)**	5157	80.00	(55)
GEO1	351	**14.48**	351	**14.48**	**(100)**	357	14.70	(101)
GEO2	1987	84.36	422	**23.09**	**(27)**	446	24.94	(29)
GEO3	2018	88.43	507	**29.36**	**(33)**	529	30.56	(34)
JS	504	101.36	216	**45.17**	**(44)**	253	52.00	(51)
S2.7	197	51.91	91	**31.57**	**(60)**	124	41.20	(79)
L3	543	**130.21**	543	130.43	(100)	693	159.57	(122)
R8	1145	217.73	1145	220.32	(101)	1145	**217.57**	**(99)**
HM3	754	**232.13**	754	233.28	(100)	809	244.77	(105)
HM4	3716	1820.20	3392	1761.58	(96)	3400	**1753.41**	**(96)**
Average					**(75)**			(83)

Table 2.2: Different strategies for processing a box by subdivision after the interval Krawczyk method

Problem	Krawczyk	Adaptive Krawczyk					
		t=1		t=2		t=3	
	Time	Time	(%)	Time	(%)	Time	(%)
S5	0.69	0.56	(81)	**0.53**	**(77)**	**0.53**	**(77)**
S7	0.99	0.83	(84)	**0.77**	**(78)**	0.85	(86)
S10	1.28	1.09	(85)	**1.07**	**(84)**	1.15	(90)
SHCB	1.52	**1.33**	**(87)**	1.57	(103)	2.03	(134)
BR	1.09	1.09	(100)	1.09	(100)	1.09	(100)
RO	0.27	0.24	(89)	**0.19**	**(70)**	0.24	(89)
L8	2.16	2.19	(101)	2.16	(100)	2.16	(100)
L9	2.99	2.99	(100)	**2.96**	**(99)**	**2.96**	**(99)**
H3	2.67	2.61	(98)	**2.37**	**(89)**	2.83	(106)
G5	**2.88**	2.91	(101)	2.91	(101)	2.91	(101)
R4	11.68	10.40	(89)	11.60	(99)	**9.55**	**(82)**
L12	11.81	**11.73**	**(99)**	11.76	(100)	**11.73**	**(99)**
L18	6.75	**6.69**	**(99)**	**6.69**	**(99)**	**6.69**	**(99)**
G7	**5.57**	5.60	(101)	5.60	(101)	5.60	(101)
G10	**9.12**	9.17	(101)	9.20	(101)	9.20	(101)
GP	**90.03**	102.13	(113)	106.53	(118)	530.67	(589)
H6	28.24	23.09	(82)	21.97	(78)	**20.45**	**(72)**
S2.14	78.93	65.07	(82)	**60.53**	**(77)**	102.08	(129)
GEO1	14.48	**14.37**	**(99)**	14.43	(100)	14.53	(100)
GEO2	23.09	21.31	(92)	**20.96**	**(91)**	22.35	(97)
GEO3	29.36	25.07	(85)	**23.63**	**(82)**	24.35	(83)
JS	**45.17**	49.68	(110)	59.84	(132)	51.81	(115)
S2.7	31.57	33.44	(106)	**29.47**	**(93)**	35.12	(111)
L3	130.43	**128.08**	**(98)**	132.61	(102)	139.97	(107)
R8	220.32	205.87	(93)	**201.23**	**(91)**	210.43	(96)
HM3	233.28	**230.88**	**(99)**	231.84	(99)	249.20	(107)
HM4	1761.57	**1659.68**	**(94)**	1661.92	(94)	1844.53	(111)
Average			(95)		**(94)**		(117)

Table 2.3: Time measured with different t

Problem	Krawczyk	Adaptive Krawczyk					
		t=1		t=2		t=3	
	Time	Time	(%)	Time	(%)	Time	(%)
R4	11.68	10.40	(89)	11.60	(99)	**9.55**	**(82)**
L12	11.81	**11.73**	**(99)**	11.76	(100)	**11.73**	**(99)**
L18	6.75	**6.69**	**(99)**	**6.69**	**(99)**	**6.69**	**(99)**
G7	**5.57**	5.60	(101)	5.60	(101)	5.60	(101)
G10	**9.12**	9.17	(101)	9.20	(101)	9.20	(101)
GP	**90.03**	102.13	(113)	106.53	(118)	530.67	(589)
H6	28.24	23.09	(82)	21.97	(78)	**20.45**	**(72)**
S2.14	78.93	65.07	(82)	**60.53**	**(77)**	102.08	(129)
GEO1	14.48	**14.37**	**(99)**	14.43	(100)	14.53	(100)
GEO2	23.09	21.31	(92)	**20.96**	**(91)**	22.35	(97)
GEO3	29.36	25.07	(85)	**23.63**	**(82)**	24.35	(83)
JS	**45.17**	49.68	(110)	59.84	(132)	51.81	(115)
S2.7	31.57	33.44	(106)	**29.47**	**(93)**	35.12	(111)
L3	130.43	**128.08**	**(98)**	132.61	(102)	139.97	(107)
R8	220.32	205.87	(93)	**201.23**	**(91)**	210.43	(96)
HM3	233.28	**230.88**	**(99)**	231.84	(99)	249.20	(107)
HM4	1761.57	**1659.68**	**(94)**	1661.92	(94)	1844.53	(111)
Average			**(96)**		(97)		(129)

Table 2.4: Time measured with different t for medium and hard problems only

Problem	Newton	Adaptive Newton					
		t=1		t=2		t=3	
	Time	Time	(%)	Time	(%)	Time	(%)
S5	0.67	0.56	(84)	**0.53**	**(79)**	**0.53**	**(79)**
S7	0.93	0.80	(86)	**0.75**	**(81)**	**0.75**	**(81)**
S10	1.31	1.09	(83)	**1.04**	**(79)**	1.07	(82)
SHCB	**0.69**	**0.69**	**(100)**	0.88	(128)	1.01	(146)
BR	1.09	1.09	(100)	1.09	(100)	1.09	(100)
RO	0.27	0.21	(78)	**0.19**	**(70)**	0.24	(89)
L8	2.51	2.51	(100)	2.51	(100)	2.51	(100)
L9	**2.93**	2.96	(101)	2.96	(101)	2.96	(101)
H3	2.61	2.51	(96)	**2.37**	**(91)**	2.83	(108)
G5	**2.88**	**2.88**	**(100)**	**2.88**	**(100)**	3.01	(105)
R4	11.49	10.05	(87)	10.29	(90)	**9.31**	**(81)**
L12	**13.23**	**13.23**	**(100)**	**13.23**	**(100)**	13.31	(101)
L18	**7.20**	**7.20**	**(100)**	**7.20**	**(100)**	7.31	(102)
G7	**5.60**	**5.60**	**(100)**	**5.60**	**(100)**	5.68	(101)
G10	**9.15**	**9.15**	**(100)**	**9.15**	**(100)**	9.33	(102)
GP	**83.71**	89.55	(107)	95.23	(114)	538.45	(643)
H6	27.95	23.49	(084)	22.37	(80)	**20.83**	**(75)**
S2.14	111.68	93.60	(084)	87.52	(78)	**85.76**	**(77)**
GEO1	14.48	**14.45**	**(100)**	14.48	(100)	14.75	(102)
GEO2	23.36	21.20	(91)	**20.83**	**(89)**	22.48	(96)
GEO3	29.47	25.33	(86)	**23.63**	**(80)**	24.37	(83)
JS	**38.48**	39.95	(104)	50.80	(132)	46.53	(121)
S2.7	31.68	31.39	(99)	**30.99**	**(98)**	33.33	(105)
L3	123.23	**120.19**	**(98)**	124.45	(101)	129.97	(105)
R8	217.81	205.79	(94)	201.23	(92)	**199.23**	**(91)**
HM3	164.32	161.89	(99)	**161.81**	**(98)**	161.95	(99)
HM4	**1086.96**	1088.08	(100)	1087.79	(100)	1087.89	(100)
Average:			**(95)**		(96)		(118)

Table 2.5: Applying the adaptive approach to the algorithm with the interval Newton method

Problem	Adaptive methods				Wiethoff	
	Krawczyk with $t = 1$		Newton with $t = 1$			
	Time	(%)	Time	(%)	Time	(%)
S5	**0.56**	**(28)**	**0.56**	**(28)**	0.88	(44)
S7	0.83	(29)	**0.80**	**(28)**	1.17	(41)
S10	**1.09**	**(28)**	**1.09**	**(28)**	1.65	(43)
SHCB	1.33	(37)	**0.69**	**(19)**	1.55	(43)
BR	**1.09**	**(27)**	**1.09**	**(27)**	1.79	(45)
RO	0.24	(34)	**0.21**	**(30)**	0.24	(34)
L8	**2.19**	**(28)**	2.51	(32)	3.01	(39)
L9	2.99	(26)	**2.96**	**(26)**	5.33	(47)
H3	2.61	(28)	**2.51**	**(26)**	4.19	(45)
G5	2.91	(26)	**2.88**	**(25)**	5.33	(47)
R4	10.40	(33)	**10.05**	**(31)**	11.01	(34)
L12	**11.73**	**(27)**	13.23	(31)	16.99	(40)
L18	6.69	(37)	7.20	(40)	**4.05**	**(22)**
G7	**5.60**	**(6)**	**5.60**	**(6)**	69.25	(86)
G10	9.17	(26)	**9.15**	**(26)**	15.92	(46)
GP	102.13	(37)	89.55	(32)	**82.96**	**(30)**
H6	23.09	(33)	23.49	(33)	**23.01**	**(33)**
S2.14	**65.07**	**(19)**	93.60	(28)	168.48	(51)
GEO1	14.37	(33)	14.45	(33)	**13.92**	**(32)**
GEO2	21.31	(31)	**21.20**	**(31)**	25.55	(37)
GEO3	**25.07**	**(33)**	25.33	(33)	25.41	(33)
JS	49.68	(33)	39.95	(26)	60.37	(40)
S2.7	33.44	(30)	**31.39**	**(28)**	45.65	(41)
L3	128.08	(33)	**120.19**	**(31)**	136.37	(35)
R8	205.87	(49)	205.79	(49)	**8.21**	**(1)**
HM3	230.88	(40)	**161.89**	**(28)**	170.77	(30)
HM4	1659.68	(43)	1088.08	(28)	**1048.59**	**(27)**
Average		(31)		**(29)**		(39)

Table 2.6: Comparison of Krawczyk and Newton methods using the adaptive approach with the algorithm by Wiethoff (with weighted percentage)

	Original		Modified	
Problem	Iterations	Time	Iterations	Time
S7	22	1.95	21	**1.79**
S10	22	2.69	21	**2.53**
RO	49	**0.53**	53	0.56
H6	291	52.80	290	**51.73**
GEO1	352	17.58	351	**17.39**
GEO2	425	29.27	424	**29.15**
S2.7	94	69.44	91	**68.80**

Table 2.7: The use of Algorithm 2.7 instead of Algorithm 2.6 gives a little improvement (using automatic differentiation)

Problem	Whole box		One component			Always		
	Iters	Time	Iters	Time	(%)	Iters	Time	(%)
S5	17	**1.28**	17	1.31	(102)	23	1.81	(141)
S7	21	1.87	21	**1.79**	**(95)**	27	2.75	(147)
S10	21	2.56	21	**2.53**	**(98)**	27	3.79	(148)
SHCB	695	11.49	235	**3.20**	**(27)**	271	4.67	(40)
BR	38	**1.44**	38	1.47	(102)	42	2.19	(152)
RO	49	**0.51**	53	0.56	(109)	47	0.64	(125)
L8	35	4.37	12	**1.41**	**(32)**	12	1.65	(37)
L9	38	6.45	14	**2.27**	**(35)**	14	2.80	(43)
H3	34	**3.41**	37	3.55	(104)	59	5.95	(174)
G5	31	**4.24**	31	4.27	(100)	31	5.15	(121)
R4	261	13.33	261	**13.23**	**(99)**	293	17.28	(129)
L12	166	100.88	34	**15.39**	**(15)**	33	17.44	(17)
L18	90	30.16	29	**8.05**	**(26)**	28	8.80	(29)
G7	42	**8.43**	42	8.48	(100)	42	10.00	(118)
G10	44	**13.89**	44	13.97	(100)	44	18.19	(130)
GP	7605	206.24	7687	**195.73**	**(94)**	7726	257.20	(124)
H6	289	53.84	290	**51.73**	**(96)**	320	62.67	(116)
S2.14	10221	251.49	5149	**140.67**	**(55)**	5160	169.73	(67)
GEO1	351	**17.39**	351	**17.39**	**(100)**	357	20.77	(119)
GEO2	1880	106.48	424	**29.15**	**(27)**	450	39.04	(36)
GEO3	1936	114.51	507	**36.61**	**(31)**	529	50.08	(43)
JS	503	121.09	216	**51.23**	**(42)**	253	68.77	(56)
S2.7	195	126.11	91	**68.80**	**(54)**	94	85.71	(67)
L3	634	184.00	634	**178.91**	**(97)**	778	239.79	(130)
R8	1145	286.19	1145	**267.60**	**(93)**	1145	301.31	(105)
HM3	754	249.23	754	**240.08**	**(96)**	809	309.73	(124)
HM4	3716	1976.27	3392	**1837.25**	**(92)**	3400	2248.64	(113)
Average					**(75)**			(98)

Table 2.8: Different strategies for processing the box by subdivision after the interval Krawczyk method (using automatic differentiation)

Problem	Krawczyk	Adaptive Krawczyk					
		t=1		t=2		t=3	
	Time	Time	(%)	Time	(%)	Time	(%)
S5	1.28	1.10	(85)	1.09	(85)	**1.07**	**(83)**
S7	1.79	**1.62**	**(90)**	1.72	(96)	1.64	(91)
S10	2.53	**2.15**	**(84)**	2.21	(87)	2.23	(88)
SHCB	3.12	**2.41**	**(77)**	3.72	(119)	4.13	(132)
BR	**1.47**	1.48	(100)	1.48	(100)	**1.47**	**(100)**
RO	0.56	**0.42**	**(74)**	0.44	(78)	0.51	(91)
L8	**1.39**	1.43	(102)	1.43	(102)	1.41	(101)
L9	**2.27**	2.30	(101)	2.45	(107)	2.28	(100)
H3	3.52	3.61	(102)	**3.33**	**(94)**	3.85	(109)
G5	**4.35**	4.37	(100)	4.40	(101)	4.52	(103)
R4	13.09	11.72	(89)	13.08	(99)	**10.74**	**(82)**
L12	**15.28**	15.62	(102)	15.59	(102)	15.64	(102)
L18	**8.03**	8.17	(101)	8.21	(102)	8.19	(101)
G7	**8.40**	11.64	(138)	8.60	(102)	8.59	(102)
G10	**13.89**	14.11	(101)	14.68	(105)	15.43	(111)
GP	198.21	**144.70**	**(73)**	182.16	(91)	632.05	(318)
H6	51.73	41.13	(79)	36.73	(71)	**33.83**	**(65)**
S2.14	139.44	115.48	(82)	**108.49**	**(77)**	172.12	(123)
GEO1	**17.39**	17.50	(100)	17.67	(101)	17.60	(101)
GEO2	29.15	27.27	(93)	**26.69**	**(91)**	27.76	(95)
GEO3	37.12	31.81	(85)	**29.47**	**(79)**	29.92	(80)
JS	**51.57**	58.48	(113)	68.95	(133)	57.63	(111)
S2.7	68.80	72.23	(104)	**63.12**	**(91)**	75.51	(109)
L3	179.57	**177.12**	**(98)**	180.74	(100)	185.72	(103)
R8	270.64	252.58	(93)	245.63	(90)	**242.67**	**(89)**
HM3	**238.88**	245.15	(102)	243.96	(102)	249.53	(104)
HM4	1855.68	1849.42	(99)	**1765.38**	**(95)**	1768.30	(95)
Average			**(95)**		(96)		(107)

Table 2.9: Time measured with different t (using automatic differentiation)

Problem	Newton	Adaptive Newton					
		t=1		t=2		t=3	
	Time	Time	(%)	Time	(%)	Time	(%)
S5	1.33	1.10	(82)	**1.04**	**(78)**	1.05	(78)
S7	1.87	1.59	(85)	**1.49**	**(79)**	**1.49**	**(79)**
S10	2.61	2.18	(83)	**2.05**	**(78)**	2.06	(78)
SHCB	1.47	**1.32**	**(89)**	1.86	(126)	1.89	(128)
BR	1.47	1.48	(100)	1.49	(101)	**1.46**	**(99)**
RO	0.51	0.42	(82)	**0.36**	**(70)**	0.41	(80)
L8	1.65	**1.60**	**(96)**	1.73	(104)	1.61	(97)
L9	2.64	**2.52**	**(95)**	2.72	(103)	2.53	(95)
H3	3.68	3.48	(94)	**3.45**	**(93)**	3.77	(102)
G5	4.61	4.30	(93)	4.44	(96)	**4.29**	**(93)**
R4	14.11	11.21	(79)	12.71	(90)	**10.29**	**(72)**
L12	18.85	17.59	(93)	19.10	(101)	**17.50**	**(92)**
L18	8.59	8.56	(99)	8.59	(100)	**8.53**	**(99)**
G7	8.67	9.06	(104)	9.08	(104)	**8.48**	**(97)**
G10	14.64	14.02	(95)	15.75	(107)	**13.96**	**(95)**
GP	185.47	**134.84**	**(72)**	176.00	(94)	612.76	(330)
H6	52.11	41.74	(80)	39.53	(75)	**34.60**	**(66)**
S2.14	213.79	175.60	(82)	167.59	(78)	**157.55**	**(73)**
GEO1	17.60	**17.56**	**(99)**	18.23	(103)	17.96	(102)
GEO2	29.52	**26.84**	**(90)**	27.96	(94)	29.40	(99)
GEO3	37.60	31.97	(85)	**29.54**	**(78)**	31.64	(84)
JS	47.20	**45.35**	**(96)**	76.53	(162)	55.79	(118)
S2.7	68.32	**67.81**	**(99)**	94.08	(137)	75.05	(109)
L3	171.33	**169.06**	**(98)**	227.57	(132)	184.43	(107)
R8	270.37	265.53	(98)	**242.12**	**(89)**	251.59	(93)
HM3	166.40	166.88	(100)	**165.75**	**(99)**	173.29	(104)
HM4	1138.16	1148.31	(100)	1136.09	(99)	1347.88	(118)
Average			**(91)**		(99)		(103)

Table 2.10: Applying the adaptive approach to the algorithm with the interval Newton method (using automatic differentiation)

Problem	Adaptive methods				Wiethoff	
	Krawczyk with $t = 1$		Newton with $t = 1$			
	Time	(%)	Time	(%)	Time	(%)
S5	**1.10**	**(30)**	**1.10**	**(30)**	1.38	(38)
S7	1.62	(31)	**1.59**	**(31)**	1.91	(37)
S10	**2.15**	**(30)**	2.18	(30)	2.80	(39)
SHCB	2.41	(42)	**1.32**	**(23)**	1.97	(34)
BR	**1.48**	**(31)**	**1.48**	**(31)**	1.74	(37)
RO	**0.42**	**(34)**	**0.42**	**(34)**	0.38	(31)
L8	**1.43**	**(28)**	1.60	(32)	1.92	(38)
L9	**2.30**	**(29)**	2.52	(31)	3.10	(39)
H3	3.61	(32)	**3.48**	**(31)**	3.88	(35)
G5	4.37	(30)	**4.30**	**(29)**	5.80	(40)
R4	11.72	(35)	**11.21**	**(33)**	10.51	(31)
L12	**15.62**	**(30)**	17.59	(34)	18.15	(35)
L18	**8.17**	**(18)**	8.56	(19)	26.52	(61)
G7	11.64	(9)	**9.06**	**(7)**	99.05	(82)
G10	14.11	(30)	**14.02**	**(30)**	18.07	(39)
GP	144.70	(38)	134.84	(35)	**98.94**	**(26)**
H6	41.13	(35)	41.74	(36)	**32.05**	**(27)**
S2.14	**115.48**	**(20)**	175.60	(31)	268.86	(48)
GEO1	**17.50**	**(34)**	17.56	(34)	16.29	(31)
GEO2	27.27	(32)	**26.84**	**(31)**	30.75	(36)
GEO3	**31.81**	**(32)**	31.97	(32)	35.01	(35)
JS	58.48	(36)	**45.35**	**(28)**	54.99	(34)
S2.7	72.23	(31)	**67.81**	**(29)**	90.87	(39)
L3	177.12	(30)	**169.06**	**(29)**	226.70	(39)
R8	**252.58**	**(30)**	265.53	(32)	303.73	(36)
HM3	245.15	(43)	**166.88**	**(29)**	146.39	(26)
HM4	1849.42	(45)	1148.31	(28)	**1095.20**	**(26)**
Average		(31)		**(30)**		(38)

Table 2.11: Comparison of Krawczyk and Newton methods using the adaptive approach with Wiethoff's method (with weighted percentage, using automatic differentiation)

Chapter 3

Nonlinear Systems: Convergence and Divergence of Interval Newton and Krawczyk Methods

In this chapter we will investigate properties of interval Newton and interval Krawczyk methods and their modifications. In the previous chapter we used them for accelerating the serial method for verified global optimization. These interval methods can be used not only as an accelerating device in methods for verified global optimization but also alone for the solution of systems of nonlinear equations. In some cases, using exclusive properties of systems, we can decrease the running time of the method. In this chapter we give sufficient conditions for convergence and divergence of modified Newton and Krawczyk methods.

3.1 Notations and Preliminaries

We denote the point vector $(e_i) \in \mathbb{R}^n$, $e_i = 1$, $1 \leq i \leq n$, and the point matrix $(e_{ij}) \in \mathbb{R}^{n \times n}$, $e_{ij} = 1$, $1 \leq i, j \leq n$, by e and E, respectively. By I we denote the identity matrix of the proper dimension.
We will use norms

$$\|x\| = \|x\|_\infty = \max_i |x_i|, \quad \|A\| = \max_{i,j} |a_{ij}|.$$

For lower triangular and upper triangular matrices L and U with non-zero diagonal entries we denote $L \setminus$ and $U \setminus$ the forward and backward substitutions, respectively. This notation has been given by Rump in [22]; see (3.3) and (3.4) below.
Given $L = (l_{ij}) \in \mathbb{R}^n$, a lower triangular matrix, and $U = (u_{ij}) \in \mathbb{R}^n$, an upper triangular

matrix, we define matrices:

$$L^i = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ -\frac{l_{i1}}{l_{ii}} & \dots & -\frac{l_{i,i-1}}{l_{ii}} & \frac{1}{l_{ii}} & & & & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \end{pmatrix}, \qquad l_{ii} \neq 0, \quad 1 \leq i \leq n, \tag{3.1}$$

$$U^i = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \frac{1}{u_{ii}} & -\frac{u_{i,i+1}}{u_{ii}} & \dots & -\frac{u_{in}}{u_{ii}} & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \end{pmatrix}, \qquad u_{ii} \neq 0, \quad 1 \leq i \leq n. \tag{3.2}$$

The i-th step in the forward substitution with matrix L is equivalent to multiplication by the matrix L^i, we have

$$L \setminus [x] = (L^n(L^{n-1} \dots (L^1[x]) \dots)). \tag{3.3}$$

Here we cannot omit parenthasis because of subdistributivity in interval arithmetic. Moreover, for the matrices $L^1, \dots, L^n$ we have

$$L^n \cdot \dots \cdot L^1 = L^{-1}. \tag{3.4}$$

The same is valid for upper triangular matrices and backward substitution:

$$U \setminus [x] = (U^n(U^{n-1} \dots (U^1[x]) \dots)), \tag{3.5}$$

$$U^n \cdot \dots \cdot U^1 = U^{-1}. \tag{3.6}$$

Lemma 1. *For point matrices A, B, an interval matrix $[C]$ and an interval vector $[x]$*

$$A \cdot (B \cdot [x]) \supseteq (A \cdot B) \cdot [x],$$
$$A \cdot (B + [C]) = A \cdot B + A \cdot [C]$$

holds.

Proof. Application of subdistributivity several times results in $A \cdot (B \cdot [x]) \supseteq (A \cdot B) \cdot [x]$. And $A \cdot (B + [C]) = A \cdot B + A \cdot [C]$ is trivial. □

Lemma 2. *For a nonsingular lower triangular matrix L, a nonsingular upper triangular matrix U and an interval vector $[x]$*

$$L \setminus [x] \supseteq L^{-1}[x],$$
$$U \setminus [x] \supseteq U^{-1}[x]$$

are valid.

Proof. Because of the subdistributivity, from

$$L \setminus [x] = (L_n(L_{n-1} \dots (L_1[x]) \dots)),$$
$$U \setminus [x] = (U_n(U_{n-1} \dots (U_1[x]) \dots))$$

we get

$$L \setminus [x] \supseteq (L_n \cdot L_{n-1} \cdot \dots \cdot L_1)[x],$$
$$U \setminus [x] \supseteq (U_n \cdot U_{n-1} \cdot \dots \cdot U_1)[x]$$

and therefore

$$L \setminus [x] \supseteq L^{-1}[x],$$
$$U \setminus [x] \supseteq U^{-1}[x].$$

□

Lemma 3. *For an interval matrix* $[A]$ *and a point vector* b

$$[A] \cdot b = \{A \cdot b \mid A \in [A]\}$$

holds.

Proof. Let $[A] = ([a]_{ij})$ and $b = (b_i)$ be the given interval matrix and point vector. $\{A \cdot b | A \in [A]\} \subseteq [A] \cdot b$ is obvious. We will prove that $[A] \cdot b \subseteq \{A \cdot b | A \in [A]\}$.
Let $c = (c_i)$ be an arbitrary point vector from $[A] \cdot b$. Then $c_1 \in \sum_{i=1}^{n} [a_{1i}] \cdot b_i$. Hence there are $a^*_{11},\ a^*_{12},\ ...,\ a^*_{1n} \in \mathbb{R}$ such that $c_1 = \sum_{j=1}^{n} a^*_{1j} \cdot b_j$. In the same manner we find $a^*_{21},\ ...,\ a^*_{2n}, a^*_{31},\ ...,\ a^*_{nn} \in \mathbb{R}$ such that $c_i = \sum_{j=1}^{n} a^*_{ij} \cdot b_j$. For $A^* = (a^*_{ij})$ we get $A^* \cdot b = c$.
Since $\{A \cdot b | A \in [A]\} \subseteq [A] \cdot b$ and $[A] \cdot b \subseteq \{A \cdot b | A \in [A]\}$ we get equality. □

Lemma 4. *For an interval matrix* $[A]$ *and a point vector* b

$$[A] \cdot b = [A^1 \cdot b, A^2 \cdot b]$$

is valid for some $A^1,\ A^2 \in [A]$.

Proof. Let $n^1,\ n^2 \in [A] \cdot b$ be the lower and upper bounds of $[A] \cdot b$. Then based on the previous lemma they can be represented as a product of a point matrix from $[A]$ and the given point vector b:

$$n^1 = A^1 \cdot b,\ n^2 = A^2 \cdot b, \qquad A^1,\ A^2 \in [A].$$

Hence $[A] \cdot b = [A^1 \cdot b, A^2 \cdot b]$. □

Let there be given a Fréchet-differentiable mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ with components $f_i(x)$, $1 \le i \le n$, of the vector variable $x = (x_1, ..., x_n)^T \in [x]^0$. Let us consider the function $f_i(x_1, ..., x_n)$ only in dependence on x_1, and then let it be expanded at the point y_1 as

$$f_i(x_1, x_2, ..., x_n) \;=\; f_i(y_1, x_2, ..., x_n) \;+\; \frac{\partial f_i}{\partial x_1}(y_1 + \theta_{i1} \cdot (x_1 - y_1), x_2, ..., x_n) \;\cdot\; (x_2 \;-\; y_2)$$

with $\theta_{i1} \in (0,1)$.
We then expand $f_i(y_1, x_2, ..., x_n)$ as a function of x_2 at the point y_2 and get

$$f_i(y_1, x_2, x_3, ..., x_n) = f_i(y_1, y_2, x_3, ..., x_n) + \frac{\partial f_i}{\partial x_2}(y_1, y_2 + \theta_{i2} \cdot (x_2 - y_2), x_3, ..., x_n) \cdot (x_2 - y_2).$$

We now expand $f_i(y_1, y_2, x_3, ..., x_n)$ again as a function only of x_3 at the point y_3 and so on until $f_i(y_1, ..., y_{n-1}, x_n)$ is expanded at the point y_n. If we now insert the expressions formed in this manner into each other we get

$$f_i(x_1, ..., x_n) = f_i(y_1, ..., y_n) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(y_1, y_{j-1}, y_j + \theta_{ij} \cdot (x_j - y_j), x_{j+1}, ..., x_n) \cdot (x_j - y_j),$$

from which we have the relation

$$f(x) - f(y) = \delta f(x,y) \cdot (x - y), \qquad x, y \in [x]^0, \tag{3.7}$$

where the matrix $\delta f(x,y)$ is defined by

$$\delta f(x,y) = \left(\frac{\partial f_j}{\partial x_j}(y_1, ..., y_{j-1}, z_j, x_{j+1}, ..., x_n) \Big|_{z_j = y_j + \theta_{ij} \cdot (x_j - y_j)} \right), \quad 1 \le j \le n.$$

It clearly follows that for any enclosure function F' of f' we have

$$\delta f(x^1, x^2) \in F'([x]^0), \quad x^1, x^2 \in [x]^0, \tag{3.8}$$

since

$$f'([x]) = \left(\frac{\partial f}{\partial x_j}([x]_1, ..., [x]_n) \right), \quad 1 \le j \le n.$$

Assume now that we have given an $n \times n$ interval matrix $[A] = ([a]_{ij})$ and an interval vector $[b] = ([b]_i)$ with n components. By applying the formulae of the Gaussian algorithm we can compute an interval vector $[x] = ([x]_i)$ for which the relation

$$s = \{A^{-1} \cdot b \big| A \in [A], b \in [b]\} \subseteq [x]$$

holds (see [53], p.20ff, for example). Algorithm 3.1 describes this process of interval Gaussian elimination in more details. We set $[a]^1_{ij} = [a]_{ij}$, $1 \le i, j \le n$, and $[b]^1_i = [b]_i$, $1 \le i \le n$. Here, we assume that the necessary pivoting has been performed to the matrix in advance in order to prevent division by an interval which contains zero. Therefore in the formulae we will not take into account exchanges of rows or columns. If one programs Algorithm 3.1 then the upper index can be suppressed and the loop over l can be ignored.
Given an interval matrix $[A]$ we define two matrices $[L] = ([l]_{ij})$ and $[U] = ([u]_{ij})$ as follows:

$$\begin{aligned}
[l]_{ij} &= \frac{[a]^j_{ij}}{[a]^j_{ii}}, \qquad 1 \le j < i \le n, \\
[l]_{ii} &= 1 \qquad 1 \le i \le n, \\
[l]_{ij} &= 0 \qquad 1 \le i < j \le n; \\
[u]_{ij} &= [a]^n_{ij} \qquad 1 \le i \le j \le n, \\
[u]_{ij} &= 0 \qquad 1 \le j < i \le n,
\end{aligned}$$

Algorithm 3.1 Gaussian Elimination Algorithm

for $k = 1$ to $n-1$ **do**
  for $i = k+1$ to n **do**
    for $j = k+1$ to n **do**
      $[a]_{ij}^{k+1} = [a]_{ij}^{k} - \frac{[a]_{ik}^{k}}{[a]_{kk}^{k}} \cdot [a]_{kj}^{k}$
    end for
    $[b]_{i}^{k+1} = [b]_{i}^{k} - \frac{[a]_{ik}^{k}}{[a]_{kk}^{k}} \cdot [b]_{k}^{k}$
  end for
  for $l = 1$ to k **do**
    for $j = l$ to n **do**
      $[a]_{lj}^{k+1} = [a]_{lj}^{k}$
    end for
    $[b]_{j}^{k+1} = [b]_{j}^{k}$
    for $j = l+1$ to n **do**
      $[a]_{jl}^{k+1} = 0$
    end for
  end for
end for
$[x]_n = \frac{[b]_n^n}{[a]_{nn}^n}$
for $i = n-1$ to 1 **do**
  $[x]_i = \frac{[b]_i^n - \sum_{j=i+1}^{n} [a]_{ij}^n \cdot [x]_j}{[a]_{ii}^n}$
end for

where $[a]_{ij}^{k}$ come from Algorithm 3.1. Because of the construction for the case when $[A]$ is a point matrix we get the equality

$$A = L \cdot U,$$

and in the general case

$$[A] \subseteq [L] \cdot [U]. \tag{3.9}$$

For $[L]$ and $[U]$ we define matrices $[L]^1, \ldots [L]^n$, $[U]^1, \ldots, [U]^n$ similarly to (3.1) and (3.2):

$$[L]^i = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ -\frac{[l]_{i1}}{[l]_{ii}} & \ldots & -\frac{[l]_{i,i-1}}{[l]_{ii}} & \frac{1}{[l]_{ii}} & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}, \qquad 1 \le i \le n, \tag{3.10}$$

$$[U]^i = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \frac{1}{[u]_{ii}} & -\frac{[u]_{i,i+1}}{[u]_{ii}} & \cdots & -\frac{[u]_{in}}{[u]_{ii}} \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}, \qquad 1 \le i \le n. \tag{3.11}$$

If we define the interval-matrices

$$[\mathbf{P}]^k = \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & 1 & 0 \\ & -\frac{[a]^k_{k+1,k}}{[a]^k_{kk}} & \\ & \vdots & \\ 0 & -\frac{[a]^k_{nk}}{[a]^k_{kk}} & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n-1, \tag{3.12}$$

$$[\mathbf{Q}]^k = \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & \frac{1}{[a]^n_{kk}} & 0 \\ 0 & 0 & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n, \tag{3.13}$$

$$[\mathbf{R}]^k = \begin{pmatrix} I_{k-1} & 0 & & & 0 \\ 0 & 1 & -[a]^n_{k,k+1} & \cdots & -[a]^n_{kn} \\ 0 & 0 & & & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n-1, \tag{3.14}$$

then for the interval vector $[x]$, resulting from Algorithm 3.1 we have the representation

$$[x] = [Q]^1([R]^1([Q]^2([R]^2(\dots([Q]^{n-1}([R]^{n-1}([Q]^n \cdot [\tilde{b}])\dots),$$

where

$$[\tilde{b}] = [P]^{n-1}([P]^{n-2}(\dots([P]^2([P]^1 \cdot [b])\dots).$$

This has been first given by Schwandt in [53]. As in [53] we denote this interval vector by $IGA([A],[b])$, that is we have the representation

$$IGA([A],[b]) = [Q]^1([R]^1 \dots ([Q]^{n-1}([R]^{n-1}([Q]^n([P]^{n-1}(\dots([P]^2([P]^1 \cdot [b])\dots). \tag{3.15}$$

Notice that it is not possible to omit the parenthesis in general. We define an interval-matrix $IGA([A])$ by using the interval-matrices occuring on the right-hand side of (3.15):

$$IGA([A]) = [Q]^1([R]^1([Q]^2([R]^2(\dots([Q]^{n-1}([R]^{n-1}([Q]^n([P]^{n-1}(\dots([P]^2[P]^1)\dots). \tag{3.16}$$

$IGA([A])$ exists if the matrices $[P]^i$, $[Q]^i$ and $[R]^i$ exist. And matrices $[P]^i$, $[Q]^i$ and $[R]^i$ exist if the Gaussian elimination algorithm can be performed for $[A]$ and every right-hand side $[b]$. In other words $IGA([A])$ exists if $0 \notin [a]^k_{kk}$, $k = 1, ..., n$, in Algorithm 3.1.
From the construction of $IGA([A])$ and from inclusion isotony in interval arithmetic we get

Lemma 5. *If $[A] \subseteq [B]$ and $IGA([B])$ exists then*

$$IGA([A]) \subseteq IGA([B]).$$

Lemma 6. *If $C \in [A]$ and $IGA([A])$ exists then C is nonsingular and $C^{-1} \in IGA([A])$.*

Proof. Since $IGA([A])$ exists, we can perform the Gaussian elimination algorithm for $[A]$. Since $C \in [A]$, the Gaussian elimination algorithm can be performed for C, too. Therefore C is nonsingular.
We set $C^1 = C = (c_{ij})$, $1 \le i, j \le n$, and perform the Gaussian elimination algorithm. We get the matrices C^k, $1 \le k \le n$. We define point matrices:

$$P^k = \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & 1 & 0 \\ & -\frac{c^k_{k+1,k}}{c^k_{kk}} & \\ & \vdots & \\ 0 & -\frac{c^k_{nk}}{c^k_{kk}} & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n-1,$$

$$Q^k = \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & \frac{1}{c^n_{kk}} & 0 \\ 0 & 0 & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n,$$

$$R^k = \begin{pmatrix} I_{k-1} & 0 & & & 0 \\ 0 & 1 & -c^n_{k,k+1} & \cdots & -c^n_{kn} \\ 0 & 0 & & & I_{n-k} \end{pmatrix}, \qquad 1 \le k \le n-1.$$

From the construction of P^k, Q^k and R^k and inclusion isotony in interval arithmetic we have

$$P^k \in [P]^k, \quad Q^k \in [Q]^k, \quad R^k \in [R]^k. \tag{3.17}$$

Now we calculate the matrix:

$$C^* = Q^1(R^1(Q^2(R^2(\dots(Q^{n-1}(R^{n-1} \cdot Q^n)\dots).$$

In the calculation of $C^* \cdot C$ we can rearrange parenthesis because we deal with point matrices:

$$C^* \cdot C = Q^1(R^1(Q^2(R^2(\dots(Q^{n-1}(R^{n-1}(Q^n \cdot C)\dots) = I.$$

We get that $C^* = C^{-1}$ and from (3.17) we get $C^* \in IGA([A])$. □

Lemma 7. *Let $[B]$ be an interval matrix such that $IGA([B])$ exists. Then there exist positive numbers α^k_{ij} and β_k, $i, j, k = 1, ..., n$ such that for all interval matrices $[A] \subseteq [B]$ we have*

$$\begin{aligned} d\left([a]^k_{ij}\right) &\le \alpha^k_{ij} \cdot \|d([B])\|, \\ d\left(\frac{1}{[a]^k_{kk}}\right) &\le \beta_k \cdot \|d([B])\|, \end{aligned} \tag{3.18}$$

where $[a]^k_{ij}$, $1 \le i, j, k \le n$ come from performing Algorithm 3.1 on the matrix $[A]$.

Proof. We set $[\hat{a}]^1_{ij} = [b]_{ij}$, $1 \le i, j \le n$, and perform Algorithm 3.1. We get intervals $[\hat{a}]^k_{ij}$, $1 \le i, j \le n$, $2 \le k \le n$. Then we determine constants α^k_{ij} and β_k, $1 \le i, j, k \le n$, such that

$$\begin{aligned} d([\hat{a}]^k_{ij}) &\le \alpha^k_{ij} \cdot \|d([B])\|, \\ d\left(\frac{1}{[\hat{a}]^k_{kk}}\right) &\le \beta_k \cdot \|d([B])\| \end{aligned}$$

are fulfilled. For intervals $[a]_1 \subseteq [a]_2$, $[b]_1 \subseteq [b]_2$ we have:

$$\frac{1}{[a]_1} \subseteq \frac{1}{[a]_2}, \qquad \text{if } 0 \notin [a]_2,$$
$$[a]_1 \cdot [b]_1 \subseteq [a]_2 \cdot [b]_2, \qquad [a]_1 - [b]_1 \subseteq [a]_2 - [b]_2.$$

Therefore for all intervals $[a]_{ij}^k$, $1 \le i, j, k \le n$, resulting from Algorithm 3.1 performed on $[A] \subseteq [B]$

$$d\left([a]_{ij}^k\right) \le d\left([\hat{a}]_{ij}^k\right) \le \alpha_{ij}^k \cdot \|d([B])\|,$$
$$d\left(\frac{1}{[a]_{kk}^k}\right) \le d\left(\frac{1}{[\hat{a}]_{kk}^k}\right) \le \beta_k \cdot \|d([B])\|$$

are valid. □

Lemma 8. *Let $[A]$, $[B]$ be interval matrices and c a positive constant such that*

$$d([A]) \le c \cdot U,$$
$$d([B]) \le c \cdot V,$$

where U, V are nonnegative matrices. Then

$$d(([A] \cdot [B])) \le c \cdot W$$

is valid, where W is a nonnegative matrix which depends on $|[A]|$, $|[B]|$ and U, V.

Proof. Indeed

$$\begin{aligned} d(([A] \cdot [B])) &\le d(A) \cdot |B| + |A| \cdot d(B) \\ &\le c \cdot U \cdot |B| + c \cdot |A| \cdot V \\ &= c \cdot W, \qquad \text{where } W = U \cdot |B| + |A| \cdot V. \end{aligned}$$

□

Lemma 9. *Let there be given a function $F' : \mathbb{IR}^n \to \mathbb{IR}^n$. If $IGA\left(F'([x]^0)\right)$ exists and*

$$d\left(F'([x])\right) \le \|d([x])\| \cdot S, \quad [x] \in [x]^0, \tag{3.19}$$

then

$$d\left(IGA(F'([x]))\right) \le \|d([x])\| \cdot T, \quad [x] \in [x]^0,$$

where S and T are nonnegative matrices.

Proof. For matrices $[P]^k$, $[Q]^k$, $[R]^k$ (see (3.12), (3.13) and (3.14)) from Lemma 7 and (3.19) we have

$$d\left([P]^k\right) \le \|d([x])\| \cdot C_p^k, \qquad 1 \le k \le n-1,$$
$$d\left([Q]^k\right) \le \|d([x])\| \cdot C_q^k, \qquad 1 \le k \le n,$$
$$d\left([R]^k\right) \le \|d([x])\| \cdot C_r^k, \qquad 1 \le k \le n-1,$$

with nonnegative point matrices C_p^k, C_q^k and C_r^k. Now using Lemma 8 (with $\|d([x])\|$ instead of c) $3n-3$ times we get that

$$d\left(IGA([A])\right) \le \|d([x])\| \cdot T.$$

□

Lemma 10. *Let there be given an interval vector* $[x]^0 \subset \mathbb{R}^n$ *and an inclusion monotonic function* $F' : [x]^0 \subset \mathbb{IR}^n \to \mathbb{IR}^n$.
Suppose $IGA\left(F'([x]^0)\right)$ *exists and denote* $A = \left|I - IGA\left(F'([x]^0)\right) \cdot F'([x]^0)\right|$.
If $\rho(A) < 1$ *(ρ spectral radius) then all point matrices from* $IGA\left(F'([x]^0)\right)$ *are nonsingular.*

Proof. If $U \in IGA\left(F'([x]^0)\right)$, $V \in F'([x]^0)$ then

$$|I - U \cdot V| \leq \left|I - IGA\left(F'([x]^0)\right) \cdot F'([x]^0)\right| = A$$

For nonnegative matrices $|I - U \cdot V|$ and A from the theorem of Perron and Frobenius (see [41]) it follows that

$$\rho(|I - U \cdot V|) \leq \rho(A) < 1.$$

Since for an arbitrary matrix C the inequality $\rho(C) \leq \rho(|C|)$ holds, for the matrix $I - U \cdot V$ we have

$$\rho(I - U \cdot V) \leq \rho(|I - U \cdot V|) < 1.$$

Hence

$$(U \cdot V)^{-1} = (I - (I - U \cdot V))^{-1}$$

exists, from which the existence of U^{-1} follows. □

Lemma 11. *For two given interval vectors* $[x] = [x^1, x^2]$ *and* $[y] = [y^1, y^2]$ *the relation* $[x] \cap [y] = \varnothing$ *is valid if and only if*

$$\left(\frac{1}{2}d([x]) + \frac{1}{2}d([y])\right)_{i_0} < |m[x] - m[y]|_{i_0}.$$

This takes place if and only if there is at least one index $i_0 \in \{1, 2, \ldots, n\}$ *such that*

$$\left(y^2 - x^1\right)_{i_0} < 0 \qquad or \qquad \left(x^2 - y^1\right)_{i_0} < 0.$$

Proof. Let $[x] \cap [y] = \varnothing$, then there is at least one index $i_0 \in \{1, 2, \ldots, n\}$ such that

$$[x]_{i_0} \cap [y]_{i_0} = \varnothing.$$

This is equivalent to

$$y^2_{i_0} < x^1_{i_0} \qquad or \qquad x^2_{i_0} < y^1_{i_0}. \tag{3.20}$$

This proves the second part of the lemma. Since $m[x]$ and $m[y]$ are midpoints of $[x]$ and $[y]$ we can write:

$$\begin{aligned} [x] &= \left[m[x] - \frac{1}{2}d([x]), m[x] + \frac{1}{2}d([x])\right], \\ [y] &= \left[m[y] - \frac{1}{2}d([y]), m[y] + \frac{1}{2}d([y])\right]. \end{aligned} \tag{3.21}$$

Using (3.21) we write (3.20) as follows:

$$m[y]_{i_0} + \frac{1}{2}d([y])_{i_0} < m[x]_{i_0} - \frac{1}{2}d([x])_{i_0}$$

or

$$m[x]_{i_0} + \frac{1}{2}d([x])_{i_0} < m[y]_{i_0} - \frac{1}{2}d([y])_{i_0}.$$

This is equivalent to

$$\frac{1}{2}d([x])_{i_0} + \frac{1}{2}d([y])_{i_0} < m[x]_{i_0} - m[y]_{i_0}$$

or

$$\frac{1}{2}d([y])_{i_0} + \frac{1}{2}d([x])_{i_0} < -(m[x]_{i_0} - m[y]_{i_0}).$$

Since the sum of diameters is nonnegative, only that one of these two inequalities, whose right-hand side is positive, takes place. And in that case the right-hand side is equal to $|m[x]_{i_0} - m[y]_{i_0}|$. Therefore we get

$$\left(\frac{1}{2}d([x]) + \frac{1}{2}d([y])\right)_{i_0} < |m[x] - m[y]|_{i_0}.$$

□

Lemma 12. *Let there be given a fixed point matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$. Then for all point vectors $b = (b_i) \in \mathbb{R}$ we have*

$$|A \cdot b| = O(\|b\|) \cdot e.$$

Proof. Denote $A \cdot b$ by $c = (c_i)$.

$$c_i = \sum_{j=1}^{n} a_{ij} \cdot b_j.$$

Therefore

$$|c_i| = \left|\sum_{j=1}^{n} a_{ij} \cdot b_j\right| \le \sum_{j=1}^{n} |a_{ij} \cdot b_j| = \|b\| \sum_{j=1}^{n} \left|a_{ij} \cdot \frac{b_j}{\|b\|}\right| \le \|b\| \sum_{j=1}^{n} |a_{ij}| \le \|b\| \cdot m_i,$$

where $m_i = \sum_{j=1}^{n} |a_{ij}| \ge 0$. But each $m_i = O(1)$, so $|c| = \|b\| \cdot O(1) \cdot e = O(\|b\|) \cdot e$. □

Lemma 13. *Let $[A] = ([a]_{ij})$, $1 \le i, j \le n$ be an interval matrix that does not contain singular point matrices and let $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ be a continuous mapping for which $f(x) \ne 0, x \in [x]^0$. Then there exist $c > 0$ and $\varepsilon(c) > 0$ such that for each $x \in [x]^0$ there is at least one index $i(x)$ such that*

$$\left|(A \cdot f(x))_{i(x)}\right| > c, \qquad \text{for all } A \in [A]$$

and

$$sign\left(\left(A^1 \cdot f(x^1)\right)_{i(x^1)}\right) = sign\left(\left(A^2 \cdot f(x^2)\right)_{i(x^1)}\right), \qquad \text{for all } A^1, A^2 \in [A],$$

once $|x^1 - x^2| < \varepsilon(c) \cdot e$.

Proof. From the assumption it follows

$$0 \notin [A] \cdot f(x), \qquad x \in [x]^0. \tag{3.22}$$

Otherwise by Lemma 3 there would exist $A \in [A],\ x \in [x]$ such that $A \cdot f(x) = 0$. But A is nonsingular and $f(x) \neq 0$, which is a contradiction.
Since (3.22) holds for each $x \in [x]^0$ there is at least one index $i(x)$ such that

$$0 \notin ([A] \cdot f(x))_{i(x)}.$$

If there are several indices we take the one for which

$$\omega_i = \min_{A \in [A]} |(A \cdot f(x))_i|$$

is the largest. For the selected $i(x)$

$$\omega_{i(x)} > 0 \tag{3.23}$$

holds.
Suppose we can not choose $c > 0$ such that

$$|(A \cdot f(x))_{i(x)}| > c \quad \text{for all } A \in [A],\ x \in [x]^0.$$

Then for any $c > 0$ there exists a pair $(A(c), x(c)), A(c) \in [A], x(c) \in [x]^0$ such that

$$\left|(A(c) \cdot f(x(c)))_{i(x(c))}\right| < c.$$

For a sequence $\{c^n\}_{n=1}^{\infty} = \left\{\frac{1}{n}\right\}_{n=1}^{\infty}$ we obtain a sequence $\{(A(c^n), x^n)\}_{n=1}^{\infty}$. Since $[A]$ and $[x]^0$ are both compact sets there exists a subsequence $\{(A^{n_k}, x^{n_k})\}_{k=1}^{\infty}$ such that $(A^*, x^*) = \lim_{k\to\infty}(A^{n_k}, x^{n_k})$. For (A^*, x^*)

$$\left|(A^* \cdot f(x^*))_{i(x^*)}\right| \leq \lim_{k\to\infty} c^{n_k} = 0.$$

is valid. But this contradicts (3.23). Consequently there exists such $c > 0$.
$([A] \cdot f(x))_{i(x)}$ is an interval that does not contain zero. Therefore either $\sup\left(([A] \cdot f(x))_{i(x)}\right) < 0$ or $\inf\left(([A] \cdot f(x))_{i(x)}\right) > 0$ and from this it follows

$$sign\left(\left(A^1 \cdot f(x)\right)_{i(x)}\right) = sign\left(\left(A^2 \cdot f(x)\right)_{i(x)}\right) \qquad \text{for all } A^1, A^2 \in [A]. \tag{3.24}$$

The n functions $g_i(x) = A_i^2 \cdot f(x)$, $1 \leq i \leq n$, where A_i^2 is the i-th row of A^2, are continuous. Therefore there exist ε_i, $1 \leq i \leq n$, such that $|g_i(x^1) - g_i(x^2)| < c$ when $|x^1 - x^2| < \varepsilon_i \cdot e$. Define $\varepsilon = \min_i \varepsilon_i$. At this point ε depends on A^2. We will show that we can choose $\varepsilon > 0$ which can be applied to all $A \in [A]$. Suppose we cannot choose such $\varepsilon > 0$. Then for every $\varepsilon_n = \frac{1}{n}$ there exist $A^n \in [A]$ and $x^n, y^n \in [x]$ such that

$$|A_i^n \cdot f(x^n) - A_i^n \cdot f(y^n)| \geq c \qquad \text{and } |x^n - y^n| < \varepsilon_n \cdot e$$

for $i = i(n)$. Again $[A]$ and $[x]$ are compact sets. Therefore there exists a convergent subsequence $\{(A^{n_k}, x^{n_k}, y^{n_k})\}_{k=1}^{\infty}$ such that, in addition, $i(n_k) = i^*$ for all k. Denote $(A^*, x^*, y^*) = \lim_{k\to\infty}(A^{n_k}, x^{n_k}, y^{n_k})$. For (A^*, x^*, y^*) we have

$$|A_{i^*}^* \cdot f(x^*) - A_{i^*}^* \cdot f(y^*)| \geq c \qquad \text{for } x^*, y^*. \tag{3.25}$$

From the selection of $\varepsilon_n = \frac{1}{n}$ it follows

$$x^* = \lim_{k\to\infty} x^{n_k} = \lim_{k\to\infty} y^{n_k} = y^*.$$

Hence

$$|A_i^* \cdot f(x^*) - A_i^* \cdot f(y^*)| = 0,$$

which contradicts to (3.25). Therefore we can choose $\varepsilon > 0$ independently from $A \in [A]$. Then for the given x^1 we have

$$\left|\left(A^2 \cdot f(x_1)\right)_{i(x_1)}\right| = \left|g_{i(x_1)}(x_1)\right| > c$$

and for x^2 fulfilling the condition $|x^1 - x^2| < \varepsilon \cdot e$ we have

$$sign\left(\left(A^2 \cdot f(x^1)\right)_{i(x^1)}\right) = sign\left(\left(A^2 \cdot f(x^2)\right)_{i(x^1)}\right) \qquad \text{for all } A^2 \in [A],$$

once $|x^1 - x^2| < \varepsilon \cdot e$.
From the last equation and (3.24) it follows that

$$sign\left(\left(A^1 \cdot f(x^1)\right)_{i(x^1)}\right) = sign\left(\left(A^2 \cdot f(x^2)\right)_{i(x^1)}\right) \qquad \text{for all } A^1, A^2 \in [A],$$

once $|x^1 - x^2| < \varepsilon \cdot e$. □

Lemma 14. *Let A be a point matrix and $\{x^n\}_{n=1}^{\infty}$ a sequence of nonnegative point vectors. We assume*

$$\rho(A) < 1,$$
$$x^{k+1} \le x^k,$$
$$x^{k+1} \le A \cdot x^k.$$

Then

$$\lim_{n\to\infty} x^n = 0.$$

Proof. $\{x^n\}_{n=1}^{\infty}$ is a nonincreasing sequence bounded from below. Therefore $x^* = \lim_{n\to\infty} x^n$ exists. From $x^{k+1} \le A \cdot x^k$ it follows

$$x^{k+1} \le A^k \cdot x^1.$$

Using the continuity of operations from the last inequality and tending $k \to \infty$ we get $x^* \le 0$, since $\rho(A) < 1$ inplies $\lim_{k\to\infty} A^k = 0$. And since $x^k \ge 0, \quad k = 0, 1, \ldots$ we have $x^* = 0$. □

Lemma 15. *If $IGA([A])$ exists then*

$$IGA([A], b) \subseteq IGA([A]) \cdot b.$$

Proof. For interval matrices $[A]$ and $[B]$ and a point vector c we have

$$\begin{aligned}([A] \cdot ([B] \cdot c))_i &= \sum_{j=1}^{n} [a]_{ij} \cdot ([B] \cdot c)_j \\ &= \sum_{j=1}^{n} [a]_{ij} \cdot \left(\sum_{k=1}^{n} [b]_{jk} \cdot c_k\right).\end{aligned}$$

And using the subdistributivity in interval arithmetic we have

$$\begin{aligned}([A]\cdot([B]\cdot c))_i &\subseteq \sum_{j=1}^{n}\sum_{k=1}^{n}[a]_{ij}\cdot[b]_{jk}\cdot c_k\\ &= \sum_{k=1}^{n}\left(\sum_{j=1}^{n}[a]_{ij}\cdot[b]_{jk}\right)\cdot c_k\\ &= \sum_{k=1}^{n}([A]\cdot[B])_{ik}\cdot c_k\\ &= ([A]\cdot[B]\cdot c)_i.\end{aligned}$$

Applying this property several times to (3.15) and using (3.16) we get

$$IGA([A],b)\subseteq IGA([A])\cdot b.$$

□

3.2 Quadratic Divergence of the Newton Method

For an interval vector $[x]=([x]_i)\subseteq[x]^0\subset\mathbb{R}^n$ and F', the inclusion function for the Fréchet-derivative f' of a mapping $f:[x]\subseteq[x]^0\subset\mathbb{R}^n\to\mathbb{R}^n$ over $[x]^0$, we define the interval Newton operator as:

$$N[x]=[n^1,n^2]=\check{x}-IGA\left(F'([x]),f(\check{x})\right),$$

where $\check{x}\in[x]$. Using $N[x]$ we define the interval Newton method for a given interval vector $[x]^0$ as follows:

$$[x]^{k+1}=N[x]^k\cap[x]^k,\qquad k=0,1,2,...,$$

where the intersection is taken componentwise.

Theorem 1. *Let there be given an interval vector $[x]^0\subset\mathbb{R}^n$, a mapping $f:[x]^0\subset\mathbb{R}^n\to\mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$.*
We assume that $IGA\left(F'\left([x]^0\right)\right)$ exists, $\rho(A)<1$ (ρ spectral radius), where $A=\left|I-IGA\left(F'\left([x]^0\right)\right)\cdot F'\left([x]^0\right)\right|$, and

$$d\left(F'([x])\right)<C\cdot\|d([x])\|,\qquad \text{for all } [x]\subseteq[x]^0, C\geq 0.$$

Then

$$\begin{aligned}x^2-n^1&\leq O\left(\|d([x])\|^2\right)\cdot e+A^2\cdot f\left(x^2\right),\\ n^2-x^1&\leq O\left(\|d([x])\|^2\right)\cdot e-A^1\cdot f\left(x^1\right)\end{aligned}$$

are valid for some $A^1,\ A^2\in IGA\left(F'([x])\right)$.
Moreover if $[x]^0$ does not contain any zero of $f(x)$ then there is $\varepsilon>0$ and at least one index $i\in\{1,\ldots,n\}$ such that

$$(x^2-n^1)_i<0\qquad or\qquad (n^2-x^1)_i<0$$

once $d([x])<\varepsilon\cdot e$.

The proof of this theorem and further details on the divergence of the Interval-Newton method can be found in [3]. The Newton method possesses the quadratic convergence. Theorem 1 shows that the Newton method also possesses a quadratic behavior in the case of divergence. It describes what kind of divergence we should expect if there is no zero of the given function, i.e. how fast the iteration terminates resulting in an empty set. Unlike convergence, where we consider all the components of the interval vector, in the case of divergence it is enough that intersection results an empty set at least for one component.

Lemma 16. *Let the assumption of Theorem 1 hold and assume that* $[x]^k$ *and* $[x]^{k+1}$ *exist. Then*

$$m[x]^k \notin [x]^{k+1}.$$

Proof. See [1], Lemma 6. □

This lemma says that at least one component of the diameter $d\left([x]^k\right)$ is more than halved in the $(k+1)$th step.

3.3 The Simplified Newton Method

For an interval vector $[x] = ([x]_i) \subseteq [x]^0 \subseteq \mathbb{IR}^n$ and an inclusion function F' for the Fréchet-derivative f' of a mapping $f : [x] \subseteq [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ over $[x]^0$ we define the simplified interval Newton operator as:

$$SN[x] = \check{x} - IGA\left(F'\left([x]^0\right), f(\check{x})\right),$$

where $\check{x} \in [x]$. Using $SN[x]$ we define the simplified Interval-Newton method for a given interval vector $[x]^0$ as follows:

$$[x]^{k+1} = SN[x]^k \cap [x]^k \qquad k = 0, 1, 2, ...,$$

where the intersection is taken componentwise.
We denote the lower and upper bounds of $SN[x]$ by n^1 and n^2.
The following Theorem gives sufficient conditions for the convergence of the simplified Newton method.

Theorem 2. *Let there be given an interval vector* $[x]^0 \subset \mathbb{R}^n$*, a mapping* $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ *and an inclusion function* F' *for the Fréchet derivative* f' *of* f *over* $[x]^0$*. We assume that* $IGA\left(F'\left([x]^0\right)\right)$ *exists and that* $\rho(A) < 1$*, where* $A = \left|I - IGA\left(F'\left([x]^0\right)\right) \cdot F'\left([x]^0\right)\right|$*. Then*
a) if $[x]^0$ *contains a zero* x^* *of* $f(x)$ *then* $x^* \in [x]^k$ *and* $\lim_{k\to\infty}[x]^k = x^*$*;*
b) if $[x]^0$ *does not contain any zero then there is* k_0 *such that* $SN[x]^{k_0} \cap [x]^{k_0} = \varnothing$*.*

The proof of this theorem can be found in [1].

3.4 Linear divergence of the Simplified Newton Method

We have loosened the operator $N[x]$ by substituting $F'([x]^0)$ for $F'([x])$. Under certain assumptions we still get divergence. But the speed of divergence is loosened from quadratic to linear. The following theorem gives sufficient conditions for linear divergence.

Theorem 3. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$.*
We assume that $IGA\left(F'\left([x]^0\right)\right)$ exists and that $\rho(A) < 1$,
where $A = \left|I - IGA\left(F'\left([x]^0\right)\right) \cdot F'\left([x]^0\right)\right|$.
Then for any $[x] \subseteq [x]^0$ with $[x] = [x^1, x^2]$, $SN[x] = [n^1, n^2]$

$$x^2 - n^1 \leq O(\|d([x])\|) \cdot e + A^2 \cdot f\left(x^2\right), \tag{3.26}$$

$$n^2 - x^1 \leq O(\|d([x])\|) \cdot e - A^1 \cdot f\left(x^1\right) \tag{3.27}$$

are valid for some $A^1,\ A^2 \in IGA\left(F'\left([x]^0\right)\right)$.
Moreover if $[x]^0$ does not contain any zero of f then there is $\varepsilon > 0$ and at least one index $i \in \{1, \ldots, n\}$ such that

$$\left(x^2 - n^1\right)_i < 0 \qquad or \qquad \left(n^2 - x^1\right)_i < 0 \tag{3.28}$$

once $d([x]) > \varepsilon \cdot e$.

Proof. Using Lemma 15 we have

$$\begin{aligned} SN[x] &= \check{x} - IGA\left(F'\left([x]^0\right), f(\check{x})\right) \\ &\subseteq \check{x} - IGA\left(F'\left([x]^0\right)\right) \cdot f(\check{x}) := \widetilde{SN}[x] = [\tilde{n}^1, \tilde{n}^2]. \end{aligned}$$

The inclusion $SN[x] \subseteq \widetilde{SN}[x]$ is equivalent to

$$\tilde{n}^1 \leq n^1 \leq n^2 \leq \tilde{n}^2. \tag{3.29}$$

From the definition of $\widetilde{SN}[x]$ and Lemma 3 it follows that

$$\begin{aligned} \tilde{n}^1 &= \check{x} - A^2 \cdot f(\check{x}) \\ \tilde{n}^2 &= \check{x} - A^1 \cdot f(\check{x}) \end{aligned} \tag{3.30}$$

for some $A^1,\ A^2 \in IGA\left(F'\left([x]^0\right)\right)$. For the right end point x^2 of the vector $[x]$, using (3.29), (3.30) and (3.7), we can write the following:

$$\begin{aligned} x^2 - n^1 &\leq x^2 - \tilde{n}^1 \\ &= x^2 - \check{x} + A^2 \cdot f(\check{x}) \\ &= x^2 - \check{x} + A^2\left(f\left(x^2\right) + \delta f\left(\check{x}, x^2\right) \cdot \left(\check{x} - x^2\right)\right) \\ &= \left(I - A^2 \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right) + A^2 \cdot f\left(x^2\right). \end{aligned} \tag{3.31}$$

Because of (3.8) and Lemma 6, the matrix $\delta f\left(\check{x}, x^2\right)$ is nonsingular and $\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} \in IGA\left(F'\left([x]^0\right)\right)$.
Therefore we have

$$\left|\left(I - A^2 \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = \left|\left(\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} - A^2\right) \cdot \delta f\left(\check{x}, x^2\right) \cdot \left(x^2 - \check{x}\right)\right|.$$

Since $\left|\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} - A^2\right| \leq d\left(IGA\left(F'\left([x]^0\right)\right)\right)$ and $\left|\delta f\left(\check{x}, x^2\right)\right| \leq \left|F'\left([x]^0\right)\right|$ we get:

$$\left|\left(I - A^2 \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| \leq d\left(IGA\left(F'\left([x]^0\right)\right)\right) \cdot \left|F'\left([x]^0\right)\right| \cdot d([x]).$$

Applying Lemma 12 to $d\left(IGA\left(F'\left([x]^0\right)\right)\right)\cdot\left|F'\left([x]^0\right)\right|\cdot d([x])$ from the last inequality we get:

$$\left|\left(I-A^2\cdot\delta f\left(\check{x},x^2\right)\right)\cdot\left(x^2-\check{x}\right)\right|=O(\|d([x])\|)\cdot e. \tag{3.32}$$

From (3.31) and (3.32) we get (3.26):

$$x^2-n^1\leq O(\|d([x])\|)\cdot e+A^2\cdot f\left(x^2\right).$$

In the same manner we can obtain (3.27).
From (3.7) we get:

$$\begin{aligned}f(\check{x})&=f\left(x^1\right)+\delta f\left(\check{x},x^1\right)\cdot\left(\check{x}-x^1\right),\\ f(\check{x})&=f\left(x^2\right)+\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right).\end{aligned}$$

Therefore

$$\begin{aligned}A^1\cdot f(\check{x})&=A^1\cdot f\left(x^1\right)+A^1\cdot\delta f\left(\check{x},x^1\right)\cdot\left(\check{x}-x^1\right)\\ A^2\cdot f(\check{x})&=A^2\cdot f\left(x^2\right)+A^2\cdot\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right).\end{aligned} \tag{3.33}$$

Here

$$\begin{aligned}A^1\cdot\delta f\left(\check{x},x^1\right)\cdot\left(\check{x}-x^1\right)&\to 0 \qquad \text{for} \qquad d([x])\to 0,\\ A^2\cdot\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right)&\to 0 \qquad \text{for} \qquad d([x])\to 0.\end{aligned} \tag{3.34}$$

According to Lemma 10 the matrix $IGA\left(F'\left([x]^0\right)\right)$ does not contain singular point matrices. Moreover, $f(x)\neq 0$, for all $x\in[x]^0$ and $f(x)$ is continuous by the assumption (see page 3). Then from Lemma 13 it follows that there exist $c>0$ and $\varepsilon(c)>0$ such that

$$|A\cdot f(x)|_{i(x)}>c \quad \text{for all } A\in IGA\left(F'\left([x]^0\right)\right),\ x\in[x]^0, \tag{3.35}$$

and

$$\begin{aligned}sign\left(\left(A^1\cdot f(x^1)\right)_{i(x^1)}\right)&=sign\left(\left(A^2\cdot f(x^2)\right)_{i(x^1)}\right),\\ &\text{for all } A^1,A^2\in IGA\left(F'\left([x]^0\right)\right),x^1,x^2\in[x]^0,\end{aligned} \tag{3.36}$$

once $|x^1-x^2|<\varepsilon(c)\cdot e$.
Define $\alpha>0$ such that in (3.26) the term $O(\|d([x])\|)$ is less than $\alpha\cdot\|d([x])\|$. Now let $\|d([x])\|$ be small enough such that in (3.34) we have

$$\begin{aligned}\left|A^1\cdot\delta f\left(\check{x},x^1\right)\cdot\left(\check{x}-x^1\right)\right|&<c/2\cdot e,\\ \left|A^2\cdot\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right)\right|&<c/2\cdot e\end{aligned} \tag{3.37}$$

and also

$$\begin{aligned}\alpha\cdot\|d([x])\|&<c/2,\\ \|d([x])\|&<\varepsilon(c).\end{aligned} \tag{3.38}$$

Then from (3.33), (3.35), (3.37) and (3.38) for $i_0=i(\check{x})$ we get:

$$\begin{aligned}\left|\left(A^1\cdot f\left(x^1\right)\right)_{i_0}\right|&>c/2\\ \left|\left(A^2\cdot f\left(x^2\right)\right)_{i_0}\right|&>c/2\end{aligned} \tag{3.39}$$

Since $\alpha\cdot\|d([x])\|<c/2$, from (3.26), (3.27), (3.36) and (3.39) we get

$$(x^2-n^1)_{i_0}<0 \qquad or \qquad (n^2-x^1)_{i_0}<0.$$

So (3.28) has been proven together with part b) of Theorem 3. □

From Theorem 3 and Lemma 11 it follows that if the starting interval vector does not contain a zero of the function then the Simplified Newton method **"diverges linearly"**. Sometimes it is preferrable to choose a linear divergence of the simplified Newton method over a quadratic divergence of the Newton method. It can happen if the evaluation of the Hessian in the Newton method is more expensive than several iterations of the simplified Newton method. And therefore the simplified Newton method diverges faster than the usual Newton method.

3.5 The Krawczyk Method

Another interval method for solving systems of nonlinear equations is the Krawczyk method. Below we introduce the Krawczyk method and give sufficient conditions for convergence and divergence of this method.
For an interval vector $[x] = ([x]_i) \subseteq [x]^0 \subset \mathbb{R}^n$ and an inclusion function F' for the Fréchet-derivative f' of a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ over $[x]$ we define the Krawczyk operator as:

$$K([x], \check{x}) = \check{x} - Y \cdot f(\check{x}) + (I - Y \cdot F'([x])) \cdot ([x] - \check{x}),$$

where $Y = (m(F'([x])))^{-1}$ and $\check{x} \in [x]$.
Using $K([x], \check{x})$, we define the Krawczyk method for a given interval vector $[x]^0$ as follows:

$$[x]^{k+1} = K([x]^k, \check{x}^k) \cap [x]^k, \qquad k = 0, 1, 2, \dots ,$$

where the intersection is taken componentwise.
We denote by k^1, k^2 the lower and upper bounds of $K([x], \check{x})$.

Theorem 4. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f. We assume that $IGA\left(F'\left([x]^0\right)\right)$ exists and that $\rho(2 \cdot A) < 1$ where $A = \left|I - IGA\left(F'\left([x]^0\right)\right) \cdot F'\left([x]^0\right)\right|$. Then*
a) if $[x]^0$ contains a zero x^* of $f(x)$ then $x^* \in [x]^k$ and $\lim_{k\to\infty}[x]^k = x^*$;
b) if $[x]^0$ does not contain any zero then there is k_0 such that $K[x]^{k_0} \cap [x]^{k_0} = \varnothing$.
If one chooses $\check{x}^k$ to be the center of $[x]^k$ then the condition $\rho(2 \cdot A) < 1$ can be replaced by $\rho(A) < 1$.

The proof can be found in [41], by observing that from our assumption, $F'([x])$ is a Lipschitz matrix in the sense of [41], p.171 which is strongly regular (see Theorem of [41], p.116).

Theorem 5. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$.*
We assume that $IGA\left(F'\left([x]^0\right)\right)$ exists and $\rho(2 \cdot A) < 1$,
where $A = \left|I - IGA\left(F'\left([x]^0\right)\right) \cdot F'\left([x]^0\right)\right|$, and

$$d\left(F'([x])\right) < \|d([x])\| \cdot C, \quad [x] \subseteq [x]^0, \quad C \geq 0. \tag{3.40}$$

Then

$$x^2 - k^1 \leq O\left(\|d([x])\|^2\right) \cdot e + Y \cdot f\left(x^2\right), \tag{3.41}$$

$$k^2 - x^1 \leq O\left(\|d([x])\|^2\right) \cdot e - Y \cdot f\left(x^1\right) \tag{3.42}$$

are valid for $Y = (m(F'([x])))^{-1}$.
If $[x]^k \neq \varnothing$ for all k then

$$d([x]^{k+1}) = O(\|d([x]^k)\|^2) \cdot e. \tag{3.43}$$

Moreover, if $[x]^0$ *does not contain any zero of* $f(x)$ *then there is* $n_0 \geq 0$ *and at least one index* $i \in \{1, \ldots, n\}$ *such that*

$$(x^2 - k^1)_i < 0 \qquad or \qquad (k^2 - x^1)_i < 0$$

for $[x]^{n_0} = [x^1, x^2]$.
If one chooses $\check{x}^k$ *to be the center of* $[x]^k$ *then the condition* $\rho(2 \cdot A) < 1$ *can be replaced by* $\rho(A) < 1$.

We will give the proof of this theorem together with the proof of Theorem 7, since the proves differ only slightly.

3.6 The Simplified Krawczyk Method

For an interval vector $[x] = ([x]_i) \subseteq [x]^0 \subset \mathbb{R}^n$ and an inclusion function F' for the Fréchet-derivative f' of a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ over $[x]^0$ we define the simplified Krawczyk operator as:

$$SK([x], \check{x}) = \check{x} - Y \cdot f(\check{x}) + \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot ([x] - \check{x}),$$

where $Y = \left(m(F'\left([x]^0\right))\right)^{-1}$, and $\check{x} \in [x]$.
Using $SK([x], \check{x})$, we define the simplified Krawczyk method for a given interval vector $[x]^0$ as follows:

$$[x]^{k+1} = SK\left([x]^k, \check{x}^k\right) \cap [x]^k, \qquad k = 0, 1, 2, ...,$$

where the intersection is taken componentwise.
We denote by k^1, k^2 the lower and upper bounds of $SK([x], \check{x})$.

Theorem 6. *Let there be given an interval vector* $[x]^0 \subset \mathbb{R}^n$*, a mapping* $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ *and an inclusion function* F' *for the Fréchet derivative* f' *of* f *over* $[x]^0$*. We assume that* $IGA(F'([x]^0))$ *exists. Suppose that* $\rho(2 \cdot A) < 1$*, where* $A = \left|I - \left(m(F'\left([x]^0\right))\right)^{-1} \cdot F'\left([x]^0\right)\right|$.
Then
a) if $[x]^0$ *contains a zero* x^* *of* $f(x)$ *then* $x^* \in [x]^k$ *and* $\lim_{k\to\infty}[x]^k = x^*$*;*
b) if $[x]^0$ *does not contain any zero then there is* k_0 *such that* $SK\left([x]^{k_0}, \check{x}^{k_0}\right) \cap [x]^{k_0} = \varnothing$.
If one chooses $\check{x}^k$ *to be the center of* $[x]^k$ *then the condition* $\rho(2 \cdot A) < 1$ *can be replaced by* $\rho(A) < 1$.

Proof. First we prove a).
Denote by x^* a zero of f. Suppose $x^* \in [x]^k$. Then

$$SK\left([x]^k, \check{x}^k\right) - x^* = \check{x}^k - x^* - Y \cdot f\left(\check{x}^k\right) + \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right). \tag{3.44}$$

Using (3.7) for $\check{x}^k$ and x^* we have:

$$f\left(\check{x}^k\right) = \delta f\left(\check{x}^k, x^*\right) \cdot \left(\check{x}^k - x^*\right). \tag{3.45}$$

From (3.44) and (3.45) it follows that:

$$\begin{aligned} SK\left([x]^k, \check{x}^k\right) - x^* = &- \left(I - Y \cdot \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right) \\ &+ \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right). \end{aligned} \tag{3.46}$$

According to (3.8) we have $\delta f\left(\check{x}^k, x^*\right) \in F'\left([x]^0\right)$ and therefore

$$\left(I - Y \cdot \delta f\left(\check{x}^k, x^*\right)\right) \in \left(I - Y \cdot F'\left([x]\right)\right). \tag{3.47}$$

Hence

$$\left(I - Y \cdot \delta f\left(\check{x}^k, x^*\right) \cdot \left(x^* - \check{x}^k\right)\right) \in \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right).$$

From this it follows that

$$0 \in \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right) - \left(I - Y \cdot \delta f\left(m([k]^k), x^*\right)\right) \cdot \left(x^* - \check{x}^k\right)$$

or, by (3.46),

$$0 \in SK\left([x]^k, \check{x}^k\right) - x^*$$

or

$$x^* \in SK\left([x]^k, \check{x}^k\right).$$

And since $[x]^{k+1} = [x]^k \cap SK\left([x]^k, \check{x}^k\right)$ also $x^* \in [x]^{k+1}$ is true.
By assumption $x^* \in [x]^0$ and therefore by induction $x^* \in [x]^k$ for each k.
From (3.46) it follows that

$$\begin{aligned}\left|SK\left([x]^k, \check{x}^k\right) - x^*\right| &\leq \left|\left(I - Y \cdot \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right)\right| \\ &\quad + \left|\left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right)\right| \\ &\leq \left|I - Y \cdot \delta f\left(\check{x}^k, x^*\right)\right| \cdot \left|x^* - \check{x}^k\right| \\ &\quad + \left|I - Y \cdot F'\left([x]^0\right)\right| \cdot \left|[x]^k - \check{x}^k\right|.\end{aligned} \tag{3.48}$$

Using (3.47) and the fact that $x^* - \check{x}^k \in [x]^k - \check{x}^k$ we get

$$\left|SK\left([x]^k, \check{x}^k\right) - x^*\right| \leq 2 \cdot A \cdot \left|[x]^k - \check{x}^k\right|.$$

Using

$$\begin{aligned}q\left([x]^k, x^*\right) &= \left|[x]^k - x^*\right| \\ q\left(SK\left([x]^k, \check{x}^k\right), x^*\right) &= \left|SK\left([x]^k, \check{x}^k\right) - x^*\right|\end{aligned}$$

this can be written as

$$q\left(SK\left([x]^k, \check{x}^k\right), [x]^*\right) \leq 2 \cdot A \cdot q\left([x]^k, x^*\right).$$

Because $x^* \in SK\left([x]^k, \check{x}^k\right) \cap [x]^k = [x]^{k+1} \subseteq SK\left([x]^k, \check{x}^k\right)$ it also holds that

$$q\left([x]^{k+1}, x^*\right) \leq q\left(SK\left([x]^k, \check{x}^k\right), x^*\right) \leq 2 \cdot A \cdot q\left([x]^k, x^*\right)$$

and therefore

$$q\left([x]^{k+1}, x^*\right) \leq (2 \cdot A)^{k+1} \cdot q\left([x]^0, x^*\right).$$

Since $\rho(2 \cdot A) < 1$, from Lemma 14 it follows that $\lim_{k\to\infty}[x]^k = x^*$. If one chooses $\check{x}^k$ to be the center of $[x]^k$ in (3.48) we would have $|x^* - \check{x}^k| \leq \frac{1}{2}d([x]^k)$ and $|[x]^k - \check{x}^k| = \frac{1}{2}d([x]^k)$ and

therefore the condition $\rho(A) < 1$ is enough.
Now we prove b).
Assume that for all $k \geq 0$ the intersections $SK\left([x]^k, \check{x}^k\right) \cap [x]^k$ are not empty. Then it holds that

$$[x]^0 \supseteq [x]^1 \supseteq \ldots [x]^k \supseteq [x]^{k+1} \ldots ,$$

from which it follows that the sequence is converging to an interval vector $[z]^*$. We now consider the sequence $\{\check{x}^k\}_{k=0}^{\infty}$. There exists a convergent subsequence $\{\check{x}^{k_n}\}_{n=0}^{\infty}$. Denote $m^* = \lim_{n\to\infty} \check{x}^{k_n}$. For elements of the sequence $\{\check{x}^{k_n}\}_{n=0}^{\infty}$ it holds that $\check{x}^{k_n} \in [x]^{k_n}$. From this remark it follows that beside of $\lim_{n\to\infty}[x]^{k_n} = [z]^*$ we also have $m^* \in [z]^*$. Using the continuity of operations from

$$\begin{aligned} SK\left([x]^k, \check{x}^k\right) &= \check{x}^k - Y \cdot f\left(\check{x}^k\right) + \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([x]^k - \check{x}^k\right), \\ [x]^{(k+1)} &= SK\left([x]^k, \check{x}^k\right) \cap [x]^k \end{aligned}$$

we get the relations

$$\begin{aligned} [s]^* &= m^* - Y \cdot f\left(m^*\right) + \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([z]^* - m^*\right), \\ [z]^* &= [s]^* \cap [z]^*, \end{aligned} \tag{3.49}$$

where $[s]^* = \lim_{n\to\infty} SK\left([x]^{k_n}, \check{x}^{k_n}\right) = SK\left([z]^*, m[z]^*\right)$.
From the first equation we get

$$\begin{aligned} \left|[s]^* - m^* + Y \cdot f\left(m^*\right)\right| &= \left|\left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left([z]^* - m^*\right)\right| \\ &\leq \left|\left(I - Y \cdot F'\left([x]^0\right)\right)\right| \cdot \left|\left([z]^* - m^*\right)\right| \end{aligned}$$

and then

$$\left|[s]^* - m^* + Y \cdot f\left(m^*\right)\right| \leq A \cdot \left|[z]^* - m^*\right|. \tag{3.50}$$

$0 \in \left([z]^* - m^*\right)$ implies $\left|[z]^* - m^*\right| \leq d\left([z]^* - m^*\right) = d\left([z]^*\right)$. The inequality $\frac{1}{2} \cdot d([x]) \leq |[x]|$ holds for all $[x] \in \mathbb{IR}^n$. From the last two remarks and (3.50) we get

$$\begin{aligned} \frac{1}{2} \cdot d\left([s]^*\right) &= \frac{1}{2} \cdot d\left([s]^* - m^* + Y \cdot f\left(m^*\right)\right) \\ &\leq \left|[s]^* - m^* + Y \cdot f\left(m^*\right)\right| \\ &\leq A \cdot \left|[z]^* - m^*\right| \\ &\leq A \cdot d\left([z]^*\right) \end{aligned} \tag{3.51}$$

or

$$d\left([s]^*\right) \leq 2 \cdot A \cdot d\left([z]^*\right)$$

and using the second equation of (3.49) this can be written as

$$d\left([s]^*\right) \leq 2 \cdot A \cdot d\left([s]^*\right),$$

from which we get $d\left([s]^*\right) \leq (2 \cdot A)^n \cdot d\left([s]^*\right)$ for all n. Then $d\left([s]^*\right) = 0$, since $\rho(2 \cdot A) < 1$. Therefore $[s^*]$ and $[z^*]$ are point vectors and $s^* = z^* = m^*$. Then from the first equation of (3.49) it follows that

$$m^* = m^* - Y \cdot f\left(m^*\right) + \left(I - Y \cdot F'\left([x]^0\right)\right) \cdot \left(m^* - m^*\right),$$

from which we get

$$0 = Y \cdot f\left(m^*\right).$$

Since Y is nonsingular it follows that $f\left(m^*\right) = 0$. But this contradicts the fact that $f(x)$ has no zeros in $[x]^0$. Hence there is k_0 such that $SK\left([x]^{k_0}, \check{x}^{k_0}\right) \cap [x]^{k_0} = \varnothing$ If one chooses $\check{x}^k$ to be the center of $[x]^k$ from (3.51) we would have $d\left([s]^*\right) \leq A \cdot d\left([z]^*\right)$ and therefore the condition $\rho(A) < 1$ is sufficient. □

Theorem 7. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$. Suppose that $\rho(2 \cdot A) < 1$, where $A = \left|I - \left(m(F'\left([x]^0\right))\right)^{-1} \cdot F'\left([x]^0\right)\right|$. Then*

$$x^2 - k^1 \leq O(\|d([x])\|) \cdot e + Y \cdot f\left(x^2\right), \tag{3.52}$$

$$k^2 - x^1 \leq O(\|d([x])\|) \cdot e - Y \cdot f\left(x^1\right) \tag{3.53}$$

are valid for $Y = \left(m(F'\left([x]^0\right))\right)^{-1}$.
If $[x]^k \neq \varnothing$ for all k then

$$d([x]^{k+1}) = O(\|d([x]^k)\|) \cdot e. \tag{3.54}$$

Moreover, if $[x]^0$ does not contain any zero of $f(x)$ then there is $n_0 \geq 0$ and at least one index $i \in \{1, \ldots, n\}$ such that

$$(x^2 - k^1)_i < 0 \qquad or \qquad (k^2 - x^1)_i < 0$$

for $[x]^{n_0} = [x^1, x^2]$.
If one chooses $\check{x}^k$ to be the center of $[x]^k$ then the condition $\rho(2 \cdot A) < 1$ can be replaced by $\rho(A) < 1$.

Proof. Here we prove both Theorem 5 and Theorem 7. For the proof of Theorem 5 we take $[z] = [x]^0$ and for the proof of Theorem 7 $[z] = [x]$.
Denote $(I - Y \cdot F'([z])) \cdot ([x] - \check{x})$, by $[r]$ and the lower and upper bounds of $[r]$, as usually, by r^1 and r^2.
Then

$$\begin{aligned} k^1 &= \check{x} - Y \cdot f(\check{x}) + r^1, \\ k^2 &= \check{x} - Y \cdot f(\check{x}) + r^2 \end{aligned} \tag{3.55}$$

are obvious.
For the right end point x^2 of the interval vector $[x]$ using (3.55) we can write the following

$$x^2 - k^1 = x^2 - \check{x} + Y \cdot f(\check{x}) - r^1.$$

Using (3.7) for $\check{x}$ and x^2 we have $f(\check{x}) = f\left(x^2\right) + \delta f\left(\check{x}, x^2\right) \cdot \left(\check{x} - x^2\right)$ and therefore

$$\begin{aligned} x^2 - k^1 &= x^2 - \check{x} + Y\left(f\left(x^2\right) + \delta f\left(\check{x}, x^2\right) \cdot \left(\check{x} - x^2\right)\right) - r^1 \\ &= \left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right) + Y \cdot f\left(x^2\right) - r^1. \end{aligned} \tag{3.56}$$

Because of (3.8) and Lemma 6 the matrix $\delta f\left(\check{x}, x^2\right)$ is nonsingular and $\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} \in IGA\left(F'([z])\right)$. Therefore we have

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = \left|\left(\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} - Y\right) \cdot \delta f\left(\check{x}, x^2\right) \cdot \left(x^2 - \check{x}\right)\right|$$

and since $\left|\left(\delta f\left(\check{x}, x^2\right)\right)^{-1} - Y\right| \leq d\left(IGA\left(F'([z])\right)\right)$ and $\left|\delta f\left(\check{x}, x^2\right)\right| \leq \left|F'\left([x]^0\right)\right|$ we get:

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right)\left(x^2 - \check{x}\right)\right| \leq d\left(IGA\left(F'([x])\right)\right) \cdot \left|F'\left([x]^0\right)\right| \cdot d([x]).$$

For Theorem 5 taking into account (3.40) and applying Lemma 9 we obtain

$$d(IGA(F'([z]))) \leq O(\|d([z]\|) \cdot M, \quad \text{with } [z] = [x]$$

and then using Lemma 12 we get

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = O(\|d([x])\|) \cdot \|d([x])\| \cdot e = O(\|d([x])\|^2) \cdot e. \tag{3.57}$$

For Theorem 7 we have

$$d(IGA(F'([z]))) \leq C_1$$

where C_1 is a constant point matrix and therefore using Lemma 12 we get

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = O(\|d([x])\|) \cdot e. \tag{3.58}$$

Now we consider $[r]$:

$$\begin{aligned} |[r]| &= |(I - Y \cdot F'([z])) \cdot ([x]) - \check{x})| \\ &\leq |I - Y \cdot F'([z])| \cdot |[x] - \check{x}| \\ &\leq |Y| \cdot \left|Y^{-1} - F'([z])\right| \cdot |[x] - \check{x}| \\ &\leq |Y| \cdot \left|Y^{-1} - F'([z])\right| \cdot d([x]). \end{aligned} \tag{3.59}$$

Since $Y^{-1} = m(F'([z])) \in F'([z])$ we have

$$\left|Y^{-1} - F'([z])\right| \leq d(F'([z])).$$

For Theorem 5 from (3.40) we have $d(F'([z])) = O(\|d([z])\|) \cdot e$ and therefore

$$\left|Y^{-1} - F'([z])\right| \leq d(F'([z])) = O(\|d([z])\|) \cdot E, \quad \text{with } [z] = [x]. \tag{3.60}$$

For Theorem 7 the matrix $d(F'([z]))$ is constant:

$$d(F'([z])) = C_2. \tag{3.61}$$

From Lemma 6 the relation $Y \in IGA\left(F'([z])\right)$ holds and therefore

$$|Y| \leq C, \tag{3.62}$$

where C is a constant point matrix. Using (3.59), (3.60) and (3.62) for Theorem 5 we have the following

$$|[r]| = O\left(\|d([x])\|^2\right) \cdot e. \tag{3.63}$$

For Theorem 5 using (3.59) and (3.62) we have

$$|[r]| = O\left(\|d([x])\|\right) \cdot e, \tag{3.64}$$

and from (3.56), (3.57) and (3.63) we get

$$x^2 - k^1 \leq O\left(\|d([x])\|^2\right) \cdot e + Y \cdot f\left(x^2\right).$$

For Theorem 7 from (3.56), (3.58) and (3.64) we get

$$x^2 - k^1 \leq O\left(\|d([x])\|\right) \cdot e + Y \cdot f\left(x^2\right).$$

So we have proved (3.41) and (3.52). In the same manner we can get (3.42) and (3.53). From (3.41) and (3.42) it follows:

$$\begin{aligned} d([x]) + d(K([x], \check{x})) &= (x^2 - x^1) + (k^2 - k^1) = (x^2 - k^1) + (k^2 - x^1) \\ &\leq O(\|d([x])\|^2) \cdot e + Y \cdot (f(x^2) - f(x^1)). \end{aligned}$$

And from (3.52) and (3.53) it follows:

$$d([x]) + d(SK([x], \check{x})) \leq O(\|d([x])\|) \cdot e + Y \cdot (f(x^2) - f(x^1)).$$

We consider $Y \cdot (f(x^2) - f(x^1))$. Using (3.8) we get:

$$Y \cdot (f(x^2) - f(x^1)) = Y \cdot \delta f(x^2, x^1) \cdot (x^2 - x^1).$$

For $\delta f(x^2, x^1)$ we have:

$$\delta f(x^2, x^1) = m(F'([z])) + Z$$

where $|Z| \leq d(F'([z]))$, since $\delta f(x^2, x^1)$, $m(F'([z])) \in F'([z])$. Then

$$\begin{aligned} Y \cdot (f(x^2) - f(x^1)) &= Y \cdot \left(m\left(F'([z])\right) \cdot d([x]) + Z \cdot d([x])\right) \\ &= d([x]) + Y \cdot Z \cdot d([x]) \\ &\leq d([x]) + |Y| \cdot |Z| \cdot d([x]) \\ &\leq d([x]) + |Y| \cdot d(F'([z])) \cdot d([x]). \end{aligned} \tag{3.65}$$

For Theorem 5 using (3.40) and (3.62) from (3.65) we get:

$$Y \cdot (f(x^2) - f(x^1)) \leq d([x]) + O(\|d([x])\|^2) \cdot e.$$

Then

$$d([x]) + d(K([x], \check{x})) \leq d([x]) + O(\|d([x])\|^2) \cdot e$$

or

$$d(K([x], \check{x})) \leq O(\|d([x])\|^2) \cdot e.$$

For Theorem 7 from (3.65), (3.62) and (3.61) we have

$$Y \cdot (f(x^2) - f(x^1)) \leq d([x]) + O(\|d([x])\|) \cdot e$$

and further

$$d(K([x],\check{x})) \leq O(\|d([x])\|) \cdot e.$$

If $[x]^{k+1} \neq \varnothing$ then $d([x]^{k+1}) \leq d(K([x]^k,\check{x}^k))$. Then we have

$$d([x]^{k+1}) \leq O(\|d([x]^k)\|^2) \cdot e$$

for Theorem 5 and

$$d([x]^{k+1}) \leq O(\|d([x]^k)\|) \cdot e$$

for Theorem 7. We have proved (3.43) and (3.54).
To finish the proof assume now that f does not have a zero in $[x]^0$.
All matrices $(m(F'([z])))^{-1}$, $[z] \subseteq [x]^0$, are nonsingular. Also $f(x) \neq 0$, for all $x \in [x]^0$ and $f(x)$ is continuous by the assumption. From Lemma 13 we have that there exist $c > 0$ and $\varepsilon(c) > 0$ such that:

$$\left|(m(F'([z])))^{-1} \cdot f(x)\right|_{i(x)} > c \quad \text{for all } x \in [x]^0 \tag{3.66}$$

and

$$sign\left(\left((m(F'([z])))^{-1} \cdot f(x^1)\right)_{i(x^1)}\right) = sign\left(\left((m(F'([z])))^{-1} \cdot f(x^2)\right)_{i(x^1)}\right) \quad \text{for } x_1, \quad x_2 \in [x], \tag{3.67}$$

once $|x^1 - x^2| < \varepsilon(c) \cdot e$.
From the inclusion isotonicity in interval arithmetic we have

$$\begin{aligned} &F'([x]^k) \subseteq F'([x]^0), \\ &Y \in IGA(F'([x]^0)). \end{aligned} \tag{3.68}$$

We estimate $d([x]^{k+1})$. For Theorem 5 we have

$$\begin{aligned} d\left([x]^{k+1}\right) &\leq d(K([x]^k,\check{x}^k)) \\ &= d\left(\check{x}^k - Y \cdot f(\check{x}^k) + \left(I - Y \cdot F'([x]^k)\right) \cdot \left([x]^k - \check{x}^k\right)\right) \\ &= d\left(\left(I - Y \cdot F'([x]^k)\right) \cdot \left([x]^k - \check{x}^k\right)\right) \\ &\leq 2 \cdot \left|\left(I - Y \cdot F'([x]^k)\right) \cdot \left([x]^k - \check{x}^k\right)\right| \\ &\leq 2 \cdot \left|I - Y \cdot F'([x]^k)\right| \cdot \left|[x]^k - \check{x}^k\right|. \end{aligned}$$

Using (3.68) we get $d\left([x]^{k+1}\right) \leq 2 \cdot A \cdot \left|[x]^k - \check{x}^k\right|$. For Theorem 7 we have

$$\begin{aligned} d\left([x]^{k+1}\right) &\leq d(SK([x]^k,\check{x}^k)) \\ &= d\left(\check{x}^k - Y \cdot f(\check{x}^k) + \left(I - Y \cdot F'([x]^0)\right) \cdot \left([x]^k - \check{x}^k\right)\right) \\ &= d\left(\left(I - Y \cdot F'([x]^0)\right) \cdot \left([x]^k - \check{x}^k\right)\right) \\ &\leq 2 \cdot \left|\left(I - Y \cdot F'([x]^0)\right) \cdot \left([x]^k - \check{x}^k\right)\right| \\ &\leq 2 \cdot \left|I - Y \cdot F'([x]^0)\right| \cdot \left|[x]^k - \check{x}^k\right| \\ &= 2 \cdot A \cdot \left|[x]^k - \check{x}^k\right|. \end{aligned}$$

Since $0 \in [x]^k - \check{x}^k$ we have

$$\left|[x]^k - \check{x}^k\right| \leq d\left([x]^k - \check{x}^k\right) = d\left([x]^k\right). \tag{3.69}$$

Therefore

$$d\left([x]^{k+1}\right) \leq 2 \cdot A \cdot d\left([x]^k\right).$$

and

$$d\left([x]^{k+1}\right) \leq (2 \cdot A)^{k+1} \cdot d\left([x]^0\right).$$

Suppose $[x]^k \neq \varnothing$ for all k. Then we have $\rho(2 \cdot A) < 1$ and therefore from Lemma 14 it follows $\lim_{k\to\infty} d([x]^k) = 0$. Let the term $O(\|d([x])\|^2)$ in (3.41) and (3.42) be bounded by $\alpha \cdot \|d([x])\|^2$, $\alpha \geq 0$. From $\lim_{k\to\infty} d([x]^k) = 0$ it follows that there exists $n_0 \geq 0$ such that

$$\alpha \cdot \|d([x]^{n_0})\|^2 < c,$$
$$d([x]^{n_0}) < \varepsilon(c) \cdot e.$$

Then for Theorem 5 from (3.41), (3.42), (3.66) and (3.67) for at least one index $i_0 \in \{1, \ldots, n\}$ either $(x^2 - k^1)_{i_0} < 0$ or $(k^2 - x^1)_{i_0} < 0$ takes place for $[x]^{n_0} = [x^1, x^2]$.

The same can be obtained for Theorem 7.

From Lemma 11 it follows that $[x]^{n_0+1} = [x]^{n_0} \cap [K([x]^{n_0}, m([x]^{n_0}) = \varnothing$. This contradicts $[x]^k \neq \varnothing$ for all k. Hence there is $[x]^{k_0+1} = \varnothing$ or from Lemma 11 either $(x^2 - k^1)_{i_0} < 0$ or $(k^2 - x^1)_{i_0} < 0$ takes place for $[x]^{n_0} = [x^1, x^2]$ and for at least one index $i_0 \in \{1, \ldots, n\}$. If one chooses $\check{x}^k$ to be the center of $[x]^k$ instead of (3.69) we would have $|[x]^k - \check{x}^k| = \frac{1}{2} d([x]^k)$ and therefore the condition $\rho(A) < 1$ is sufficient. □

3.7 The Modified Krawczyk Method (with LU Factorization)

Many systems of nonlinear equations come from the discretization of certain problems. Such systems possess a property which can be advantageous: they are sparse. The sparse structure allows us to simplify a time consuming part of the interval Krawczyk method - computation of the inverse matrix in the normal way. In this and the next subsections we will show how this can be done and will give sufficient conditions for modified methods to work. As it will be shown in the numerical results the use of the LU factorization can significantly decrease the time for large dimensions. But the number of classes of functions, to which modified methods can be applied, shrinks.

For an interval vector $[x] = ([x]_i) \subseteq [x]^0 \in \mathbb{R}^n$ and an inclusion function F' for the Fréchet-derivative f' of a mapping $f : [x]^0 \to \mathbb{R}^n$ over $[x]$, we define the modified Krawczyk operator as:

$$MK([x], \check{x}) = \check{x} - U \setminus \{L \setminus \{f(\check{x}) - (m(F'([x])) - F'([x])) \cdot ([x] - \check{x})\}\},$$

where L, U come from the LU factorization of $m(F'([x]))$, i.e. $m(F'([x])) = L \cdot U$, and $\check{x} \in [x]$. From (3.3), (3.5) and Lemma 1 it follows

$$\begin{aligned} MK([x], \check{x}) &= \check{x} - U \setminus\{\ L \setminus\{f(\check{x})\}\} + U \setminus\{\ L \setminus\{(m(F'([x])) - F'([x])) \cdot ([x] - \check{x})\}\} \\ &= \check{x} - Y \cdot f(\check{x}) + U \setminus\{\ L \setminus\{(m(F'([x])) - F'([x])) \cdot ([x] - \check{x})\}\} \end{aligned}$$

where $Y = (m(F'([x])))^{-1} = U^{-1} \cdot L^{-1}$.
We denote by k^1, k^2 the lower and upper bounds of $MK([x], \check{x})$.

Theorem 8. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$. Denote $A = |[U]^n| \cdot \ldots \cdot |[U]^1| \cdot |[L]^n| \cdot \ldots \cdot |[L]^1| \cdot d\left(F'\left([x]^0\right)\right)$.*
We assume that $IGA\left(F'\left([x]^0\right)\right)$ exists and that $\rho(2 \cdot A) < 1$, where matrices $[U]^i$ and $[L]^i$ are defined by (3.10) and (3.11). Then
a) if $[x]^0$ contains a zero x^* of f then $x^* \in [x]^k$ and $\lim_{k\to\infty}[x]^k = x^*$,
b) if $[x]^0$ does not contain any zero then there is k_0 such that $MK[x]^{k_0} \cap [x]^{k_0} = \varnothing$.
If one chooses $\check{x}^k$ to be the center of $[x]^k$ then the condition $\rho(2 \cdot A) < 1$ can be replaced by $\rho(A) < 1$.

We omit the proof of this theorem since it is similar to the proof of Theorem 10.

Theorem 9. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$.*
Denote $A = |[U]^n| \cdot \ldots \cdot |[U]^1| \cdot |[L]^n| \cdot \ldots \cdot |[L]^1| \cdot d\left(F'\left([x]^0\right)\right)$, where the matrices $[U]^i$ and $[L]^i$ are defined by (3.10) and (3.11) (with $[L] \cdot [U] \supseteq [A] = F'([x]^0)$).
We assume that $\rho(2 \cdot A) < 1$, $IGA\left(F'\left([x]^0\right)\right)$ exists and

$$d\left(F'([x])\right) < \|d([x])\| \cdot C, \quad [x] \subseteq [x]^0, \quad C \geq 0.$$

Then

$$x^2 - k^1 \leq O\left(\|d([x])\|^2\right) \cdot e + Y \cdot f\left(x^2\right), \tag{3.70}$$

$$k^2 - x^1 \leq O\left(\|d([x])\|^2\right) \cdot e - Y \cdot f\left(x^1\right) \tag{3.71}$$

are valid for $Y = (m(F'([x])))^{-1} = U^{-1} \cdot L^{-1}$.
If $[x]^k \neq \varnothing$ for all k then

$$d([x]^{k+1}) = O(\|d([x]^k)\|^2) \cdot e. \tag{3.72}$$

Moreover, if $[x]^0$ does not contain any zero of $f(x)$ then there is $n_0 \geq 0$ and at least one index $i \in \{1, \ldots, n\}$ such that

$$(x^2 - k^1)_i < 0 \qquad or \qquad (k^2 - x^1)_i < 0$$

for $[x]^{n_0} = [x^1, x^2]$.
If one chooses $\check{x}^k$ to be the center of $[x]^k$ then the condition $\rho(2 \cdot A) < 1$ can be replaced by $\rho(A) < 1$.

We will give the proof of this theorem together with the proof of Theorem 11.

3.8 The Simplified Modified Krawczyk Method (with LU Factorization)

For an interval vector $[x] = ([x]_{ij}) \subseteq [x]^0 \in \mathbb{R}^n$ and an inclusion function F' for the Fréchet-derivative f' of a mapping $f : [x]^0 \to \mathbb{R}^n$ over $[x]$, we define the modified Simplified Krawczyk operator as:

$$MSK([x], \check{x}) = \check{x} - U \setminus \{L \setminus \{f(\check{x}) - \left(m(F'([x]^0)) - F'([x])\right) \cdot ([x] - \check{x})\}\},$$

where L, U come from the LU factorization of $m(F'([x]^0))$, i.e. $m(F'([x]^0)) = L \cdot U$, and $\check{x} \in [x]$.
From (3.3), (3.5) and Lemma 1 it follows

$$\begin{aligned} MSK([x], \check{x}) &= \check{x} - U \setminus \{ L \setminus \{f(\check{x})\}\} + U \setminus \{ L \setminus \{(m(F'([x]^0)) - F'([x])) \cdot ([x] - \check{x})\}\} \\ &= \check{x} - Y \cdot f(\check{x}) + U \setminus \{ L \setminus \{(m(F'([x]^0)) - F'([x])) \cdot ([x] - \check{x})\}\} \end{aligned}$$

where $Y = \left(m(F'\left([x]^0\right))\right)^{-1} = U^{-1} \cdot L^{-1}$.
We denote by k^1, k^2 the lower and upper bounds of $SMK([x], \check{x})$.

Theorem 10. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$.*
Denote $A = |U^n| \cdot \ldots \cdot |U^1| \cdot |L^n| \cdot \ldots \cdot |L^1| \cdot d(F'([x]^0))$, where the matrices U^i, L^i, $i = 1, \ldots n$ are defined by (3.1) *and* (3.2) *(with $L \cdot U = A = m(F'([x]^0))$).*
We assume that $\rho(2 \cdot A) < 1$ and $IGA\left(F'\left([x]^0\right)\right)$ exists.
Then
a) if $[x]^0$ contains a zero x^* of f then $x^* \in [x]^k$ and $\lim_{k\to\infty}[x]^k = x^*$,
b) if $[x]^0$ does not contain any zero then there is k_0 such that $MSK\left([x]^{k_0}, \check{x}^{k_0}\right) \cap [x]^{k_0} = \varnothing$.
If one chooses $\check{x}^k$ to be the center of $[x]^k$ then the condition $\rho(2 \cdot A) < 1$ can be replaced by $\rho(A) < 1$.

Proof. First we prove a).
Denote by x^* a zero of f. Suppose $x^* \in [x]^k$. Then

$$\begin{aligned} MSK\left([x]^k, \check{x}^k\right) - x^* = \check{x}^k - x^* - Y \cdot f\left(\check{x}^k\right) \\ + U \setminus \{L \setminus \{\left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right)\}\}. \end{aligned} \tag{3.73}$$

Using (3.7) for $\check{x}^k$ and x^* we have:

$$f\left(\check{x}^k\right) = \delta f\left(\check{x}^k, x^*\right) \cdot \left(\check{x}^k - x^*\right). \tag{3.74}$$

From (3.73) and (3.74) it follows that:

$$\begin{aligned} MSK\left([x]^k, \check{x}^k\right) - x^* &= \check{x}^k - x^* - Y \cdot \delta f\left(\check{x}^k, x^*\right) \cdot \left(\check{x}^k - x^*\right) \\ &\quad + U \setminus \{L \setminus \{\left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right)\}\} \\ &= -Y \cdot \left(m(F'([x]^0)) - \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right) \\ &\quad + U \setminus \{L \setminus \{\left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right)\}\} \\ &= U \setminus \{L \setminus \{\left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right) \\ &\quad - Y \cdot \left(m(F'([x]^0)) - \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right)\}\}. \end{aligned}$$

According to (3.8) we have $\delta f\left(\check{x}^k, x^*\right) \in F'\left([x]\right)$ and therefore

$$m(F'([x]^0)) - \delta f\left(\check{x}^k, x^*\right) \in m(F'([x]^0)) - F'\left([x]^0\right).$$

From this it follows that

$$0 \in \left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right) - \left(m(F'([x]^0)) - \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right)$$

and

$$0 \in U \setminus \{L \setminus \{(m(F'([x]^0)) - F'\left([x]\right)) \cdot ([x]^k - \check{x}^k) - \left(m(F'([x]^0) - \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right)\}\}$$

which implies

$$0 \in MSK\left([x]^k, \check{x}^k\right) - x^*$$

or

$$x^* \in MSK\left([x]^k, \check{x}^k\right).$$

And since $[x]^{k+1} = [x]^k \cap MSK\left([x]^k, \check{x}^k\right)$ also $x^* \in [x]^{k+1}$ is true.
By assumption $x^* \in [x]^0$ and therefore by induction $x^* \in [x]^k$ for each k. For $d([x]^{k+1})$ we can write

$$\begin{aligned} d([x]^{k+1}) &\leq d(MSK([x]^k, \check{x}^k)) \\ &= d(U \backslash \{L \backslash \{\left(m(F'([x]^k)) - \delta f\left(\check{x}^k, x^*\right)\right) \cdot \left(x^* - \check{x}^k\right) \\ &\quad - \left(m(F'([x]^0)) - F'([x])\right) \cdot \left([x]^k - \check{x}^k\right)\}\}) \\ &= d(U \backslash \{L \backslash \{\left(m(F'([x]^0)) - F'([x])\right) \cdot \left([x]^k - \check{x}^k\right)\}\}) \qquad (3.75) \\ &\leq 2 \cdot |U \backslash \{L \backslash \{\left(m(F'([x]^0)) - F'([x])\right) \cdot \left([x]^k - \check{x}^k\right)\}\}| \\ &\leq 2 \cdot |U^n| \cdot \ldots \cdot |U^1| \cdot |L^n| \cdot \ldots |L^1| \cdot \left|m(F'([x]^0)) - F'([x])\right| \cdot \left|[x]^k - \check{x}^k\right| \\ &\leq 2 \cdot A \cdot d([x]^k). \end{aligned}$$

Hence

$$d([x]^{k+1}) \leq (2 \cdot A)^k d([x]^0).$$

Under the assumption $\rho(2 \cdot A) < 1$ from Lemma 14 we get $\lim_{k\to\infty} d([x]^k) = 0$. Now taking into account the fact that $x^* \in [x]^k$ we obtain $\lim_{k\to\infty}[x]^k = x^*$. If one chooses $\check{x}^k$ to be the center of $[x]^k$ in (3.75) we would have $\left|[x]^k - \check{x}^k\right| = \frac{1}{2} d([x]^k)$ and therefore the condition $\rho(A) < 1$ is sufficient.
Now we prove b).
Assume that for all $k \geq 0$ the intersections $MSK\left([x]^k, \check{x}^k\right) \cap [x]^k$ are not empty. Then it holds that

$$[x]^0 \supseteq [x]^1 \supseteq \ldots [x]^k \supseteq [x]^{k+1} \ldots,$$

from which it follows that the sequence is converging to an interval vector $[z]^*$. We now consider the sequence $\{\check{x}^k\}_{k=0}^{\infty}$. There exists a convergent subsequence $\{\check{x}^{k_n}\}_{n=0}^{\infty}$. Suppose that $m^* = \lim_{n\to\infty} \check{x}^{k_n}$. For elements of the sequence $\{\check{x}^{k_n}\}_{n=0}^{\infty}$ it holds that $\check{x}^{k_n} \in [x]^{k_n}$. From this remark it follows that beside of $\lim_{n\to\infty}[x]^{k_n} = [z]^*$ we also have $m^* \in [z]^*$. Using the continuity of operations from the equations

$$\begin{aligned} MSK\left([x]^k, \check{x}^k\right) &= \check{x}^k - Y \cdot f\left(\check{x}^k\right) \\ &\quad + U \backslash \left\{L \backslash \left\{\left(m(F'([x]^0)) - F'\left([x]\right)\right) \cdot \left([x]^k - \check{x}^k\right)\right\}\right\}, \\ [x]^{k+1} &= MSK\left([x]^k, \check{x}^k\right) \cap [x]^k \end{aligned}$$

we get relations

$$\begin{aligned} &[s]^* = m^* - Y \cdot f(m^*) + U \backslash \{L \backslash \{(m(F'([x]^0)) - F'([x])) \cdot ([z]^* - m^*)\}\}, \\ &[z]^* = [s]^* \cap [z]^*, \end{aligned} \qquad (3.76)$$

where $[s]^* = \lim_{n\to\infty} MSK\left([x]^{k_n}, \check{x}^{k_n}\right) = MSK\left([z]^*, m[z]^*\right)$.
From the first equation it follows

$$\begin{aligned} d([s]^*) &= d(U\backslash\{L\backslash\left(m(F'([x]^0)) - F'([x])\right)\cdot([z]^* - m^*)\}\}) \\ &\le 2\cdot|U\backslash\{L\backslash\left(m(F'([x]^0)) - F'([x])\right)\cdot([z]^* - m^*)\}\}| \\ &\le 2\cdot|U^n|\cdot\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1| \\ &\quad\times|m(F'([x]^0)) - F'([x])|\cdot|[z]^* - m^*| \\ &\le 2\cdot|U^n|\cdot\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1| \\ &\quad\times\left|m(F'([x]^0)) - F'([x])\right|\cdot d([z]^* - m^*) \\ &= 2\cdot A\cdot d([z]^*) \end{aligned} \tag{3.77}$$

and using the second equation of (3.76) this implies

$$d([s]^*) \le 2\cdot A\cdot d([s]^*).$$

From which we get $d([s]^*) \le (2\cdot A)^n d([s]^*)$ for all n. Then $d([s]^*) = 0$, since $\rho(2\cdot A) < 1$. And this implies that $[s^*]$ and $[z^*]$ are point vectors and $s^* = z^* = m^*$. From the first equation of (3.76) it follows that

$$\begin{aligned} m^* = m^* - Y\cdot f(m^*) + U\backslash\{L\backslash\{(m(F'([x]^0)) - F'([x]))\cdot(m^* - m^*)\}\}, \\ 0 = Y\cdot f(m^*). \end{aligned}$$

Since Y is nonsingular it follows that $f(m^*) = 0$. But it contradicts the fact that $f(x)$ has no zeros in $[x]^0$.
Hence there is k_0 such that $MSK\left([x]^{k_0}, \check{x}^{k_0}\right) \cap [x]^{k_0} = \varnothing$. If one chooses $\check{x}^k$ to be the center of $[x]^k$ in (3.77) we would have $|[z]^* - m^*| = \frac{1}{2}d([z]^* - m^*)$ and therefore the condition $\rho(A) < 1$ is sufficient.

□

The condition $\rho(2\cdot A_1) < 1$, where $A_1 = |I - (m(F'([x]^0)))^{-1}\cdot F'([x]^0)|$, in Theorems 6 and 7 is not stronger than the condition $\rho(2\cdot A_2) < 1$, where $A_2 = |U^n|\cdot\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1|\cdot d(F'([x]^0))$, in Theorems 10 and 11. Indeed

$$\begin{aligned} |I - (m(F'([x]^0)))^{-1}\cdot F'([x]^0)| &= |((m(F'([x]^0)))^{-1}\cdot(m(F'([x]^0)) - F'([x]^0))| \\ &\le |m(F'([x]^0)))^{-1}|\cdot|m(F'([x]^0)) - F'([x]^0))| \\ &= |m(F'([x]^0)))^{-1}|\cdot d(F'([x]^0))) \\ &\le |U^n|\cdot\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1|\cdot d(F'([x]^0))). \end{aligned}$$

From the theorem of Perron and Frobenius (see [41]) it follows that $\rho(A_1) \le \rho(A_2)$. Thus, the simplified Krawczyk method is applicable everywhere where the simplified modified Krawczyk method is applicable.

Theorem 10 gives sufficient conditions for convergence and divergence of the simplified modified Krawczyk method. The next theorem shows that under the same conditions convergence and divergence are linear.

Theorem 11. *Let there be given an interval vector $[x]^0 \subset \mathbb{R}^n$, a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$ and an inclusion function F' for the Fréchet derivative f' of f over $[x]^0$. Denote $A = |U^n|\cdot$*

$\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1|\cdot d(F'([x]^0))$, *where the matrices* U^i, L^i, $i=1,\ldots n$ *are defined by* (3.1) *and* (3.2). *Suppose that* $\rho(2\cdot A)<1$ *and* $IGA(F'([x]^0))$ *exists. Then*

$$x^2-k^1\le O(\|d([x])\|)\cdot e+Y\cdot f\left(x^2\right), \tag{3.78}$$

$$k^2-x^1\le O(\|d([x])\|)\cdot e-Y\cdot f\left(x^1\right) \tag{3.79}$$

are valid for $Y=\left(m(F'\left([x]^0\right))\right)^{-1}=U^{-1}\cdot L^{-1}$.
If $[x]^k\neq\varnothing$ *for all* k *then*

$$d([x]^{k+1})=O(\|d([x]^k)\|)\cdot e. \tag{3.80}$$

Moreover, if $[x]^0$ *does not contain any zero of* $f(x)$ *then there is* $n_0\ge 0$ *and at least one index* $i\in\{1,\ldots,n\}$ *such that*

$$(x^2-k^1)_i<0\qquad or\qquad (k^2-x^1)_i<0$$

for $[x]^{n_0}=[x^1,x^2]$.
If one chooses $\check{x}^k$ *to be the center of* $[x]^k$ *then the condition* $\rho(2\cdot A)<1$ *can be replaced by* $\rho(A)<1$.

Proof. We again prove two theorems Theorem 9 and Theorem 11 together. For the proof of Theorem 9 we take $[z]=[x]$ and for the proof of Theorem 11 we take $[z]=[x]^0$.
Denote $U\setminus\{L\setminus\{(m(F'([z]))-F'([x]))\cdot([x]-\check{x})\}\}$ by $[r]=[r^1,r^2]$.
Then

$$\begin{aligned}k^1&=\check{x}-Y\cdot f(\check{x})+r^1,\\ k^2&=\check{x}-Y\cdot f(\check{x})+r^2\end{aligned} \tag{3.81}$$

are obvious.
For the right end point x^2 of the interval vector $[x]$ using (3.81) we have

$$x^2-k^1=x^2-\check{x}+Y\cdot f(\check{x})-r^1.$$

Using (3.7) for $\check{x}$ and x^2 we have:

$$f\left(\check{x}\right)=f(x^2)+\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right).$$

Then

$$\begin{aligned}x^2-k^1&=x^2-\check{x}+Y\left(f(x^2)+\delta f\left(\check{x},x^2\right)\cdot\left(\check{x}-x^2\right)\right)-r^1\\&=\left(I-Y\cdot\delta f\left(\check{x},x^2\right)\right)\cdot\left(x^2-\check{x}\right)+Y\cdot f\left(x^2\right)-r^1.\end{aligned} \tag{3.82}$$

Because of (3.8) and Lemma 6 the matrix $\delta f\left(\check{x},x^2\right)$ is nonsingular and $\left(\delta f\left(\check{x},x^2\right)\right)^{-1}\in IGA\left(F'([x])\right)$. Therefore we have

$$\left|\left(I-Y\cdot\delta f\left(\check{x},x^2\right)\right)\cdot\left(x^2-\check{x}\right)\right|=\left|\left(\left(\delta f\left(\check{x},x^2\right)\right)^{-1}-Y\right)\cdot\delta f\left(\check{x},x^2\right)\cdot\left(x^2-\check{x}\right)\right|.$$

Since $\left|\left(\delta f\left(\check{x},x^2\right)\right)^{-1}-Y\right|\le d\left(IGA\left(F'([z])\right)\right)$ and $\left|\delta f\left(\check{x},x^2\right)\right|\le\left|F'\left([x]^0\right)\right|$ we get:

$$\begin{aligned}\left|\left(I-Y\cdot\delta f\left(\check{x},x^2\right)\right)\left(x^2-\check{x}\right)\right|&\le\left(\left(\delta f\left(\check{x},x^2\right)\right)^{-1}-Y\right)\cdot\delta f\left(\check{x},x^2\right)\cdot(x^2-\check{x})\\&\le d\left(IGA\left(F'([z])\right)\right)\cdot\left|F'\left([x]^0\right)\right|\cdot d([x]).\end{aligned}$$

From Lemma 12 it follows

$$\left|F'\left([x]^0\right)\right| \cdot d([x]) = O(\|d([x])\|) \cdot e.$$

For Theorem 9 from Lemma 9 it follows

$$d\left(IGA\left(F'([x])\right)\right) = O(\|d([x])\|) \cdot E,$$

hence

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = O\left(\|d([x])\|^2\right) \cdot e. \tag{3.83}$$

For Theorem 11 we have

$$\left|\left(I - Y \cdot \delta f\left(\check{x}, x^2\right)\right) \cdot \left(x^2 - \check{x}\right)\right| = O\left(\|d([x])\|\right) \cdot e. \tag{3.84}$$

Now we consider $[r]$:

$$\begin{aligned}
|[r]| &= |U \setminus \{L \setminus \{(m(F'([z])) - F'([x])) \cdot ([x] - \check{x})\}\}| \\
&= \left|U^n(\ldots(U^1(L^n(\ldots(L^1\left(m(F'([z])) - F'([x])\right) \cdot ([x] - \check{x}))\ldots)\right| \\
&\leq |U^n| \cdot \ldots \cdot |U^1| \cdot |L^n| \cdot \ldots \cdot |L^1| \cdot |(m(F'([z])) - F'([x])) \cdot ([x] - \check{x})| \\
&\leq |U^n| \cdot \ldots \cdot |U^1| \cdot |L^n| \cdot \ldots \cdot |L^1| \cdot d(F'([z])) \cdot d([x]) \\
&\leq |[U]^n| \cdot \ldots \cdot |[U]^1| \cdot |[L]^n| \cdot \ldots \cdot |[L]^1| \cdot d(F'([z])) \cdot d([x]) \\
&\leq M \cdot d(F'([z])) \cdot d([x]) \\
&\leq \|d([x])\| \cdot M \cdot d(F'([z])) \cdot e,
\end{aligned}$$

where $M = |[U]^n| \cdot \ldots \cdot |[U]^1| \cdot |[L]^n| \cdot \ldots \cdot |[L]^1|$.
For Theorem 9 using the assumption $d\left(F'([x])\right) < C \cdot \|d([x])\|$ we get

$$|[r]| = O\left(\|d([x])\|^2\right) \cdot e. \tag{3.85}$$

For Theorem 11 we have

$$|[r]| = O\left(\|d([x])\|\right) \cdot e. \tag{3.86}$$

The equation (3.85) together with (3.82) and (3.83) gives us

$$x^2 - k^1 \leq O\left(\|d([x])\|^2\right) \cdot e + Y \cdot f\left(x^2\right)$$

and the equation (3.86) together with (3.82) and (3.84) gives us

$$x^2 - k^1 \leq O\left(\|d([x])\|\right) \cdot e + Y \cdot f\left(x^2\right)$$

So we have proved (3.70) and (3.78).
In the same manner we can get (3.71) and (3.79).

From (3.70) and (3.71) we can get (3.72), from (3.78) and (3.79) we can get (3.80). The proof is similar to the proof in Theorem 5.

To prove the last part of Theorem 11 assume that there is no zero of f in $[x]$, i.e. $f(x) \neq 0$, for all $x \in [x]^0$. All matrices $\left(m(F'\left([z]\right))\right)^{-1}$ are nonsingular. The function f is continuous by the assumption (see page 3). From Lemma 13 we have that there exist $c > 0$ and $\varepsilon(c) > 0$ such that:

$$\left(\left(m(F'([z]))\right)^{-1} \cdot f(x)\right)_{i(x)} > c \quad \text{for all } x \in [x]^0 \tag{3.87}$$

and

$$sign\left(\left((m(F'([z])))^{-1}\cdot f(x^1)\right)_{i(x^1)}\right)=sign\left(\left((m(F'([z])))^{-1}\cdot f(x^2)\right)_{i(x^1)}\right),\quad \text{for all } x^1,x^2\in[x], \tag{3.88}$$

once $|x_1-x_2|<\varepsilon(c)\cdot e$.
We now estimate $d\left([x]^{k+1}\right)$. For Theorem 9 we have:

$$\begin{aligned} d\left([x]^{k+1}\right) &\le d(MK([x]^k,\check{x}^k)\\ &= d\left(\check{x}^k-Y\cdot f(\check{x}^k)+U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}\right)\\ &= d\left(U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}\right). \end{aligned}$$

The fact that $0\in[x]^k-\check{x}^k$ implies $0\in U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}$ and therefore

$$\begin{aligned} &d(U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\})\\ &\qquad\le 2\cdot|U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}|. \end{aligned}$$

So we have

$$\begin{aligned} d\left([x]^{k+1}\right) &\le 2\cdot\left|U\setminus\{L\setminus\{\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}\right|\\ &= 2\cdot\left|U^n\cdot\ldots\cdot(U^1\cdot(L^n\cdot\ldots\cdot(L^1\cdot\left(m(F'([x]^k))-F'([x]^k)\right)\cdot\left([x]^k-\check{x}^k\right)\right|\\ &\le 2\cdot|U^n|\cdot\ldots\cdot|U^1|\cdot|L^n|\cdot\ldots\cdot|L^1|\cdot\left|m(F'([x]^k))-F'([x]^k)\right|\cdot\left|[x]^k-\check{x}^k\right|\\ &\le 2\cdot M\cdot\left|m(F'([x]^k))-F'([x]^k)\right|\cdot\left|[x]^k-\check{x}^k\right| \end{aligned}$$

Since $0\in[x]^k-\check{x}^k$ and $0\in m(F'([x]^k))-F'([x]^k$ we have

$$\begin{aligned} \left|[x]^k-\check{x}^k\right| &\le d\left([x]^k-\check{x}^k\right)=d\left([x]^k\right),\\ \left|m(F'([x]^k))-F'([x]^k)\right| &\le d\left(m(F'([x]^k))-F'([x]^k)\right)=d\left(F'([x]^k)\right). \end{aligned} \tag{3.89}$$

Therefore $d\left([x]^{k+1}\right)\le 2M\cdot d(F'([x]^k))\cdot d([x]^k)\le 2\cdot A\cdot d\left([x]^k\right)$ and $d\left([x]^{k+1}\right)\le(2\cdot A)^{k+1}\cdot d\left([x]^0\right)$.
For Theorem 11 we have:

$$\begin{aligned} d\left([x]^{k+1}\right) &\le d(MK([x]^k,\check{x}^k)\\ &= d\left(\check{x}^k-Y\cdot f(\check{x}^k)+U\setminus\{L\setminus\{\left(m(F'([x]^0))-F'([x]^0)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}\right)\\ &= d\left(U\setminus\{L\setminus\{\left(m(F'([x]^0))-F'([x]^0)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}\right). \end{aligned}$$

The fact that $0\in[x]^k-\check{x}^k$ implies $0\in U\setminus\{L\setminus\{\left(m(F'([x]^0))-F'([x]^0)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}$ and therefore

$$\begin{aligned} &d(U\setminus\{L\setminus\{\left(m(F'([x]^0))-F'([x]^0)\right)\cdot\left([x]^k-\check{x}^k\right)\}\})\\ &\qquad\le 2\cdot|U\setminus\{L\setminus\{\left(m(F'([x]^0))-F'([x]^0)\right)\cdot\left([x]^k-\check{x}^k\right)\}\}|. \end{aligned}$$

So we have

$$\begin{aligned} d\left([x]^{k+1}\right) &\leq 2 \cdot \left|U \setminus\{L \setminus\{\left(m(F'([x]^0)) - F'([x]^0)\right) \cdot \left([x]^k - \check{x}^k\right)\}\}\right| \\ &= 2 \cdot \left|U^n \cdot \ldots \cdot (U^1 \cdot (L^n \cdot \ldots \cdot (L^1 \cdot \left(m(F'([x]^0)) - F'([x]^0)\right) \cdot \left([x]^k - \check{x}^k\right) \ldots\right| \\ &\leq 2 \cdot |U^n] \cdot \ldots \cdot |U^1| \cdot |L^n| \cdot \ldots \cdot |L^1| \cdot \left|m(F'([x]^0)) - F'([x]^0)\right| \cdot \left|[x]^k - \check{x}^k \ldots\right| \\ &= 2 \cdot A \cdot \left|[x]^k - \check{x}^k\right| \\ &\leq 2 \cdot A \cdot d([x]^k). \end{aligned} \tag{3.90}$$

consequently

$$d\left([x]^{k+1}\right) \leq (2 \cdot A)^{k+1} \cdot d\left([x]^0\right).$$

For both Theorem 9 and Theorem 11 we have

$$d\left([x]^{k+1}\right) \leq (2 \cdot A)^{k+1} \cdot d\left([x]^0\right).$$

Suppose $[x]^k \neq \varnothing$ for all k. Then we have $\rho(2 \cdot A) < 1$ and therefore from Lemma 14 it follows $\lim_{k\to\infty} d([x]^k) = 0$. Let the term $O(\|d([x])\|^2)$ in (3.70) and (3.71) be bounded by $\alpha \cdot \|d([x])\|^2$, $\alpha \geq 0$. From $\lim_{k\to\infty} d([x]^k) = 0$ it follows that there exists $n_0 \geq 0$ such that

$$\begin{aligned} \alpha \cdot \|d([x]^{n_0})\|^2 < c, \\ d([x]^{n_0}) < \varepsilon(c) \cdot e. \end{aligned}$$

Then for Theorem 9 from (3.70), (3.71), (3.66) and (3.67) for at least one index $i_0 \in \{1, \ldots, n\}$ either $(x^2 - k^1)_{i_0} < 0$ or $(k^2 - x^1)_{i_0} < 0$ takes place for $[x]^{n_0} = [x^1, x^2]$.

The same can be obtained for Theorem 11.

From Lemma 11 it follows that $[x]^{n_0+1} = [x]^{n_0} \cap [K([x]^{n_0}, m([x]^{n_0}) = \varnothing$. This contradicts $[x]^k \neq \varnothing$ for all k. Hence there is $[x]^{k_0+1} = \varnothing$ or from Lemma 11 either $(x^2 - k^1)_{i_0} < 0$ or $(k^2 - x^1)_{i_0} < 0$ takes place for $[x]^{n_0} = [x^1, x^2]$ and for at least one index $i_0 \in \{1, \ldots, n\}$. If one chooses $\check{x}^k$ to be the center of $[x]^k$ instead of (3.89) we would have $|[x]^k - \check{x}^k| = \frac{1}{2} d([x]^k)$ and therefore the condition $\rho(A) < 1$ is sufficient. □

3.9 Numerical examples

All calculations had been performed on Ultra 10 Workstation (Sun Microsystems).
The time is given in STU (see Appendix A).
As was mentioned in Section 1.2.4 on a computer not all numbers have representations. And it can happen that the calculated value of $L \cdot U$ in the modified Krawczyk and in the simplified modified Krawczyk methods does not equal to $m(F'([x]))$. In the theory we use the fact that $U\setminus\{L\setminus\{m(F'([x]))\}\} = I$. We should preserve this fact. In order to enclose the identity matrix in the resulting interval we replace a point matrix $m(F'([x]))$ with the interval matrix $[Z] \supseteq [L] \cdot [U]$, where $[L]$ and $[U]$ are point matrices L and U. It is allowable since the matrix $m(F'([x]))$ may be any nonsingular matrix. We also consider $\check{x}$ as an interval vector in the computation of $f(\check{x})$, since the exact value should be enclosed, too. In the following examples the stopping criterion is when the diameter of the interval vector is less than $\varepsilon = 10^{-6}$, convergence is when the method converges to the solution without empty inersections and divergence is when at least one component of the intersection $[x]^k \cap Operator([x]^k)$ is empty. We perform the LU factorization using the Gaussian elimination algorithm.

Example 1.

We consider the two-point boundary value problem

$$u'' = \phi(u), \quad u(0) = u(1) = 0, \quad x \in [0,1],$$
$$\phi(u) = u^4 + u^3 + 1.$$

Discretizing with finite differences at n equidistant points $t_i = ih$, $h = \dfrac{1}{n-1}$, $i = 0, \dots, n-1$, results in the nonlinear system:

$$u_{i-1} - 2u_i + u_{i+1} - h^2\phi(u_i) = 0, \quad i = 1, \dots, n-1,$$

where $u_0 = u_{n-1} = 0$, $u_i \approx u(t_i)$, $\quad i = 1, \dots, n-2$.
We consider the left-hand side of this system as a mapping $f : [u]^n \subset \mathbb{R}^n \to \mathbb{R}^n$ and use all considered modifications of the Krawczyk method and the Krawczyk method itself to solve

$$f(x) = 0.$$

In the numerical example we take $n = 150$ and $[x]^0 = [u]^n$ as a starting vector.
For $[u] = [-1, 0]$ we get convergence to the solution plotted in Figure 3.1(a).
For $[u] = [-0.12, 0]$ we get divergence. In Figure 3.1(b) we plotted the midpoints of the nonempty components of the intersection $[x]^k \cap Operator([x]^k)$.

$[u]$	Method	Steps	Time	Status
$[-1, 0]$	Krawczyk	4	158.95	convergence
	simplified Krawczyk	7	216.46	convergence
	modified Krawczyk	4	7.03	convergence
	simplified modified Krawczyk	7	13.30	convergence
$[-0.12, 0]$	Krawczyk	1	30.92	divergence
	simplified Krawczyk	1	30.92	divergence
	modified Krawczyk	1	1.45	divergence
	simplified modified Krawczyk	1	1.45	divergence

Table 3.1: Time measurements for Example 1

Example 2.

We now consider the boundary value problem

$$\Delta u = \alpha \cdot \phi(u), \ u(\partial\Omega) = 0, \ \Omega = [0,1] \times [0,1], \quad x \in \Omega$$

We will specify the function $\phi(u)$ later.

Discretizing with finite differences at $n = N^2$ equidistant points $t_{i,j} = (ih, jh)$, $h = \dfrac{1}{N-1}$, $i, j = 0, \dots, N-1$ results in the nonlinear system:

$$u_{i-N} + u_{i-1} - 4u_i + u_{i+1} + u_{i+N} - \alpha h^2 \phi(u_i) = 0, \quad i = N+1, \dots, N^2 - N - 2,$$

where $u_{i+N \cdot j} \approx u(t_{i,j})$.
We consider the right-hand side of this system as a mapping $f : [x]^0 \subset \mathbb{R}^n \to \mathbb{R}^n$, where

$[x]^0 = [u]^n$, and use the modified Krawczyk and modified simplified Krawczyk methods to solve

$$f(x) = 0.$$

In the numerical examples we take $N = 20$
In order to reduce the bandwidth and concentrate all the nonzero elements near the diagonal we use the *symrcm* function (MatLab), the symmetric reverse Cuthill-McKee ordering. See Figure 3.2. Before the reordering matrices L and U from the LU factorization of the matrix of the nonlinear system have 8019 nonzero elements each. After the reordering they have 5910 nonzero elements each.
First we consider $\phi(u) = e^u$, $[u] = [-1, 0]$, hence $[x]^0 = [-1, 0]^n$.

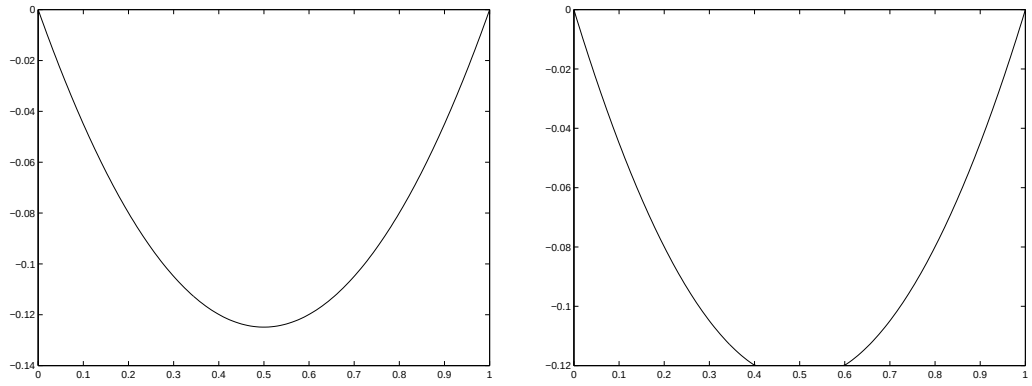
For $\alpha = 1$ we get convergence to the solution plotted in Figure 3.3(a). For $\alpha = 45$ we get divergence. In Figure 3.3(b) we plotted the midpoints of the nonempty components of the intersection $[x]^k \cap Operator([x]^k)$.

α	Method	Steps	Time	Status
1	Krawczyk	3	1728.75	convergence
	modified Krawczyk	3	52.42	convergence
	simplified modified Krawczyk	4	26.89	convergence
45	Krawczyk	2	1138.76	divergence
	modified Krawczyk	2	40.50	divergence
	simplified modified Krawczyk	2	20.49	divergence

Table 3.2: Time measurements for Example 2 with $\phi(u) = \alpha \cdot e^u$, $[u] = [-1, 0]$

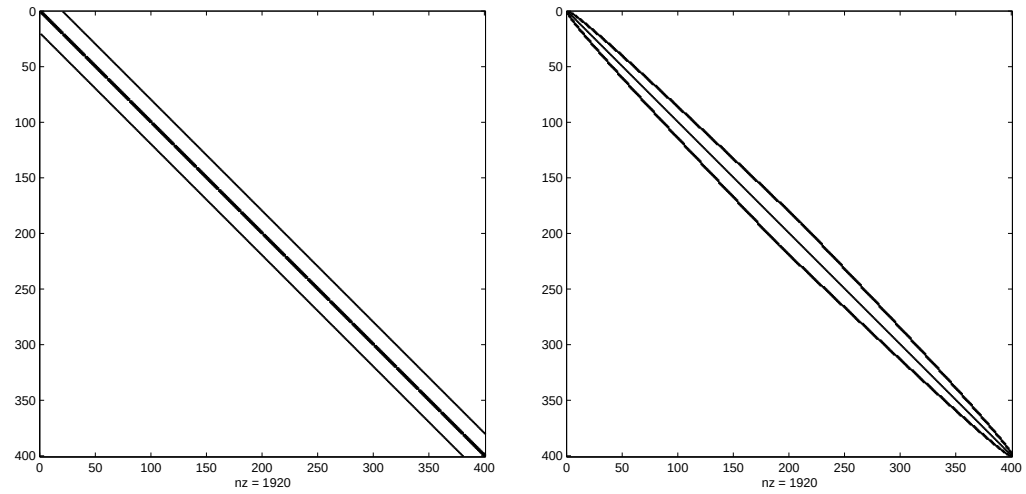
Now we consider $\phi(u) = u^4 + u^3 + 1$, $\alpha = 1$. For $[x]^0 = [-1, 0]^n$ we get convergence to the solution plotted in Figure 3.4(a). For $[x]^0 = [-0.06, 0]^n$ we get divergence. In Figure 3.4(b) we plotted the midpoints of the nonempty components of the intersection $[x]^k \cap Operator([x]^k)$.

We do not give time measurements for the simplified Krawczyk method in Example 2. Because of the overestimation in the computation of $F'([x]^0)$ after the first iteration the simplified



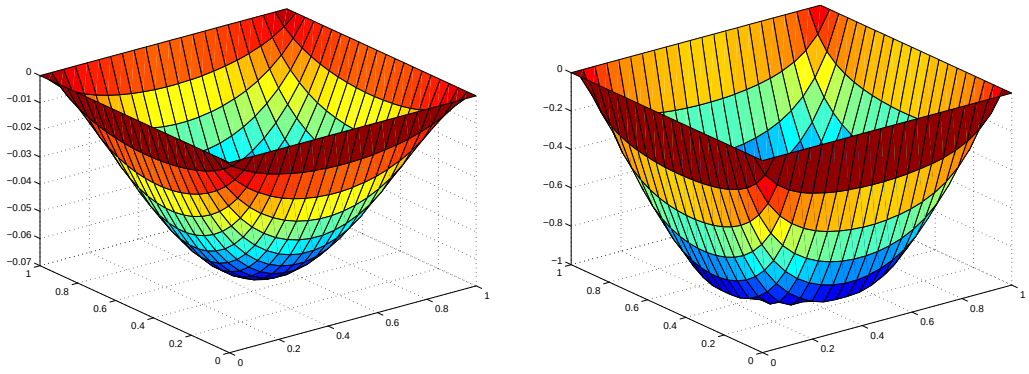
(a) Convergence (b) Divergence

Figure 3.1: The two-point boundary value problem



(a) Original (b) After permutation

Figure 3.2: The sparsity pattern of the matrix of the nonlinear system



(a) Convergence (b) Divergence

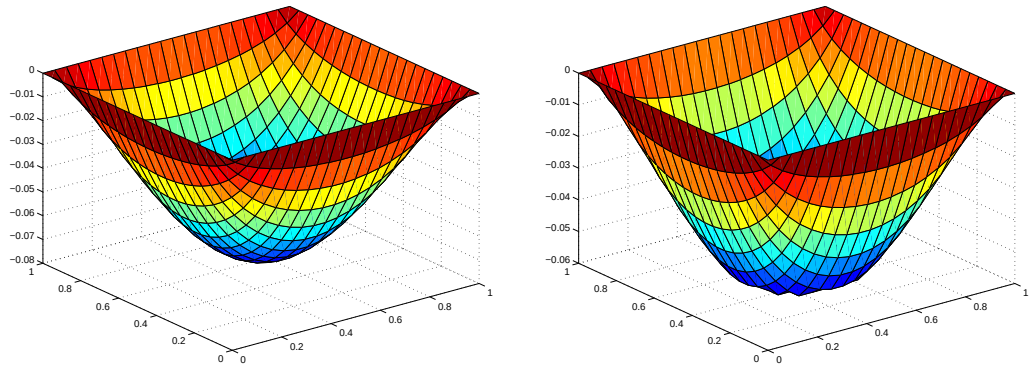
Figure 3.3: The boundary value problem with $\phi(u) = e^u$

Krawczyk operator does not shrink the interval vector any more and the method runs onto an infinite loop. Both the modified Krawczyk and the simplified modified Krawczyk methods significantly reduce the number of computations in one iteration. Which results in significantly less time compares to the normal Krawczyk method (see Table 3.1, Table 3.3 and Table 3.3). On the other hande both methods need stronger assumptions than by the normal Krawczyk method. In Example 1 we see that the modified Krawczyk method is preferrable over the simplified modified Krawczyk method in case of convergence. It is because each iteration of the modified Krawczyk method for this function is not much expensive compared to the simplified modified Krawczyk method. There we have a system of 150 equations. In Example 2 the

simplified modified Krawczyk method is preferrable. There we have a system of 400 equations. We see that even two iterations of the simplified modified Krawczyk method are cheaper than one iteration of the modified Krawczyk method. So the choice of the method depends on the complexity of the system.

$[u]$	Method	Steps	Time	Status
$[-1, 0]$	Krawczyk	3	1984.99	convergence
	modified Krawczyk	3	59.92	convergence
	simplified modified Krawczyk	6	52.47	convergence
$[-0.06, 0]$	Krawczyk	1	593.23	divergence
	modified Krawczyk	1	18.47	divergence
	simplified modified Krawczyk	1	18.47	divergence

Table 3.3: Time measurements for Example 2 with $\phi(u) = u^4 + u^3 + 1$, $\alpha = 1$



(a) Convergence

(b) Divergence

Figure 3.4: The boundary value problem with $\phi(u) = u^4 + u^3 + 1$

Chapter 4

The Parallel Method for Verified Global Optimization

4.1 Parallel Processing

The main purpose of parallel processing is to perform computations faster than they can be done with a single processor by using a number of processors concurrently. The pursuit of this goal has had a tremendous influence on almost all the activities related to computing. The need for faster solutions and for solving larger-size problems arises in a wide variety of applications. These include fluid dynamics, weather prediction, modeling and simulation of large systems, information processing and extraction, image processing, artificial intelligence, automated manufacturing and global optimization.

Three main factors have contributed to the current strong trend in favor of parallel processing. First, the hardware cost has been falling steadily; hence it is now possible to build systems with many processors at a reasonable cost. Second, the very large scale integration circuit technology has advanced to the point where it is possible to design complex systems requiring millions of transistors on a single chip. Third, the fastest cycle time of a von Neumann-type processor seems to be approaching fundamental physical limitations beyond which no improvement is possible; in addition, as higher performance is squeezed out of a sequential processor, the associated cost increases dramatically. All these factors have pushed researchers into exploring parallelism and its potential use in important applications.

Definition 6 (see [35]). *A **parallel computer** is simply a collection of processors, typically of the same type, interconnected in a certain fashion to allow the coordination of their activities and the exchange of data.*

The processors are assumed to be located within a small distance of each other, and are primarily used to solve a given problem jointly. In contrast to such systems there are distributed systems, where a set of possible many different types of processors are distributed over a large

geographic area, and where the primary goals are to use the available distributed resources, and to collect information and transmit it over a network connecting the various processors.

4.1.1 Architecture Classifications

Flynn's Hardware Taxonomy

Several alternative parallel architectures were developed throughout the 1980s and 1990s. A classification of computer architectures was proposed by Flynn [16] in 1966 and is used to distinguish between alternative parallel architectures. Flynn proposed the following four classes, based on the interaction between instruction and data streams of the processor(s):

SISD - Single Instruction stream Single Data stream. Computers in this class execute a single instruction on a single piece of data before moving on to the execution of the next instruction on a different piece of data. The traditional von Neumann computer is a scalar uniprocessor that falls in this category.

SIMD - Single Instruction stream Multiple Data stream. Computers with this architecture can execute a single instruction simultaneously on multiple data. This type of computation is possible only if the operations of an algorithm are identical over a set of data and if the data can be mapped on multiple processors for concurrent execution. Examples of SIMD systems are Connection Machine CM-2 (up to 65,536 Processing Elements) from Thinking Machines Corporation, GF11 (up to 566 Processing Elements) from IBM Watson Research Center, MasPar MP-1 and MP-2 (up to 16,384 Processing Elements) from MasPar Computer Corporation and all vector computers.

MISD - Multiple Instruction stream Single Data stream. Computers with this architecture execute multiple instructions concurrently on a single piece of data. This form of parallelism has not received much attention in practice. It appears in the taxonomy for completeness.

MIMD - Multiple Instruction stream Multiple Data stream. Computers in this class can execute multiple instructions concurrently on multiple piece of data. Multiple instructions are generated by code modules that may execute independently from each other. Each module may operate either on a subset of the data of the problem or on a copy of all the problem data, or it may access all the data of the problem, together with the other modules, in a way that avoids read or write conflicts. Examples of MIMD systems are Connection Machine CM-5 from Thinking Machines Corporation, NCUBE (up to 8,192 Processing Elements), iWarp, iPSC (up to 1,024 Processing Elements), Paragon from Intel, SP-2 from IBM, ASCI Red TOPS from Intel (9,536 Processing Elements) and Cray T3E.

Multiprocessor systems are also characterized by the number of available processors. Small-scale parallel systems have up to 16 processors, medium-scale systems up to 128, and large-scale systems up to 1024. Systems with more that 1024 processors are considered to be massively parallel. Multiprocessor systems are also characterized as **coarse-grain** or **fine-grain**. In coarse-grain systems each processor is very powerful, typically of the kind found in contemporary workstations. Fine-grain systems typically use simple processing elements. For example, an Intel iPSC/860 system with 1024 processors is a coarse-grain parallel machine. The Connection Machine CM-2 with up to 65,536 simple processing elements is a fine-grain, massively parallel machine.

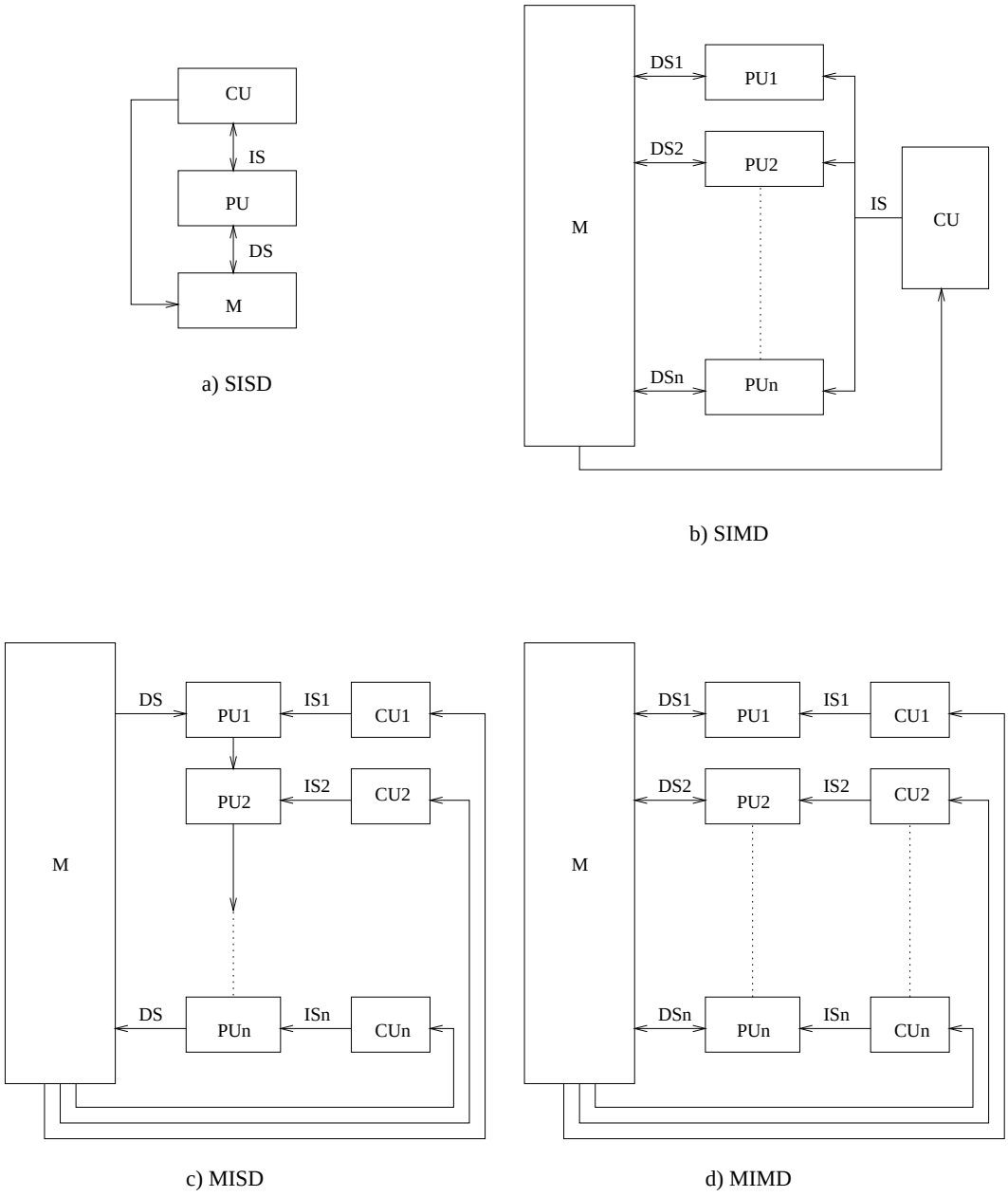


Figure 4.1: Flynn's taxonomy

Organization of Memory

Beside Flynn's taxonomy one distinguishes computers by the organization of memory. The organization of memory is defined by the so-called **interconnection network topology**.

Definition 7. *An **interconnection network topology** is a mapping function from the set of processors and memories onto the set of all subsets of processors and memories.*

In other words, the topology describes how processors and memories are connected to other processors and memories.

A fully connected topology, for example, is a mapping in which each processor is connected to all other processors. Nearly all parallel processors can be distinguished by their interconnection network topology. While the speed and capacity may vary, the most significant difference between parallel processors is their interconnection network topology.

Interconnection network topologies belong to one of two important groups:

Distributed memory. It typically combines local memory and processor at each node of the interconnection network. Message-passing is used to communicate between any two processors, and there is no global, shared memory. Commercial examples of distributed memory architectures are all cluster parallel computers: RS/6000 "ASCI White" powered by 8,192 copper microprocessor, ASCI Red TOPS from Intel powered by 9,536 Pentium II Xeon Core Processors, ALiCE from Compaq with 128 Alpha processors.

Shared memory. It typically accomplishes interprocessor coordination through a global memory shared by all processors. Commercial examples of shared memory multiprocessors are all SMP parallel computers: IBM RS/6000 SP with up to 16 Power II processors, Compaq AlphaServer ES40 with up to 4 Alpha 21264A Processors.

In recent years two architectures became most popular: SMP (symmetric multiprocessing) and cluster computing. They do not belong to a certain class of Flynn's taxonomy. The SMP belongs to shared memory topology. With SMP, multiple identical processors share a single global memory space allowing the processors to cooperatively execute a single copy of an operating system. SMP systems allow multiple concurrent processes to be executed in parallel within different processors. Through the system's shared memory, these processes can communicate rapidly with each other. SMP with cache processors does not scale well because it becomes increasingly difficult to keep track of data as the number of processors grows. Basically, every processor needs to tell every other processor what it is doing. This is relatively easy when the number of processors is small. However, the interprocessor communications problem grows quickly as the number of processors increases. Typically the relationship is not linear.

By Joseph A. Kaplan and Michael L. Nelson at Queen's University of Belfast a cluster has been defined to be (see URL: http://www.pcc.qub.ac.uk/tec/courses/intro/ohp/intro-ohp.html)

Definition 8. *A cluster is a collection of computers on a network that can function as a single computing resource through the use of additional system management software.*

A cluster computing belongs to distributed memory topology. A unified cluster providing distributed computing services is achieved through use of specialized system management software:

DQS - Distributed Queuing System - batch environment containing different queues based on architecture and group;

CODINE - COmputing in DIstributed Networked Environments - optimal utilization of the compute resources in heterogeneous networked environments;

NQE - Network Queuing Environment - UNIX batch environment consisting of NQS and NLB;

LSF - Load Sharing Facility - distributes the workload around one or more large heterogeneous clusters of workstations.

4.1.2 Measures of Performance

Definition 9 (Speedup). *Speedup $S(p)$ is the ratio of the solution time T_{ser} for a problem with the fastest known serial code executed on a single processor, to the solution time $T_{par}(p)$ with the parallel algorithm executed on multiple processors p.*
The sequential algorithm is executed on one of the processors of the parallel system.

$$S(p) = \frac{T_{ser}}{T_{par}(p)}.$$

Definition 10 (Efficiency). *Efficiency $E(p)$ is the ratio of speedup $S(p)$ to the number of processors p.*

$$E(p) = \frac{S(p)}{p}.$$

Efficiency provides a way to measure the performance of an algorithm independently from the power of processors on the particular computer. **Linear speedup** is observed when a parallel algorithm on p processors runs p times faster than on a single processor. Linear speedup corresponds to efficiency of 1. **Sublinear speedup**, or efficiency less than 1, is achieved when the improvement in performance is less than p. Sublinear speedup is common due to the presence of sequential segments of code, delays for processor synchronization, overhead in spawning independent tasks. **Superlinear speedup**, with efficiency exceeding 1, is unusual. It indicates that the parallel algorithm follows a different, more efficient solution path than the sequential algorithm. It is often possible in such situations to improve the performance of the sequential algorithm based on insights gained from the parallel algorithm.

4.2 Existing Parallel Approaches

The considered methods for verified global optimization belong to the branch and bound class. By the branch and bound method (see Chapter 2), the given initial problem will be successively subdivided into subproblems. Some of these subproblems need not be considered further, depending on the criterion of the bounding strategy. By the parallelization of the branch and bound method the independent subproblems will be distributed between the available processors and will be handled there. In order to achieve an efficient parallel method one should be concerned with the following issues:

1. All processors should always be busy handling the subproblems;

2. The total cost of handling all the subproblems should not be greater than the cost in the serial method;

3. The additional cost (for instance for the communication and synchronization) should be relatively small.

Issue 2 means more precisely that all processors should not be just busy, but be doing useful work. If subproblems, which have been discarded in the serial method, are considered, then the load of processors become high and the efficiency of the parallel method is not optimal. Note that such situations can really arise, because a parallel method will process the subproblems in a different order than the serial method. So in a parallel method for verified global optimization boxes $[x]$ with $\inf(F([x])) > f^*$ may be handled, although they would not be handled in the serial method, since they are stored behind the boxes with better upper bounds and for them $\inf(F([x])) > \tilde{f}$ is valid.

The parallelization of the branch and bound method on a MIMD computer with shared memory is relatively easy. All the subproblems and the criterion for the bounding-strategy are stored in the shared memory. Every processor takes a subproblem and the criterion in each step, handles the subproblem, and puts a new subproblem and a probably updated criterion to the shared memory and then starts over again. The method with shared memory runs almost in the same way as the serial method. But the communication between processors and the shared memory could be expensive. And also memory access conflicts could arise, which is a general problem for shared memory architecture.

Here we shall investigate the parallelization on a MIMD computer with distributed memory. In this case it is difficult to fulfill all the issues 1.-3. above simultaneously. For issue 2 it is important that the current best bounding criterion (in case of the global interval optimization this is the smallest value of $\tilde{f}$ on all processors) is sent to every processor as fast as possible. For this we need communications. Whereas in many problems from linear algebra we can estimate the communication cost of the method a priori, this is much more difficult in the branch and bound methods. Therefore, a static load balancing is not practicable. Instead, a dynamic load balancing is required. One must try to distribute the remaining subproblems between processors, in order to keep all processors equally loaded or at least keep them all busy. In addition one should try to fulfill all three issues to get an efficient parallel method.

The dynamical load balancing between the processors can be done with respect to several different criteria (see [15]). One is with respect to the quantity, i.e. to the number of the subproblems, another is with respect to the quality. The quality of a subproblem determines how soon the subproblem should be handled. "Better" subproblems are those for which we expect that handling those will allow us to discard many of the remaining subproblems. In global optimization, "better" subproblems are those which give a smaller upper bound for f^*.

When parallelizing the branch and bound method there are two possibilities for managing subproblems. The first is to store subproblems on one central processor. The other is to distribute them between processors. In our context of verified global optimization this means either to store boxes in a list on one processor or to store them in lists on every processor. The first case has a disadvantage: the maximal length of a list is limited by the amount of memory of one processor, whereas in the second case the memories of all processors can be used.

In this section we present some parallel approaches for methods for verified global optimization considered in the literature. In [14], where the parallelization of different methods for verified global optimization is investigated, one can find a very simple parallelization where boxes are stored in a central list and are handled by all processors. Similar parallelizations of this master-slave principle are in [21] and [8]. In [15] Eriksson manages processors in a ring where each processor has its own list. All these parallelizations, with exception of the approach of Moore, Hansen and Leclerc (see [40]), use the same serial method, where the box is chosen in accordance to the *best-first*-strategy and the bisection strategy A for the box multisection (see

pages 17 and 19). As an accelerating device along with the midpoint test, the monotonicity test is used. In [40] the parallelization is derived from the method that uses the *oldest-first*-strategy. In the following, p will always denote the number of processors.

4.2.1 The Approach of Dixon and Jha (1993)

Characterization

list management	best-first
multisection parameter l	1
subdivision strategy	A
accelerating devices	monotonicity test interval Newton method

The parallelization in [14] takes place on a Transputer Net with 13 T800 Transputers. They are arranged in a tree where each node has three children. The transputer in the root of the tree manages the list of boxes. If there are more than p boxes in the list then each processor handles the box. Otherwise the next box of the list is subdivided orthogonally to one direction into p parts which are distributed between processors. This method was tested on five test functions. The speedup was disappointing. Using 13 processors the speedup was between 2.83 and 8.75. In the majority of test functions it was less than 4.

4.2.2 The Approach of Henriksen and Madsen (1992)

Characterization

list management	depth-first
multisection parameter l	1
subdivision strategy	A
accelerating devices	monotonicity test interval Newton method

In the parallel methods of Henriksen and Madsen in [21] the interval Newton method, as in Section 4.2.1, is used as an accelerating device. The parallel methods were implemented in a net of T800 Transputers. For parallelization, a master-slave principle was used.

The master manages a central list L of boxes and the upper bound $\tilde{f}$. It sends boxes from the list to slaves, who in turn send back the result to the master. A result is a pair consisting of a box and a lower bound, and the actual value for $\tilde{f}$. The master sends the best (smaller) $\tilde{f}$ to the rest of the processors (see Figure 4.2). The starting box is subdivided into $p-1$ parts at the beginning.

The program was tested on 1, 4, 8, 16 and 32 processors. When passing from 16 to 32 processors in most cases the speedup increases only a little or even decreases much. The decrease was observed even earlier for the majority of test functions. One of the main reasons of the decrease of speedup is the overhead for communication.

To reduce communication, in [21] the *depth-first*-strategy was used instead of the *best-first*-strategy. This has the advantage that the slaves need not to get a box for handling in each iteration. Instead, after bisection, they keep one box for further handling and send only the second box back to the master (if not discarded). The slave must request a box from the master only if both boxes are discarded of fulfill the stopping criterion. In [21] the upper bound $\tilde{f}$ was initialized to the global minimum f^*. So depth-first becomes similar to best-first (no box is handled unnecessarily; even the set of boxes handled is minimal). Therefore it was assumed that f^* is known in advance which is not the case in general. Initializing $\tilde{f}$ to f^* one achieves

that for all selection strategies the same boxes are handled, although in different order.

The speedup for this depth-first-method was almost always better than with the best-first-method (also initialized with $\tilde{f} = f^*$), sometime even significantly. But even using the *depth-first*-strategy, the speedup does not increase any more or increases only little bit for larger p. For some test problems the speedup for 32 processors was below 16. For the others the speedup on 32 processors was between 19 and 28.

To summarize, [21] showed that using the *depth-first*-strategy one can achieve a better speedup. But one has to know a good upper bound $\tilde{f}$ in advance. The serial method with the *depth-first*-strategy otherwise is significantly less efficient than with the *best-first*-strategy. If one chooses the *best-first*-strategy to avoid these problems then one should choose also another communication model.

The approach from [21] that was described here thus has the following disadvantages:

1. the efficiency decreases with the number of processors, the master can become a bottleneck;

2. the maximal length of the list of boxes is limited by the amount of memory on the master;

3. the depth-first-method, under the realistic assumption that one cannot initialize $\tilde{f}$ by f^*, is usually less efficient than the best-first-method. In order to get an initial value $\tilde{f}$ close to f^*, [8] proposed to perform some local nonverified minimization before executing the method.

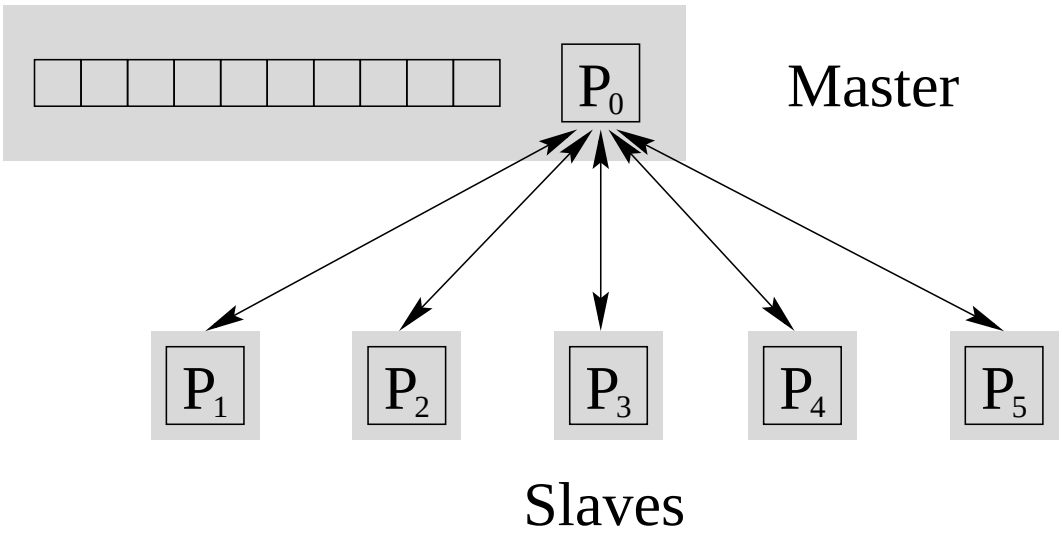


Figure 4.2: The master-slave model: The master manages the central sorted list and the upper bound $\tilde{f}$. The boxes are sent to the slaves for handling, and the results are received back.

4.2.3 The Approach of Eriksson (1991)

Characterization

list management	best-first
multisection parameter l	1
subdivision strategy	A
accelerating devices	monotonocity test nonconvexity text interval Krawczyk method

The parallelization of Eriksson in [15] is based on a serial method described also in [15] which uses the Krawczyk method as an accelerating device. In the method there is no list for potential solution boxes. The iteration is terminated as soon as all the boxes of the sorted list have diameter less than ε ($\varepsilon = 0.1$).

The method, which was implemented on a iPSC/2 Hypercube, arranges all processors in a pool. The processors are logically located in a ring. There is an orientation in the ring, so that there is a next one for every processor. On every processor there are two processes running: a

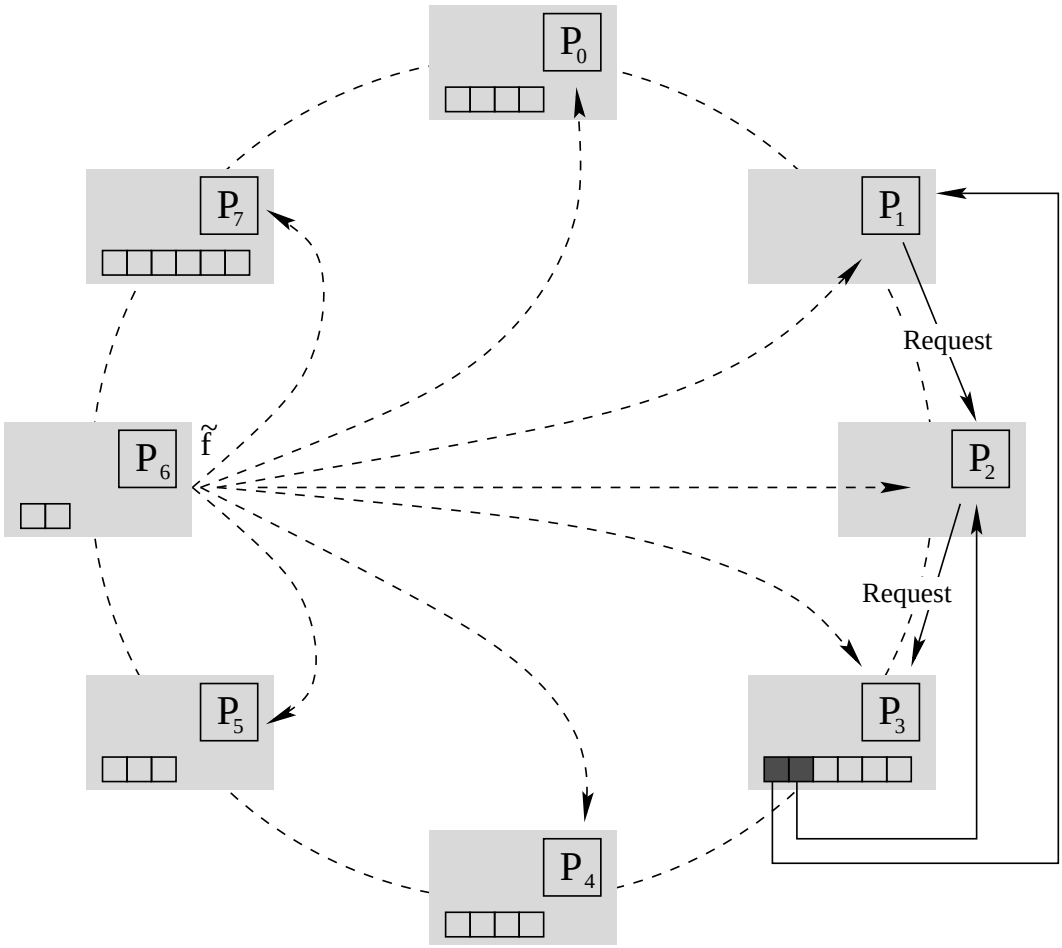


Figure 4.3: Communication structure by Eriksson

worker and a scheduler. The worker is responsible for handling boxes and for the distribution of the upper bound $\tilde{f}$. The scheduler is responsible for load balancing of boxes left to be handled. On each processor the scheduler manages its own list of boxes. When the worker has found a better upper bound $\tilde{f}$, it sends this value to all processors immediately. This becomes possible by using an asynchronous receive-command available on the iPSC/2 that indicates readiness of another worker. A signal sent to a worker is immediately received and its readiness is reset.

A scheduler is responsible for the load balancing of the boxes. Several approaches were tested. Balancing with respect to the quantity of boxes was implemented using the receive-initiated-scheduling. When the worker on a processor has handled a box and the list of the scheduler is empty, the scheduler sends a request-message to the next processor in the ring. If this processor has no boxes to give, it retransmits the request further. The first scheduler, whose list is not empty, sends the box to the scheduler of the process which initiated the request. The box will not travel in the ring, but is sent to the corresponding processor directly.

Balancing with respect to the quality of boxes was implemented as sender-initiated-scheduling. In this way one tries to achieve that boxes with small lower bounds are handled as soon as possible. More precisely, one tries to handle the p bestmost boxes analogously to the serial method. Since we have local lists of boxes, it is difficult to guarantee that really the p bestmost boxes are handled. The following scheme was developed in [15]: If the number of boxes on a certain processor is greater than the given limit (it was set to 5), then the processor sends its first box to a randomly selected processor. One modification of this approach is to use a dynamic limit instead of a static one. If after the insertion the new box is in the head of the list (lists are sorted), then the limit is decremented, otherwise it is incremented. In this manner good boxes are sent early. On the other hand the processors with least promising boxes send boxes only rarely, since their limit is incremented.

Numerical results presented in [15] show that a method which uses receive-initiated-scheduling combined with dynamic send-initiated-scheduling is efficient. Through the usage of the sender-initiated-scheduling the total number of boxes handled for a given problem is reduced. The speedup for the three considered problems were 9.71, 19.58 and 11.97 on 16 processors and 15.04, 28.26 and 30.88 on 32 processors, respectively. So superlinear speedup was achieved for one problem (on 16 processors). The reason of superlinear speedup was not explained.

4.2.4 The Approach of Moore, Hansen and Leclerc (1992)

Characterization

list management	oldest-first
multisection parameter l	1
subdivision strategy	A
accelerating devices	midpoint test monotonicity test nonconvexity test interval Newton method

As opposed to the methods considered so far the parallelization of Moore, Hansen and Leclerc in [40] is based on the serial method that use the *oldest-first*-strategy for box selection. As it was shown in Section 2.1.1 this strategy handles some boxes unnecessarily. To accelerate the method along with the midpoint test (this is simply an evaluation of the range of the function at a midpoint of the box in order to get a better upper bound of the minimum) and the monotonicity test also the nonconvexity test and the interval Newton method were used. The parallel method was implemented on a workstation cluster of 250 Sparc-Stations SLC.

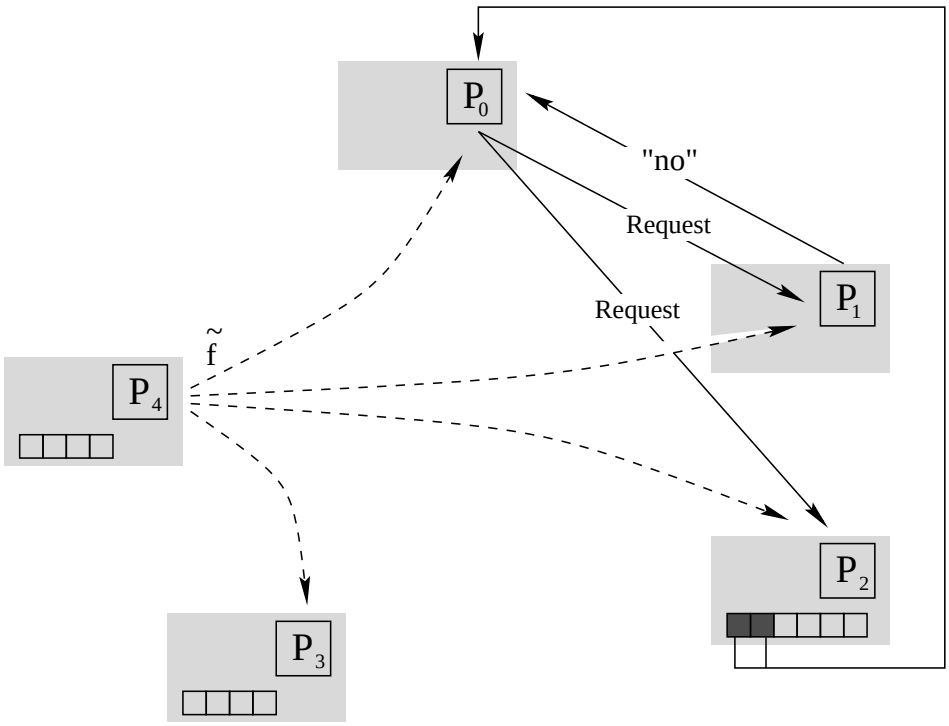


Figure 4.4: Communication structure in the parallelization by Moore, Hansen and Leclerc

Like in the approach of Eriksson every processor manages its own list. The best upper bound is sent through the broadcast command to all processors. If processor P_i has no more boxes to handle, then it sends a request to a randomly selected processor P_j. If processor P_j has boxes in the list, then it sends half of its boxes (but no more than a limit set a priori) to the processor which initiated the request. Otherwise P_i sends a request to $P_{(j+1) \bmod p}$, $P_{(j+2) \bmod p}$ and so on (see Figure 4.4).

Running this parallel method on a parameterized problem MHL (see [5]) superlinear speedup was achieved. A maximum speedup of 170 on 32 processors was achieved.

4.2.5 The Approach of Berner (1995)

Characterization

list management	best-first
multisection parameter l	2
subdivision strategy	C
accelerating devices	monotonicity test nonconvexity test interval Newton method

Berner's approach described in [5] was implemented on a CM-5, a MIMD computer with 32 processors. The parallel method is based on the serial method that uses the monotonicity test, the nonconvexity test and the interval Newton method as accelerating devices. As a box selection strategy the *best-first*-strategy is chosen. In this approach there is one centralized mediator and workers (see 4.5). Each worker manages its own list of boxes whose length is

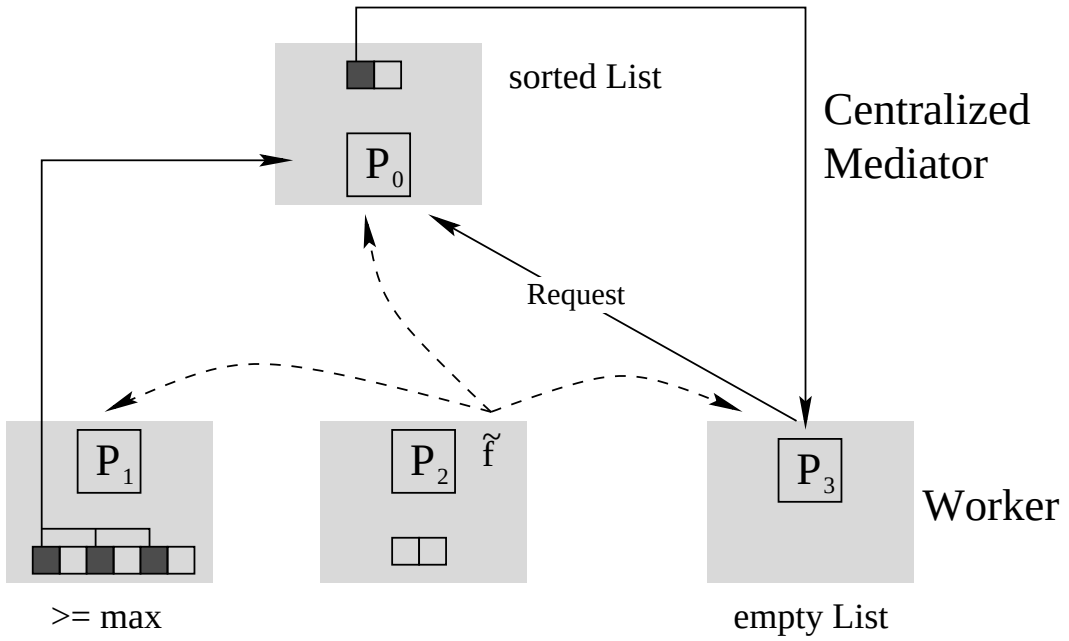


Figure 4.5: Communication structure by the parallelization by Berner

controlled by the centralized mediator. The centralized mediator waits for requests of idle processors to send them new boxes. Moreover, it keeps a limit **max**, that is changed dynamically. This limit is used to make sure that the centralized mediator does not run out of boxes, but also that not too many boxes are stored in its list. Processors which keep more than **max** boxes in their lists send some of them to the centralized mediator (see Figure 4.5).

The boxes to be sent to the centralized mediator are selected neither randomly nor from the tail. Every second box (at most **max-send** boxes are sent) is selected from the list. Each processor sends the best upper bound to all workers and the centralized mediator. An advantage of this parallel approach compared to the master-slave model used in Henriksen and Madsen is that there is less work for the centralized mediator than for the master. So it will not become a bottleneck if the number of processors used is not too large. Moreover, the whole memory including that of the workers is used. Compared to the approach of Eriksson and the one of Moore, Hansen and Leclerc, there is no need to request several processors to get boxes, if a processor becomes idle. Instead, it is the centralized mediator that directly responds to each request.

The method was run on 4, 8, 16 and 32 processors. For some test problems slight superlinear speedup was achieved.

4.2.6 The Approach of Wiethoff (1997)

Characterization

list management	best-first
multisection parameter l	2
subdivision strategy	C
accelerating devices	Monotonicity Test Nonconvexity Test Interval Newton Method

In [55] Wiethoff presented his distributed parallel method. It was implemented on IBM RS / 6000 SP. 96 processors were used. For the list management he used the *best-first*-strategy. His parallel method was based on a serial method with accelerating devices like the monotonicity test, the nonconvexity test and the interval Newton method. Boxes were subdivided into 4 parts when needed. And for this he applied strategy C. Processors were located in a pool logically arranged in a ring. All processors had their own lists of boxes. Each processor communicates only with its 4 neighbors (two nearest and two next to nearest). On every processor two processes run. One for load balancing and exchange of the best upper bound. The other is for handling local boxes. The newly found better upper bound was sent only to neighbors and from there propagated further. This method was run on 4, 8, 16, 32, 64, 96 processors. 18 problems were tested. For a few problems superlinear speedup was achieved. On 8 processors for 2 problems, on 16 processors for 3 problems, on 32 and more processors only for one problem. The method has no bottleneck at all. But the larger the number of processor the lower the efficiency. Information is distributed very slowly.

4.3 A New Parallel Approach: a Challenge Leadership Model

Characterization

list management	best-first
multisection parameter l	2
subdivision strategy	C
accelerating devices	Monotonicity Test Nonconvexity Test Interval Newton Method

In this section we present a new parallel method for verified global optimization. In the method we assume that the global minimum is not known a priori so that a sequential update of the upper bound for the global minimum is required. The method is based on the serial algorithm 2.8 described in Section 2.5.5.

4.3.1 General Conditions for the Parallelization

For the development of the verified global optimization method a cluster of 8 Ultra 10 workstations from Sun Microsystems was available. On individual computers software supporting the MPI interface with asynchronous communication was installed. In order to accelerate the data exchange we have introduced user-defined structures within MPI. The method has no constraints on the number of processors. But for comparison reasons we test the parallel program essentially for numbers of processors which are powers of 2, i.e. $p = 1, 4, 8, 16$. In the case of $p = 16$ we run 16 processes on 8 computers, two processes on each computer. For

performance evaluation we consider only the user time (the time of the program execution without the time spent for starting a process and its finalization). The source code of this parallel program can be compiled without any changes on any system supporting the MPI interface and having C-XSC installed. We also take into account that messages from different processors may arrive in a different order than initiated. When a message from a processor claiming to be a leader arrives we store it in a queue. When the leadership changes we search the queue of stored messages for messages with a leader set to the actual leader and retrieve them in the order they were inserted. According to the MPI standard messages from the same processor arrive in the same order. Therefore messages from the current leader arrive in the right order and the information is always consistent.

4.3.2 Description of the Communication Model

In the design of the challenge leadership model we tried to use the advantage of the *best-first*-strategy and advantages of the centralized management. We also tried to reduce the number of interprocess communications.

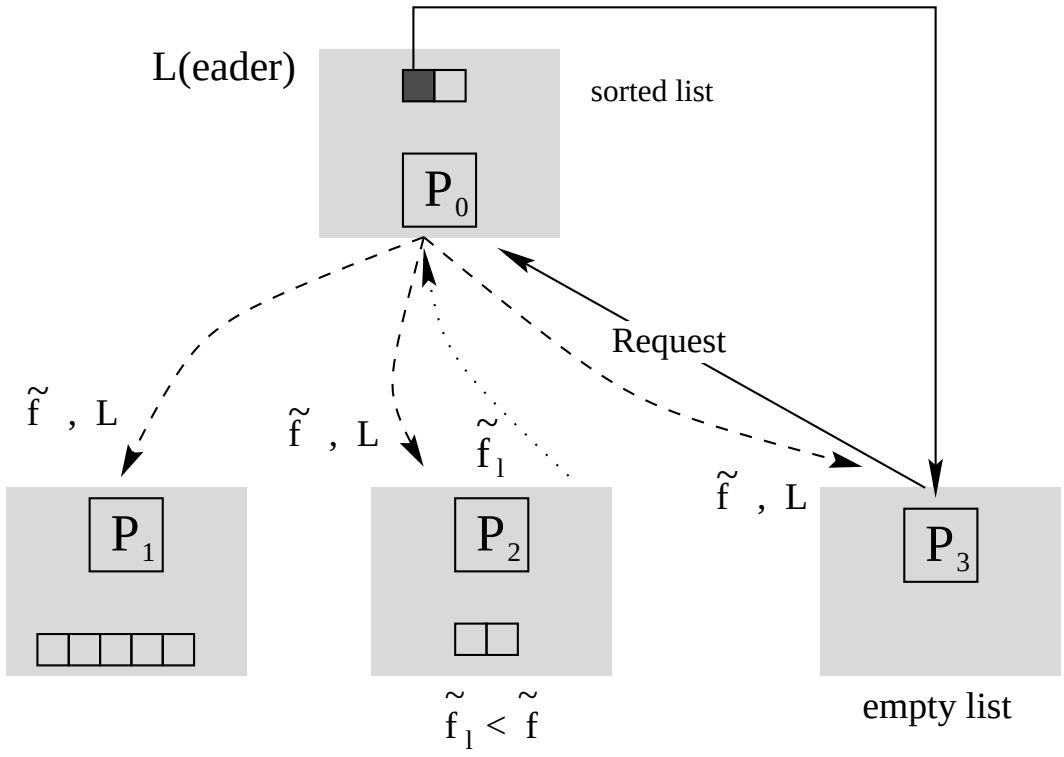


Figure 4.6: Communication structure by the challenge leadership model

In this model there is always one leader and workers (see Figure 4.6). The leader is determined as the processor holding the smallest best upper bound. The leader has boxes for handling. Therefore idle processors send requests to the leader. Every idle processor sends only one request, it sends it to the leader. This reduces the number of possible retransmissions, since the leader has at least one box which will be subdivided if there are incoming requests.

When a processor obtains a better best upper bound, it sends a "challenge" to the leader, but

not to the other processors. The leader determines the smallest of the best upper bounds, if it receives several of them, and decides who is the next leader. It sends the new best upper bound together with the information on change of the leadership in one message to all other processors. If the leader has the last box and it fulfills the termination criterion, it inserts it into the solution list and chooses any non-idle processor as the next leader. In the case, that there is no non-idle processors left, it sends a termination signal to all the processors and the method ends.

Opposite to Berner's approach there is no need to transfer boxes between processors in advance. When a processor needs a box it gets one.

One advantage of this approach is that when an idle processor requests a box it gets it from the list of a processor with the smallest best upper bound. To reduce the number of idle processors waiting for a box from the leader, we switch the leader from handling boxes to serving requests if the number of requests waiting to be served exceeds the static constant **limit**. For sending new best upper bounds (by the leader and other processors), for sending boxes we use asynchronous communication (see Section 4.3.3) to increase the performance of the sender.

The new method is based on the serial method described in section 2.5.5. If the list length is large enough then performing the cut-off tests becomes a bit expensive if it is done often, since we have to run through the whole list. By some approaches (for instance, centralized mediator) all processors send their updated upper bounds to the rest of processors. Every processor receives many upper bounds and performs the cut-off test for each of them. In the new method the centralized distribution of the best upper bound comes to help. It reduces the number of cut-off-tests eliminating the intermediate unnecessary cut-off-tests. In the following sections the method based on the challenge leadership approach is presented as a pseudo code and explained.

4.3.3 Used Communication Routines

In order to present the parallel method as a pseudo code we first introduce notations for the communication routines used in the method. First we list the passing parameters that appear in the routines:

i: is the number of the processor to which the message is sent or from which the message is received. **?** in a receive routine means that the message to be received can be from any source.

tag: In a send routine the message with tag **tag** is sent. In the receive routine only messages with that tag is received. **?** in a receive routine means that a message with any tag is received.

variable In the send routine it is the storage where the data to be sent is stored. In the receive routine it is the storage where the new data will be stored. The type of variable determines the amount of storage needed.

Now we can describe the communication routines:

receive-block $(i, tag, variable)$
Blocking receive, the processor waits until the matching message arrives.

send-async $(i, tag, variable)$
Nonblocking send, the processor sends the message and immediately continues his work

not waiting for the receiver to receive the message. In this routine one should be careful when rewriting *variable*. One must check if the message has been received before.

send-block $(i, tag, variable)$
Blocking send, the processor sends the message and waits until the receiver acknowledges the reception. In this routine one can safely rewrite *variable* after the return from the routine.

last-message-sent (i, tag)
Checks if the last message sent to processor i with tag *tag* has been received. **?** as the first argument means that the processor checks if all messages with tag *tag* sent to different processors have been received.

message-wait (i, tag)
Checks if there is a pending message from processor i with tag *tag*. **?** as a first argument means that messages from all processors are taken into account. **?** as a second argument means that messages with any tag are considered.

A **send-async** call is an asynchronous communications. An asynchronous **send-async** call initiates the send operation, but does not complete it. The **send-block** call will return before the message was copied out of the send buffer. The seperate check is needed to verify that the data has been copied out of the send buffer. A blocking **send-async** call initiates the send operation and waits until the operation completes. The operation completes when the corresponding receive is initiated.

In the communication routines the following tags are used:

tag-box the message contains data of type MPI_BOX. The message contains only one element, since there is only one box sent per request.

tag-best the message contains data of type MPI_BEST. The message contains exactly one element. The element contains the new value of $\tilde{f}$ from the leader and, signalized by a flag, the rank of a new leader.

tag-empty the message contains data of type MPI_EMPTY. This message can be sent only by the leader indicating that he worked out his boxes and appoints a new leader. It also contains status data showing which processors requested boxes.

tag-get the message contains data of type MPI_GET. The message contains exactly one element. But this one element contains all requests not served by the processor which is the originator of this message.

tag-solution the message contains data of type MPI_BOX. The message can contain more than one element, it can also contain no elements. These messages are used in the end phase to collect solutions from the processors.

4.3.4 A New Parallel Method

In this subsection we describe the implementation details including all routines of our proposed parallel method. To simplify the implementation of the method we subdivided the whole algorithm into many small functions. The overhead due to calling all these functions is close to nothing.

The new parallel method for verified global optimization is given in Algorithm 4.1.

Algorithm 4.1 A New Parallel Method for Verified Global Optimization

Input: the initial box $[x]^0$, inclusion functions F, F' and F'', the number of processors p;
Output: an enclosure for the global minimum f^* and the list $\tilde{L}$ of boxes which contain all global minimizers;

MPI_Comm_rank(MPI_COMM_WORLD, &*my_rank*); {an MPI function which returns the rank of the processor}
$[x]$ = getStartBox($[x]^0$, p, *my_rank*);
(*leader*, $\tilde{f}$) = distributeUpper(*my_rank*, $[x]$);
$L = \{([x], \inf(F([x])))\}$;
$\tilde{L} = \varnothing$;
repeat
  repeat
    take the first element $([x], \inf(F([x])), t)$ from L;
    beforeHandle();
    handleBox(L, $\tilde{L}$, $\tilde{f}$, $([x], \inf(F([x])), t)$);
    afterHandle();
    if $\tilde{f}$ has changed since the last cut-off-test **then**
      L = cut-off(L, $\tilde{f}$);
    end if
  until $L = \varnothing$
until inWait() = true
$\tilde{L}$ = cut-off($\tilde{L}$, $\tilde{f}$);
$\tilde{L}$ = collectSolution();
if $\tilde{L} \neq \varnothing$ **then**
  take the first element $([x], \inf(F([x])))$ from $\tilde{L}$;
  set $f^* = [\inf(F([x])), \tilde{f}]$;
  return f^* and $\tilde{L}$;
end if

It takes the initial box $[x]^0$, inclusion functions F, F' and F'' for the given function f, its gradient and its Hessian respectively, and the number of processors as an input. As a result it returns an enclosure for the global minimum f^* and the list $\tilde{L}$ of boxes which contain all global minimizers.
Before processors start working they divide the initial box into p parts through the call to getStartBox (see Algorithm 4.2).

Algorithm 4.2 getStartBox

Input: the initial box $[x]^0$, p - the number of processors, *my_rank* - the rank of this processor;
Output: the subbox $[x]$ that corresponds to the processor *my_rank*;

compute the factorization $p = r \cdot 2^l$, where r is odd;
determine $l + 1$ directions for subdivision; {we use strategy C}
subdivide the box $[x]^0$ into p parts;
return *my_rank*-th box;

In this function, first, the factorization $p = r \cdot 2^l$, where r is odd, is computed. Using strategy C we determine the order in which edges should be subdivided. The first edge we subdivide

into r parts. For the next l edges (if $l+1 > n$, n is the dimension, we select edges in cycle) we simply double the number of parts. Depending on the rank of the processor the function returns the corresponding part. In this manner, we obtain individual boxes for which the ratio of the lenghts in each direction remains similar to theat of the original box. After the box is subdivided each processor computes the range of the given function over its subbox. It initializes the upper bound for the minimum to the upper bound of the range. After that we call the function distributeUpper (see Algorithm 4.3) to determine the leader and distribute the verified upper bound for the global minimum. In the function the rank r and the local best upper bound $\tilde{f}$ is stored into *local*, which is of the data type MPI_DOUBLE_INT, predefined in MPI (see [36]). Then using the collective communication $MPI\_Allreduce$ with the option MPI_MINLOC we distribute the rank of the processor holding the smallest value for the upper bound and the value itself. If there are several processors, then the processor with the smallest rank is chosen. The *leader* variable is set to this rank and $\tilde{f}$ to the global best upper bound. After the leader is determined and the global upper bound is distributed we initialize the local working list L to $([x], \inf(F([x])))$ and the solution list $\tilde{L}$ to the empty list. Here $[x]$ corresponds to this processor's box.

Algorithm 4.3 distributeUpper

Input: the rank of the processor *my_rank*, the box $[x]$;
Output: the rank of the leader processor *leader*, the best upper bound $\tilde{f}$;

$local.rank = my\_rank$;
$local.f = Sup(F([x]))$;
Allreduce(*local*, *global*, MPI_MINLOC); {this is the call to MPI function MPI_Allreduce}
$leader = global.rank$;
$\tilde{f} = global.f$;
return *leader*, $\tilde{f}$;

Now all the initialization is done and we turn to the calculation. We take the first element from L. Through the call to the function beforeHandle (see Algorithm 4.4) we check for incoming requests for boxes or the change of the leader. If there are incoming requests it is efficient to serve these request in order to keep other processors doing a useful job. Therefore we serve them first. In the function beforeHandle we first check if we have processors requesting for boxes. This information is stored in the local status of the processors. If there are such processors we send boxes from the working list, as long as there are boxes. Then we receive pending messages in the same order in which they came. The order of arrival is important for information consistency. If there is a message with the **tab-best** tag we call the function serveBest (see Algorithm 4.5). There are two possible actions. First, this processor is the leader and the incoming message is the challenge. Then if the upper bound stored in the message is smaller than the current upper bound we check if there is already a pretender and if so we choose from the challenger and the pretender the one with the smallest upper bound, else we set the challenger as a pretender. Second, this processor is not the leader. Then the message is either an update of the upper bound or the appointment of a new leader. In these cases we simply update the local information: the actual value of the upper bound and the rank of the leader.

If there is a message with **tab-empty** tag we call the function serveEmpty (see algorithm 4.6). There we set new values for *leader* and upper bound. We also update the local status of other processors calling updateStatus (see Algorithm 4.7). In the local status of the processor we store requests from processors. We also indicate there which processor is idle and has

Algorithm 4.4 beforeHandle

Input: no input;
Output: no output;

```
if have a pending request then
  while L ≠ ∅ and have pending requests do
    take element req from the request queue; {an element req of the request queue has a
    field proc, the rank of the processor initiated this request}
    take the element box = ([x], lbf, t) from L;
    send-async(req.proc, tag-box, box);
  end while
end if
while message-wait(?, ?) do
  if message-wait(?, tag-best) then
    serveBest(r); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-empty) then
    serveEmpty(r); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-get) then
    serveGet(r); {r is the rank of a processor whose message is received}
  end if
end while
leaderUpdate(leader);
if best upper bound has changes since the last cut-off test then
  L = cut-off(L, f̃);
end if
```

Algorithm 4.5 serveBest

Input: the rank r of the processor whose message is pending;
Output: no output;

```
receive-block(r, tag-best, new_best);
if new_best.status = status-update then
  f̃ = new_best.f;
else if new_best.status = status-new then
  f̃ = new_best.f;
  leader = new_best.leader;
else if my_rank = leader then
  if f̃ > new_best.f then
    if there is a pretender then
      choose from the pretender and the challenger the one who has smallest upper value;
    else
      set the challenger as a pretender;
    end if
  end if
end if
```

Algorithm 4.6 serveEmpty

Input: the rank r of the processor whose message is pending;
Output: no output;

receive-block(r, tag-empty, $empty - message$);
$leader = empty - message.leader$;
$\tilde{f} = empty - message.f$;
updateStatus($empty - message.status$);

requested a box, but not from this processor. This happens when the processor already sent a message with tag **tag-get** and now either was appointed as a leader and changes leadership or retransmits someone's request. Status-empty in most cases prevents an idle processor to be appointed as a leader.

Algorithm 4.7 updateStatus

Input: array of p elements $latest\_status$ indicating statuses of processors;
Output: no output;

for $i = 0$ **to** $p - 1$ **do**
  if $latest\_status[i]$ = status-get **then**
    $procStatus[i]$ = status-get;
    insert the new element i into the request queue;
  else if $latest\_status[i]$ = status_empty and $procStatus[i]$ = status-full **then**
    $procStatus[i]$ = status-empty;
  end if
end for

A message with tag **tab-empty** means that the leader has no more boxes to handle and sets the other non-idle processor as a leader. Such message also encapsulates the requests of boxes that the leader has received.
If there is a message with the **tab-get** tag we call the function serveGet (see Algorithm 4.8). In serveGet we merely receive the message and update the local status of processors calling updateStatus. After all incoming messages are served we call the function leaderUpdate (see Algorithm 4.9). Here if the processor is the leader and there is a pretender then the pretender is set as a leader and a message is sent to all processors with the **tag-best** tag and with the flag set to **new**. If the processor is not a leader then the challenge message is sent to the leader.

After all messages are served we check if the best upper bound is changed either locally or on another processor. If it is changed then we perform the cut-off test to discard unnecessary boxes.

Algorithm 4.8 serveGet

Input: the rank r of the processor whose message is pending;
Output: no output;

receive-block(r, tag-get, $get\_message$);
updateStatus($get\_message.status$);

Now we can go further. We call the function handleBox for handling the box. For the details

of handling see Algorithm 2.9 in Section 2.5.5.

Algorithm 4.9 leaderUpdate

Input: the rank of the leader l;
Output: no output;

```
if my − rank = leader then
  if last-message-sent(?, tag-best) then
    if l ≠ my − rank then
      best − message.status = status-new;
    else
      best − message.status = status-update;
    end if
    best − message.leader = l;
    best − message.f = f̃;
    for i = 0 to p − 1 do
      if i ≠ my − rank then
        send-async(i, tag-best, best − message);
      end if
    end for
  end if
else if I have better upper bound then
  if last-message-sent(leader, tag-best) then
    send-async(leader, tag-best, challenge);
  end if
end if
```

After the box handling we receive incoming messages through the call to the function afterHandle (see Algorithm 4.10). In this function we only receive incoming messages through the

Algorithm 4.10 afterHandle

Input: no input;
Output: no output;

```
while message-wait(?, ?) do
  if message-wait(?, tag-best) then
    serveBest(r); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-empty) then
    serveEmpty(r); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-get) then
    serveGet(r); {r is the rank of a processor whose message is received}
  end if
end while
leaderUpdate(leader);
```

call to functions serveBest, serveEmpyy, serveGet and leaderUpdate which are already described. If the upper bound is changed either locally or on another processor we perform the cut-off

test to discard unnecessary boxes.
As long as the working list is not empty we take one element from the list and do everything again. If it is empty we go waiting calling the function inWait (see Algorithm 4.11). If the

Algorithm 4.11 inWait

Input: no input;
Output: a flag indicating if the method terminates;

```
if myrank = leader then
  if givingUp() = true then
    return true;
  end if
else
  store all received requests in getmessage;
  send-async(leader, tag-get, getmessage);
end if
while message-wait(?, ?) do
  if message-wait(?, tag-best) then
    serveBest(r); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-empty) then
    serveEmpty(:); {r is the rank of a processor whose message is received}
  end if
  if message-wait(?, tag-get) then
    recv-block(i, tag-get, get_message);
    for i = 0 to p − 1 do
      if get_message.status[i] = status-get then
        procStatus[i] = status-get; insert an element i into the request queue;
      end if
    end for
  end if
  serveGet(i);
  if message-wait(?, tag-box) then
    recv-block(i, tag-box, box);
    insert box into L;
    return false;
  end if
  if message-wait(?, tag-finish) then
    return true;
  end if
  if have pending requests then
    for i = 0 to p − 1 do
      if procStatus[i] = status-get then
        get_message.status[i] = status-get; procStatus[i] = status-empty;
      end if
    end for
    send-async(leader, tag-get, get_message);
  end if
end while
```

Algorithm 4.12 givingUp

Input: no input;
Output: a flag indicating if the method terminates;

```
if exists i with procStatus[i] = status-full and i ≠ my_rank then
  leader = i;
  best_message.status = status-new;
  best_message.leader = leader;
  best_message.f = f̃;
  for i = 0 to p − 1 do
    if procStatus[i] = status-get then
      best_message.status[i] = status-get;
      procStatus[i] = status-empty;
    end if
  end for
  best_message.status[my_rank] = status-get;
  procStatus[my_rank] = status-empty;
  for i = 0 to p − 1 do
    if i ≠ my_rank then
      send-async(i, tag-best, best_message);
    end if
  end for
else
  for i = 0 to p − 1 do
    if i ≠ my_rank then
      send-async(i, tag-finish, finish);
    end if
  end for
  return true;
end if
return false;
```

processor is the leader it calls the function givingUp (see Algorithm 4.12). There it selects one of the non-idle processors as a leader and informs all processors about a leadership change. It also sends requests for boxes in the same message.
The existence of a non-idle processor is checked by the number of requests and the local status of processors. If there is no such processor then all processors are informed about the final phase: solution collection.
If the processor is not a leader it stores all received requests in one message, including its own request, and sends this message to the leader.
After this we start receiving messages. If messages with tags **tag-best**, **tag-empty** or **tag-get** are received we call one of the corresponding functions serveBest, serveEmpty, serveGet. In between we retransmit collected requests for boxes.
This continues until either the message with a box or the message indicating the termination is received.
When all boxes are handled the function collectSolution (see Algorithm 4.13) is called to collect solutions. The function **collectSolution** is the final phase of the method. Here the processor with rank 0 collects solutions from other processors. The processor with rank 0 requests solutions from all processor in turn. Solutions are sent in blocks of at most MAX_SOL solutions.

Algorithm 4.13 collectSolution

Input: no input;
Output: the solution list if the rank is 0;

```
if my_rank ≠ 0 then
  for i = 1 to p − 1 do
    recv-block(i, tag-solution, sol);
    while number of received boxes is not null do {the maximal number of elements to be
    received at once is MAX_SOL}
      insert received elements into L̃;
      recv-block(i, tag-solution, sol);
    end while
  end for
  return ∅;
else
  while L̃ ≠ ∅ do
    take at most MAX_SOL elements from L̃;
    send-block(0, tag-solution, sol);
  end while
  return L̃;
end if
```

4.3.5 Numerical Results

In this section we present numerical results of the new method. For time measurement we use Standard Time Unit (STU). One STU is the time spent for evaluation of the Shekel 5 function with real arguments 1000 times. It is surely the compiler dependent. Using C-XSC on a Sun Ultra 10 one STU is 0.375 seconds.

We analyze the reason of superlinearity of the new method for some problems. Descriptions of all problems can be found in Appendix A along with their solutions.

Since the run time depends on the software and hardware in use, for fair comparison we implemented also the centralized mediator approach. We have chosen this method since it is the most efficient of all prior existing methods. Also it can be implemented quite easily.

Here we notice again that in the case of $p = 16$ we run 16 processes on 8 processors. It is not the same productivity as on 16 processors.

In Table 4.1 we give run times for the new parallel method. In the table in parenthesis we give efficiency in percentage. We do not expect superlinearity or even linearity for problems for which the serial method requires less than 100 iterations.

For the problems L12 and L18 the serial method needs 36 and 30 iterations, respectively. In the parallel method with 16 processors in the best case each processor would handle about 2 boxes. But because of the relatively slow communication and due to the parallelization, the number of boxes handled is sometimes much larger (see Table 4.2). Also the time for the start phase and finalization is long compared to the time spent purely for handling boxes. For both problems we have poor parallelization.

The same is valid for the problems G7 and G10.

For problems GP, S2.14, KOW, INF1 we have superlinear speedup. For these problems the serial method requires thousands of iterations and therefore they are well parallelizable.

Problem	Serial		Parallel					
			p = 4		p = 8		p = 16	
	Iters	Time	Time	(%)	Time	(%)	Time	(%)
R4	265	10.05	2.59	(97)	2.11	(59)	1.25	(50)
L12	36	13.23	8.53	(38)	5.66	(29)	11.07	(7)
L18	30	7.20	9.74	(18)	13.97	(6)	11.84	(3)
G7	42	5.60	33.99	(4)	6.06	(11)	9.13	(3)
G10	44	9.15	9.10	(25)	9.77	(11)	9.03	(6)
GP	9193	89.55	21.00	(106)	10.47	(106)	4.13	(135)
H6	307	23.49	11.50	(51)	5.00	(58)	4.42	(33)
S2.14	9024	93.60	23.03	(101)	4.35	(268)	4.42	(132)
GEO1	351	14.45	7.43	(48)	3.21	(56)	2.63	(34)
GEO2	443	21.20	7.03	(75)	2.94	(90)	2.63	(50)
GEO3	511	25.33	8.99	(70)	4.13	(76)	3.35	(47)
JS	226	39.95	12.57	(79)	7.21	(69)	4.57	(54)
S2.7	98	31.39	16.19	(48)	10.52	(37)	9.57	(20)
L3	531	120.19	31.05	(96)	12.50	(120)	9.28	(80)
R8	1145	205.79	110.22	(46)	51.73	(49)	48.97	(26)
HM3	697	161.89	42.32	(95)	23.03	(87)	11.85	(85)
HM4	2996	1088.08	298.79	(91)	152.49	(89)	79.61	(85)
KOW	193301	34114.24	3057.00	(278)	992.66	(429)	658.29	(323)
WK	163895	5821.49	134.32	(1084)	54.96	(1324)	40.96	(888)
INF1	57309	3275.97	285.06	(287)	38.85	(1054)	22.49	(910)

Table 4.1: Times for test problems on a cluster of Sun workstations using the challenge leadership approach

To solve these problems we need a lot of memory. But this is not the only reason for the superlinearity. In Table 4.2 we see that for these problems the number of handled boxes in the parallel method is less than the number of handled boxes in the serial method. For instance, for the problem GP the number of boxes handled in the parallel method is less than in the serial method by a factor of 2. For the problem WK the factor is even larger, it is more than 3. Therefore even with overhead for slow communication we obtained superlinear speedup for these problems. On the other hand for problems S2.14, KOW, INF1 the number of boxes handled in the parallel method are only slightly less than in the serial method. Nevertheless the speedup is very impressive: 1.32×16 for S2.14, 3.23×16 for KOW and 9.10×16 for INF1 on 16 processors. On 4 and 8 processors the parallel method achieves even better speedup. This impressive speedup is due to the intensive use of memory. For these problems the serial method manages very long lists of boxes, which do not fit into memory. Therefore every call to the cut-off test is followed by swapping to and from the hard disk. And this increases the run time significantly. In Table 4.2 we also see that for problems HM3 and HM4 the serial method handles less boxes than the parallel method. These numbers are very close and we have for them almost linear speedup.

We grouped problems, with exception of those for which the serial method requires less than 50 iterations, into four groups and plotted graphs for them.

In Figure 4.7 we see that the speedup changes better than linearly until 8 processors. Then on 16 processors it gets worse. It is due to the run of 16 processes on 8 processors.

In Figure 4.8 we observe the same behavior for three of four problems: GEO2, GEO3, JS. For

Problem	Serial				Parallel with $p = 16$			
	f eval	g eval	H eval	boxes	f eval	g eval	H eval	boxes
R4	493	396	104	265	699	539	151	397
L12	136	46	10	36	2806	847	8	839
L18	109	42	12	30	2107	782	114	668
G7	163	44	2	42	3339	844	3	841
G10	168	47	3	44	2875	728	3	739
GP	10148	13516	4323	9193	4988	6735	2430	4413
H6	625	404	97	307	1000	741	161	722
S2.14	16923	13012	3988	9024	16332	12012	3658	8401
GEO1	1310	376	25	351	2457	1250	39	1252
GEO2	1358	687	244	443	1710	918	311	651
GEO3	1755	774	263	511	2430	1358	406	1001
JS	344	334	108	226	299	446	110	352
S2.7	302	145	47	98	667	754	162	599
L3	1741	655	124	531	1711	694	117	582
R8	1613	1195	50	1145	9024	3995	75	3950
HM3	1715	1047	350	697	1700	1054	356	727
HM4	7088	4626	1630	2996	7254	5008	1822	3211
KOW	219448	269815	76514	193301	201046	254478	74558	179966
WK	164025	273105	109210	163895	49988	85037	34497	50683
INF1	163717	81865	24556	57309	163644	63297	21821	41501

Table 4.2: Number of function, gradient, Hessian evaluations by the serial and parallel methods

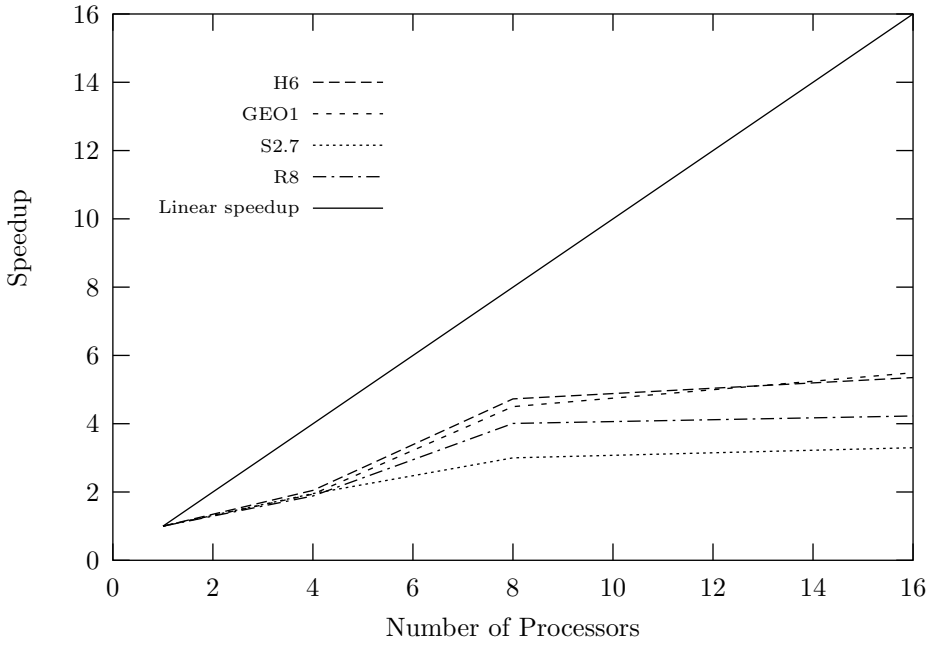


Figure 4.7: Problems which are bad parallelizable

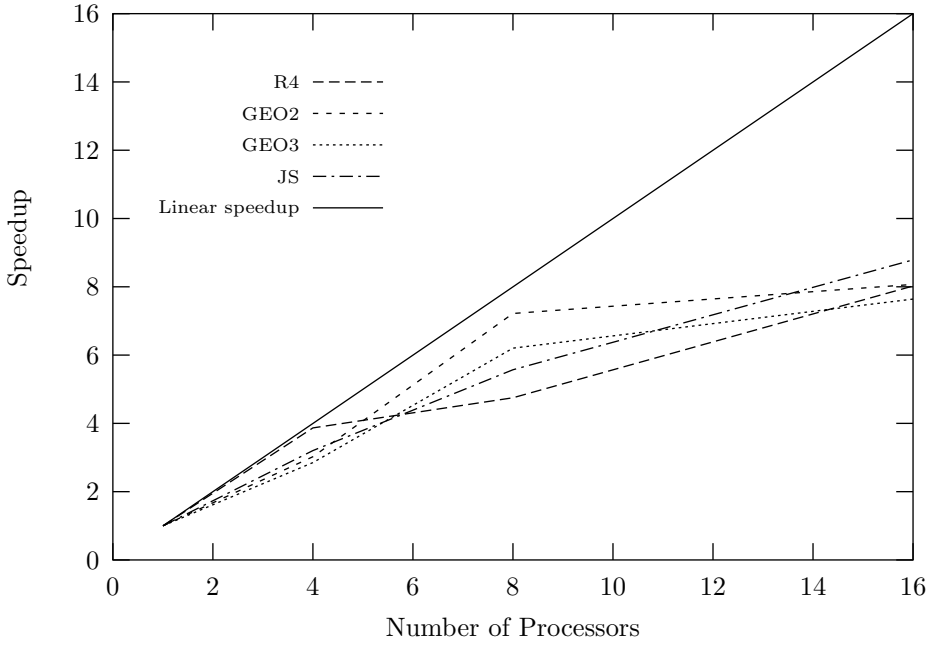


Figure 4.8: Problems with relatively good parallelization

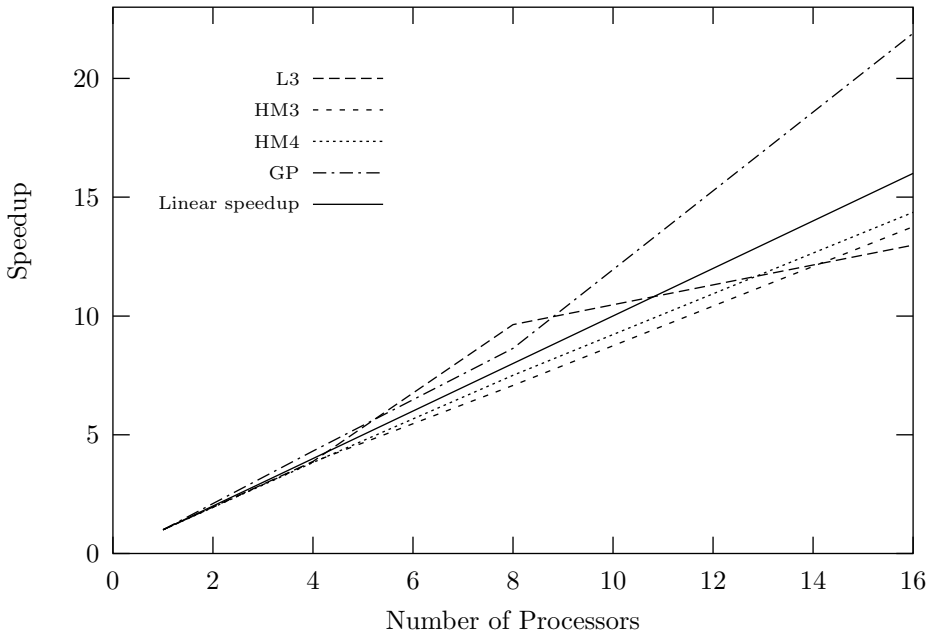


Figure 4.9: Problems with relatively good parallelization

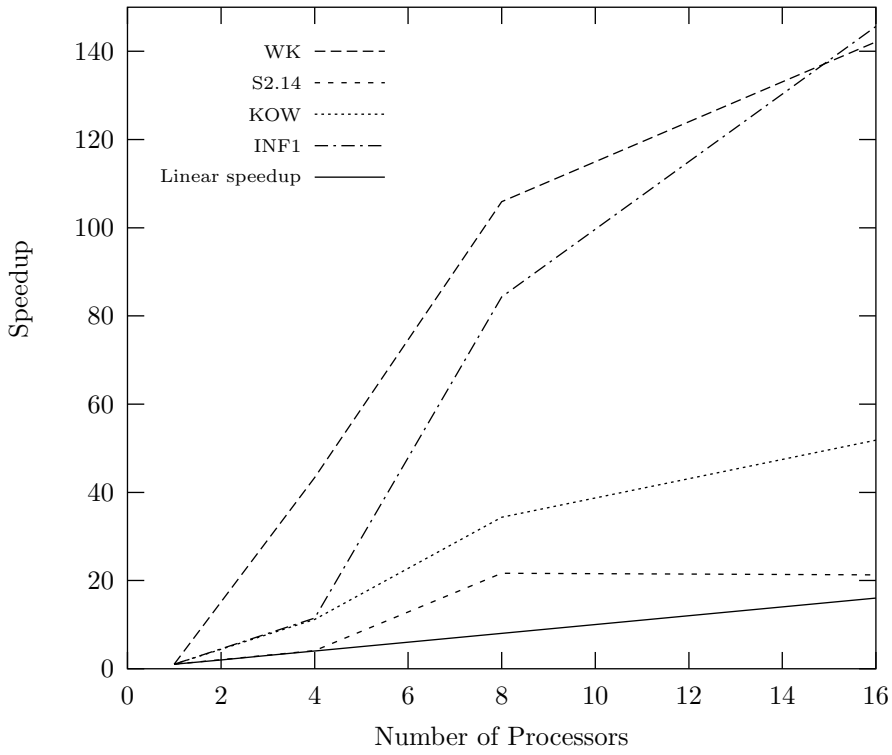


Figure 4.10: Very good parallelizable problems

the problem R4 the speedup becomes worse even on 8 processors.

In Figure 4.9 we see that all problems have nearly linear speedup. The speedup changes linearly until 8 processors for all problems. For three of four problems - HM3, HM4, GP - the speedup is still linear even for 16 processes, despite of the usage of 8 processors. For the problem L3 the speedup gets worse for 16 processes on 8 processors, as usual.

In Figure 4.10 we plotted speedup for hard problems. For them we have superlinear speedup for all numbers of processors (for $p = 16$ there are 16 processes on 8 processors). And, as usually, the is a speedup drop from 8 to 16 processes.

As we see from Figures 4.7, 4.8, 4.9 and 4.10 we have a significant drop of speedup on 8 processors only for the problem R4. For the rest of the problems the speedup gets better or remains almost the same.

4.3.6 Comparison

In Table 4.3 we presented run times for all problems by the centralized mediator approach. As we can see from the table the centralized mediator approach has achieved superlinear speedup only for two problems - KOW and WK. The solution of this problem requires very much memory. For many problems the speedup decreases dramatically when the number of processor increase. For instance, for problems H6, GEO2, S2.7, L3, HM4 there is no improvement of time at all changing from 8 processors to 16 processor. It is due to the communication overhead of the method.

Problem	Serial		Parallel					
			p = 4		p = 8		p = 16	
	Iters	Time	Time	(%)	Time	(%)	Time	(%)
R4	265	10.05	5.47	(45)	5.47	(22)	8.88	(7)
L12	36	13.23	9.60	(34)	9.81	(16)	16.24	(5)
L18	30	7.20	5.31	(33)	5.26	(17)	7.85	(5)
G7	42	5.60	4.85	(28)	3.93	(17)	6.68	(5)
G10	44	9.15	7.92	(28)	6.49	(17)	10.81	(5)
GP	9193	89.55	34.89	(64)	16.59	(67)	25.61	(21)
H6	307	23.49	8.16	(71)	5.60	(52)	5.37	(27)
S2.14	9024	93.60	55.71	(42)	14.45	(80)	39.17	(14)
GEO1	351	14.45	6.56	(55)	4.29	(42)	3.47	(26)
GEO2	443	21.20	6.63	(79)	3.33	(79)	3.35	(39)
GEO3	511	25.33	7.93	(79)	4.08	(77)	3.87	(40)
JS	226	39.95	15.36	(65)	13.39	(37)	17.93	(13)
S2.7	98	31.39	16.09	(48)	10.16	(38)	9.19	(21)
L3	531	120.19	32.51	(92)	18.19	(82)	18.37	(40)
R8	1145	205.79	59.93	(85)	45.64	(56)	53.63	(23)
HM3	697	161.89	57.27	(70)	146.92	(13)	127.20	(7)
HM4	2996	1088.08	512.23	(53)	130.48	(104)	140.06	(48)
KOW	193301	34114.24	2027.51	(420)	1229.87	(346)	1184.26	(180)
WK	163895	5821.49	233.81	(622)	174.60	(417)	177.19	(205)
INF1	57309	3275.97	2556.41	(32)	3898.67	(10)	4348.17	(4)

Table 4.3: Times for test problems on a cluster of Sun workstations using centralized mediator approach

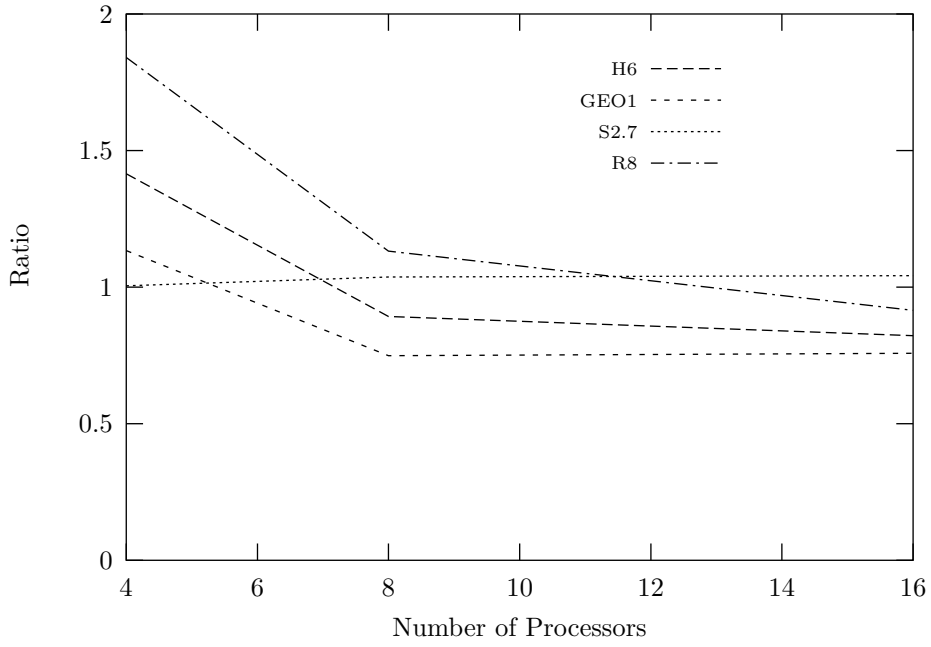


Figure 4.11: Problems which are bad parallelizable

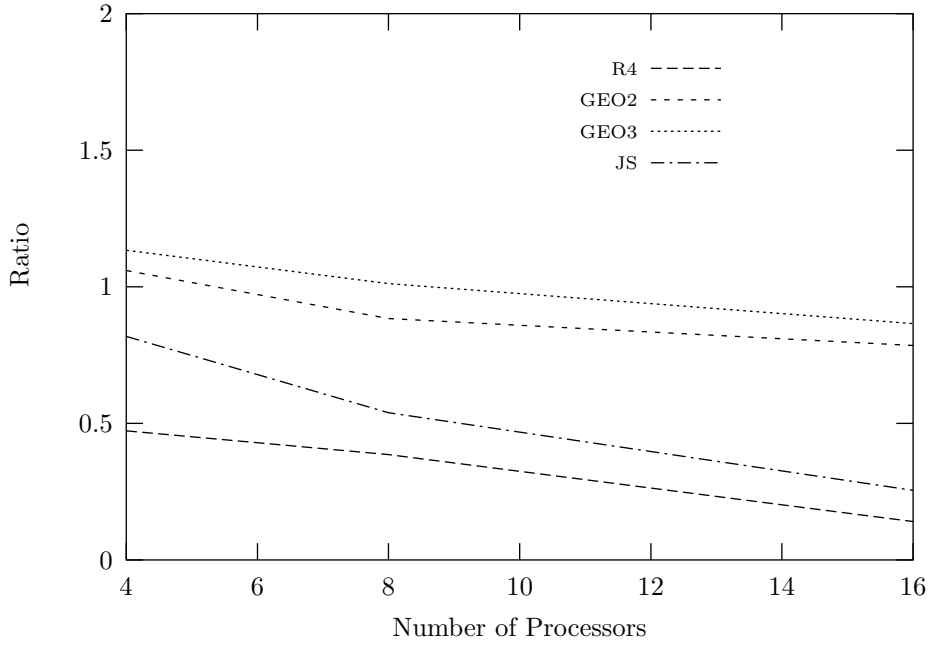


Figure 4.12: Problems with relatively good parallelization

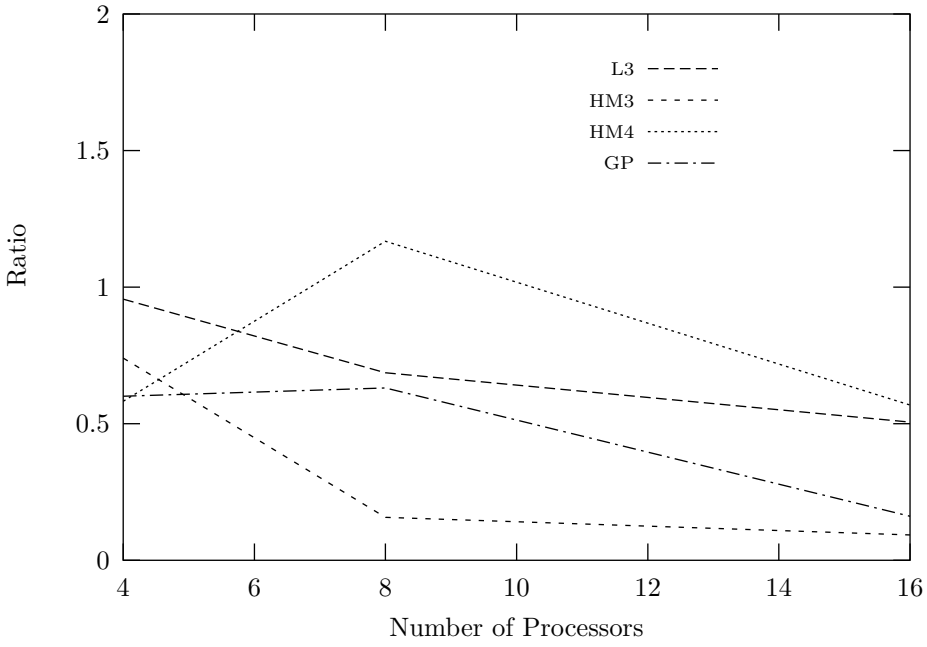


Figure 4.13: Problems with relatively good parallelization

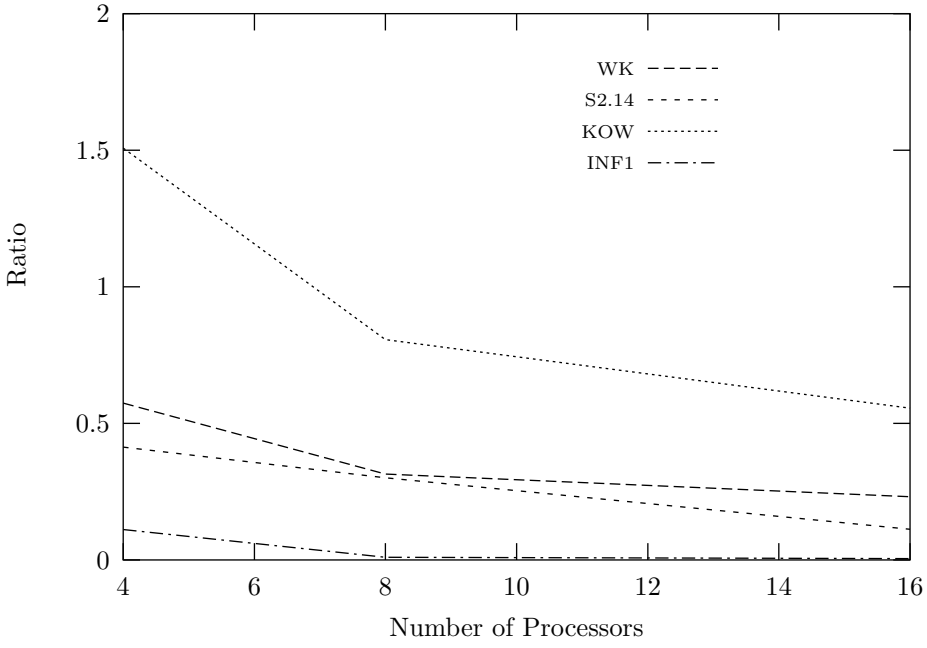


Figure 4.14: Very good parallelizable problems

In the new parallel method this overhead is reduced through the centralized propagation of the upper bound and the box transference on demand: the box is transmitted only when there is request message for a box.

To show the efficiency of the new approach we computed the ratio of the time spent by the centralized mediator to the time spent by the challenge leadership approach for each problem. This ratio gives us which part of the time spent by the centralized mediator method is required by the new method for the same problem.

In Figure 4.11 we see that the centralized mediator is good on 4 processors for 3 problems: H6, GEO1 and R8. On 8 processors the centralized mediator is good only for 2 problems. And running with 16 processes it is slightly better than the challenge leadership for one problem - S2.7.

In Figure 4.12 the centralized mediator method is better than the new method for problems GEO2 and GEO3 on 4 processors. But for these problems the new method becomes better when the number of processors increase. For 16 processes the new method is preferrable for all four problems.

An even interesting behavior we observe in Figure 4.13. The centralized mediator method is worse for all four problems on 4 processors. On 8 processors it becomes better for the problem HM4. But for 16 processes it is significantly bad again for all four problems.

In Figure 4.14 we see that the centralized mediator method gets worse, as ussually, when the number of processors increases.

Finalizing the comparison we emphasize that the challenge leadership approach is better than the centralized mediator, especially when the number of processors increase. Sometimes this difference becomes significant (see problems S2.14, GP, HM3, WK, INF1).

4.3.7 A Problem from Industry

We describe this problem in Appendix A.4. Here we explain implementation details and give numerical results.

The objective function is not defined everywhere in the search region. Therefore we use the multisection strategy A (see Section 2.3). We also cannot use accelerating devices like the monotonicity test, the nonconvexity test and the interval Newton method for boxes where the function is partly not defined. If the objective function is defined over the selected box, then the normal routine is applied. If not, the box will be subdivided until its diameter is less than ε_{bound}. And when the diameter is less than ε_{bound} the box will be inserted into the list L_b. In this implementation we therefore have the working list L, the solution list $\tilde{L}$ and the list L_b, where we store boxes for which

$$\text{neither} \quad \inf([x]_2) > \frac{454.3496}{2 \cdot \left(\frac{282.5}{[x]_1} + 3.182\right)^3} \quad \text{nor} \ \sup([x]_2) < \frac{454.3496}{2 \cdot \left(\frac{282.5}{[x]_1} + 3.182\right)^3}$$

holds, see Appendix A.4.

In the working list we store boxes ordered by the lower bound of the range of the function over that box. If the function is partly undefined over the box we cannot have the lower bound. In this case we store the diameter of the box with the minus sign as a lower bound. This way we first handle boxes for which the function is not defined.

We have run the serial and the parallel methods (modified for A.4) for this problem. In the case of the parallel method we run only on 4 processors.

The serial method for verified global optimization takes 3,466,981.00 STU and handles 2,069,315 boxes. So the parallel method on 4 processors takes 523,220.47 STU and handles 2,182,928 boxes. The parallel method handles 5 percent more boxes than the serial method. But the parallel method is 6.63 times faster than the serial method.

Appendix A

Considered Test Problems

In this work the following standard test problems for global optimization had been considered. Along with the function definition, the starting box $[x]$ and the parameter ε, that is used in the termination criteria, we give the enclosure of the global minimum and of the global minimizer. The considered test problems had been distributed into three classes according to the execution time of the serial methods: simple, medium and hard test problems. The simple test problems, the test problems with the execution time less than 10 STU, were not considered for parallelization, since the serial execution time is already small and it is meaningless to parallelize it further. But we use them in the process of designing serial methods.
All test problems achieve all their global minima at an interior point of the starting box $[x]$, so that a special edge handling for the serial and for the parallel methods was not implemented. All constant parameters appearing in the test problems were enclosed in intervals when implemented. In that way the problem of the non-representability of some decimal numbers on computers was circumvented.

A.1 Simple Problems

S5 ($n = 4$, Shekel 5)

$$f(x) = -\sum_{i=1}^{5} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

with $[x] = [0, 10]^4$, $\varepsilon = 10^{-6}$,

$$\mathrm{A} = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{pmatrix} \quad \text{and} \quad \mathrm{c} = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{pmatrix}.$$

The global minimum $f^* \in [-10.153199707210, -10.153199650879]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [4.000036851753, & 4.000037458427] \\ [4.000133096919, & 4.000133461207] \\ [4.000037057807, & 4.000037252354] \\ [4.000133225965, & 4.000133332058] \end{array}.$$

S7 ($n = 4$, Shekel 7)

$$f(x) = -\sum_{i=1}^{7} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

with $[x] = [0, 10]^4$, $\varepsilon = 10^{-6}$, A and c as in S5.
The global minimum $f^* \in [-10.402940854942, -10.402940278610]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [4.000572392022, & 4.000573448767] \\ [4.000689046287, & 4.000689693693] \\ [3.999489540210, & 3.999489885356] \\ [3.999606062342, & 3.999606263183] \end{array}.$$

S10 ($n = 4$, Shekel 10)

$$f(x) = -\sum_{i=1}^{10} \frac{1}{(x - A_i)(x - A_i)^T + c_i}$$

with $[x] = [0, 10]^4$, $\varepsilon = 10^{-6}$, A and c as in S5.
The global minimum $f^* \in [-10.536410152654, -10.536409480641]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [4.000745984918, & 4.000747087330] \\ [4.000592619822, & 4.000593257244] \\ [3.999663227190, & 3.999663575795] \\ [3.999509700323, & 3.999509908542] \end{array}.$$

SHCB ($n = 2$, Six-Hump-Camel-Back)

$$f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

with $[x] = [-2, 2]^2$, $\varepsilon = 10^{-6}$. The global minimum

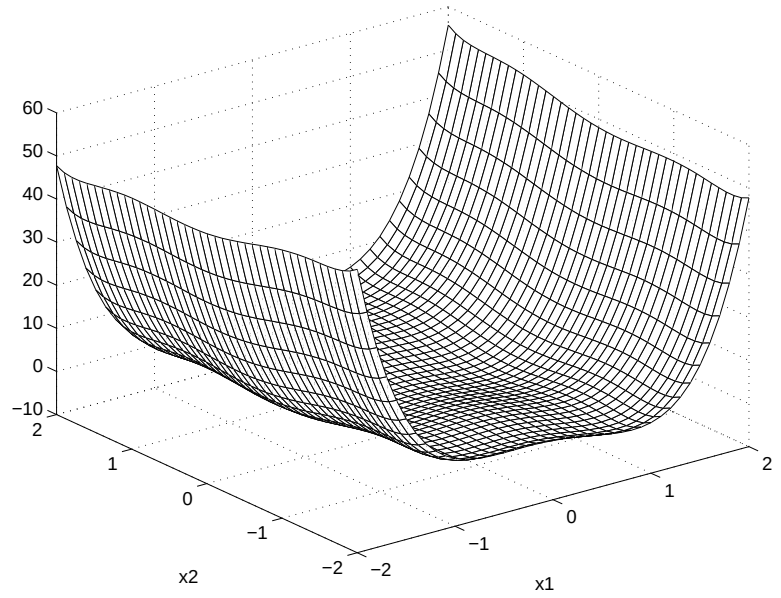


Figure A.1: The plot of the Six Hump Camel Back function

$f^* \in [-1.031628453614, -1.031628453366]$.
The unique verified local minimizers are enclosed in

$$\begin{bmatrix} \text{-0.089842013102}, & \text{-0.089842013098} \\ [\ 0.712656403010, & 0.712656403032] \end{bmatrix}, \begin{bmatrix} [\ 0.089842013098, & 0.089842013102] \\ \text{[-0.712656403032,} & \text{-0.712656403010]} \end{bmatrix}.$$

BR ($n = 2$, Branin)

$$f(x) = \left(\frac{5}{\pi}x_1 - \frac{5.1}{4\pi^2}x_1^2 + x_2 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

with $[x] = [-5, 10] \times [0, 15]$, $\varepsilon = 10^{-6}$. The global minimum
$f^* \in [0.397887357729, 0.397887361142]$.
The unique verified local minimizers are enclosed in

$$\begin{bmatrix} \text{-3.141718350524}, & \text{-3.141466904082} \\ \text{[12.274921258894,} & \text{12.275078374432]} \end{bmatrix}, \begin{bmatrix} [\ 3.141574972457, & 3.141610336919] \\ [\ 2.274998780786, & 2.275001215451] \\ [\ 9.424734796677, & 9.424821122815] \\ [\ 2.474998330906, & 2.475001666965] \end{bmatrix}.$$

RO ($n = 2$, Rosenbrock)

$$f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

with $[x] = [-5, 5]^2$, $\varepsilon = 10^{-6}$. The global minimum $f^* \in [0, 7.551320394136 \cdot 10^{-8}]$.
The unique verified global minimizer is enclosed in

$$\begin{bmatrix} [0.999988864642, & 1.000011135358] \\ [0.999994813743, & 1.000005186257] \end{bmatrix}.$$

L8 ($n = 3$, Levy 8)

$$f(x) = \sum_{i=1}^{2}(y_i - 1)^2(1 + 10\sin^2(\pi y_{i+1})) + \sin^2(\pi y_1) + (y_3 - 1)^2$$

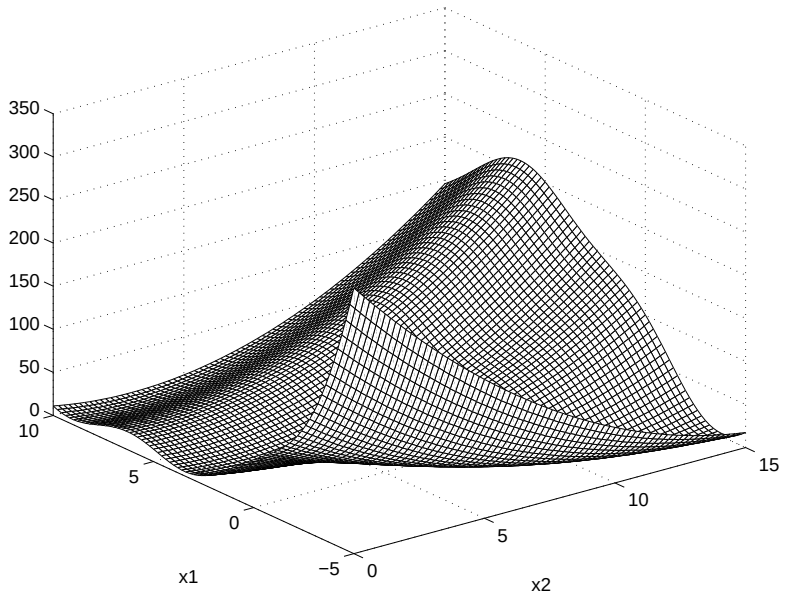


Figure A.2: The plot of Branin's function

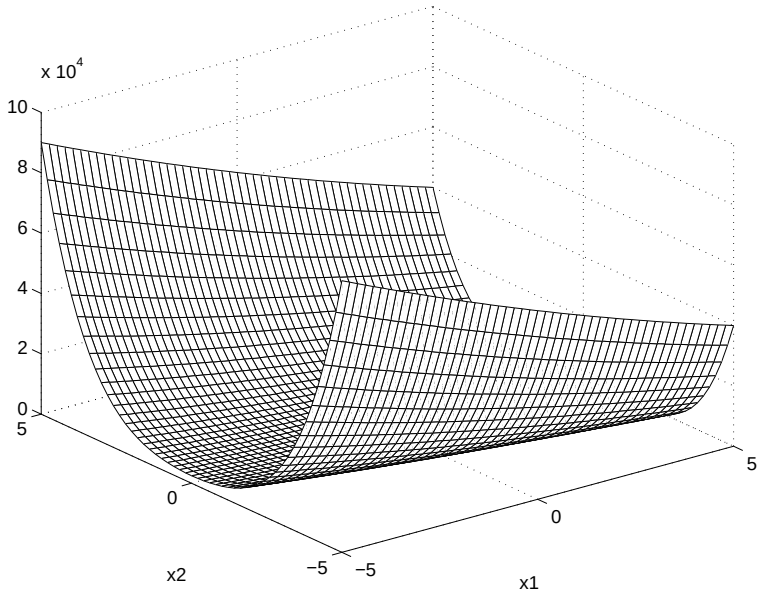


Figure A.3: The plot of Rosenbrock's function

with $y_i = 1 + (x_i - 1)/4$, $i = 1, \ldots, 3$, $[x] = [-10, 10]^3$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 2.751816814187 \cdot 10^{-9}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.999999921242, & 1.000000079089] \\ [0.999840137807, & 1.000184079674] \\ [0.999933892944, & 1.000100715801] \end{array} .$$

L9 ($n = 4$, Levy 9)

$$f(x) = \sum_{i=1}^{3} (y_i - 1)^2 (1 + 10 \sin^2(\pi y_{i+1})) + \sin^2(\pi y_1) + (y_4 - 1)^2$$

with $y_i = 1 + (x_i - 1)/4$, $i = 1, \ldots, 4$, $[x] = [-10, 10]^4$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 6.500293555635 \cdot 10^{-8}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.999999815271, & 1.000000184787] \\ [0.999159280921, & 1.000898023821] \\ [0.999664183571, & 1.000388113611] \\ [0.999790540536, & 1.000288039607] \end{array} .$$

H3 ($n = 3$, Hartman)

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} A_{ij} (x_j - P_{ij})^2 \right)$$

with $[x] = [0, 1]^3$, $\varepsilon = 10^{-6}$.

$$\mathrm{A} = \begin{pmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{pmatrix}, \ \mathrm{c} = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \text{ and } \mathrm{P} = \begin{pmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.03815 & 0.5743 & 0.8828 \end{pmatrix}.$$

The global minimum $f^* \in [-3.862782158846, -3.862782136795]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.114614313535, & 0.114614363644] \\ [0.555648849192, & 0.555648850752] \\ [0.852546953063, & 0.852546953979] \end{array} .$$

G5 ($n = 5$, Griewank 5)

$$f(x) = \sum_{i=1}^{5} \frac{x_i^2}{400} - \prod_{i=1}^{5} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$$

with $[x] = [-500, 600]^5$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [-0.000000000000, 3.162684336644 \cdot 10^{-9}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [-0.000047126184, & 0.000026346703] \\ [-0.000050237712, & 0.000025987328] \\ [-0.000049887998, & 0.000024425925] \\ [-0.000067131531, & 0.000029231509] \\ [-0.000063538412, & 0.000025042049] \end{array} .$$

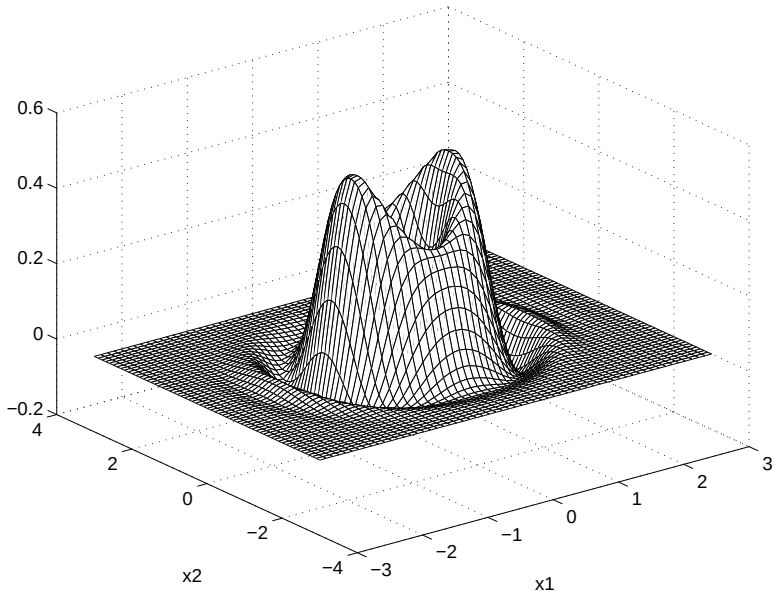


Figure A.4: The plot of Ratz's function

A.2 Medium Problems

R4 ($n = 2$, Ratz 4)

$$f(x) = \sin(x_1^2 + 2x_2^2)\exp(-x_1^2 - x_2^2)$$

with $[x] = [-3, 3]^2$, $\varepsilon = 10^{-6}$. The global minimum
$f^* \in [-0.106891344004, -0.106891338812]$.
The unique verified global minimizer is enclosed in

```
[-3.875919873641E-008],    3.875919873641E-008]
[-1.457522109420,         -1.457522101088 ]      ,
[-3.875919873641E-008],    3.875919873641E-008]
[ 1.457522101088,          1.457522109420 ]      .
```

L12 ($n = 10$, Levy 12)

$$f(x) = \sum_{i=1}^{9}(y_i - 1)^2(1 + 10\sin^2(\pi y_{i+1})) + \sin^2(\pi y_1) + (y_{10} - 1)^2$$

with $y_i = 1 + (x_i - 1)/4$, $i = 1, \ldots, 10$, $[x] = [-10, 10]^{10}$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 5.022707427890 \cdot 10^{-12}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll}
[0.999999999999, & 1.000000000001] \\
[0.999999999990, & 1.000000000010] \\
[0.999999999989, & 1.000000000011] \\
[0.999999999989, & 1.000000000012] \\
[0.999999996389, & 1.000000003645] \\
[0.999997930503, & 1.000002086249] \\
[0.999992487985, & 1.000007867647] \\
[0.999996881182, & 1.000003658406] \\
[0.999999326761, & 1.000000852685] \\
[0.999999997600, & 1.000000002395]
\end{array} .$$

L18 ($n = 7$, Levy 18)

$$\begin{aligned} f(x) = &\sum_{i=1}^{6} (x_i - 1)^2 (1 + \sin^2(3\pi x_{i+1})) \\ &+ (x_7 - 1)^2 (1 + \sin^2(2\pi x_7)) + \sin^2(3\pi x_1) \end{aligned}$$

with $[x] = [-5, 5]^7$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 5.415762071898 \cdot 10^{-12}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll}
[0.999999999999, & 1.000000000001] \\
[0.999999999988, & 1.000000000012] \\
[0.999999881074, & 1.000000117323] \\
[0.999998120221, & 1.000001807931] \\
[0.999998570159, & 1.000001318913] \\
[0.999999382590, & 1.000000509353] \\
[0.999999889173, & 1.000000097517]
\end{array} .$$

G7 ($n = 7$, Griewank 7)

$$f(x) = \sum_{i=1}^{7} \frac{x_i^2}{4000} - \prod_{i=1}^{7} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

with $[x] = [-500, 600]^7$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [-0.000000000000, 3.146680827016 \cdot 10^{-8}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll}
[-0.000152632287, & 0.000096304339] \\
[-0.000163024956, & 0.000096174777] \\
[-0.000162823505, & 0.000091817740] \\
[-0.000161711682, & 0.000105549551] \\
[-0.000152903602, & 0.000098344826] \\
[-0.000144957421, & 0.000091601764] \\
[-0.000137798478, & 0.000085371476]
\end{array} .$$

G10 ($n = 10$, Griewank 10)

$$f(x) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

with $[x] = [-100.5, 120]^{10}$, $\varepsilon = 10^{-6}$.

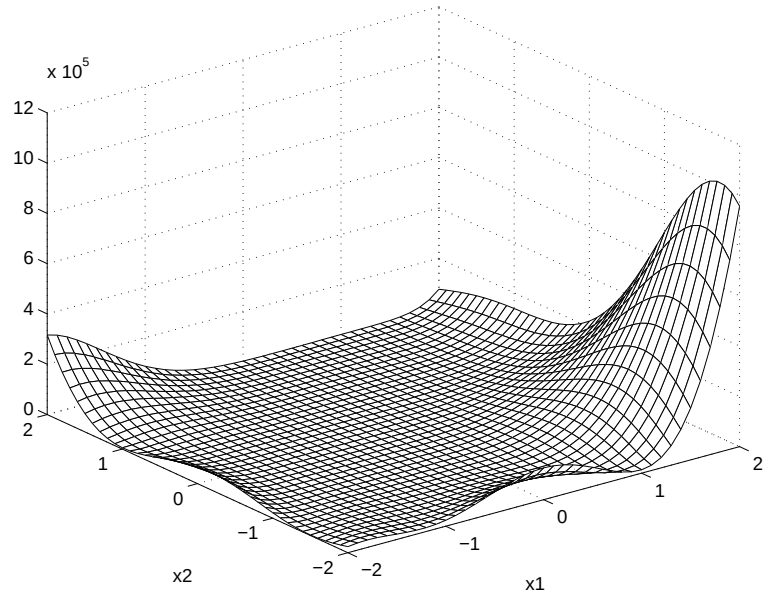


Figure A.5: The plot of Goldstein-Price's function

The global minimum $f^* \in [-0.000000000000, 1.708910790655 \cdot 10^{-9}]$.
The unique verified global minimizer is enclosed in

[-0.000042329872,	0.000041768222]
[-0.000035576283,	0.000035135435]
[-0.000040145513,	0.000040868663]
[-0.000029043363,	0.000029562000]
[-0.000021972735,	0.000022353973]
[-0.000017243896,	0.000017528873]
[-0.000013935200,	0.000014149906]
[-0.000011522315,	0.000011683990]
[-0.000009693677,	0.000009814265]
[-0.000008257013,	0.000008345096]

GP ($n = 2$, Goldstein-Price)

$$f(x) = (1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^3 - 14x_2 + 6x_1x_2 + 3x_2^2)) \times \\ (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_2^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$$

with $[x] = [-2, 2]^2$, $\varepsilon = 10^{-6}$. The global minimum $f^* \in [2.999999953835, 3.000000021153]$.
The unique verified global minimizer is enclosed in

[-7.092166092395E-011,	6.674678603178E-0-11]
[-1.000000000048,	-0.999999999962]

H6 ($n = 6$, Hartman 6)

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right)$$

with $[x] = [0,1]^6$, $\varepsilon = 10^{-6}$,

$$\mathrm{A} = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \qquad \mathrm{c} = \begin{pmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{pmatrix} \qquad \text{and}$$

$$P = \begin{pmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{pmatrix}.$$

The global minimum $f^* \in [-3.322368011452, -3.322368011379]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.201689511002, & 0.201689511012] \\ [0.150010691821, & 0.150010691826] \\ [0.476873974209, & 0.476873974235] \\ [0.275332430494, & 0.275332430495] \\ [0.311651616600, & 0.311651616601] \\ [0.657300534065, & 0.657300534066] \end{array}.$$

S2.14 ($n = 4$, Schwefel 2.14)

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

with $[x] = [-4,5]^4$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 4.475425668718 \cdot 10^{-8}]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [\text{-}0.009392075036, & 0.009317319146] \\ [\text{-}0.000907897950, & 0.000939203138] \\ [\text{-}0.002972763767, & 0.003141232908] \\ [\text{-}0.002899169922, & 0.003141401623] \\ [\text{-}0.004477072180, & \text{-}0.004272460937] \\ [\ 0.000396728515, & 0.000447702853] \\ [\ 0.002868652343, & 0.003184367790] \\ [\ 0.003143310546, & 0.003184542425] \end{array}, \quad \begin{array}{ll} [\ 0.002319335937, & 0.003033963163] \\ [\text{-}0.000289916993, & \text{-}0.000221252441] \\ [\text{-}0.003233245939, & \text{-}0.003173828125] \\ [\text{-}0.003233442004, & \text{-}0.003173828125] \\ [\text{-}0.004516072739, & \text{-}0.004272460937] \\ [\ 0.000396728515, & 0.000451600668] \\ [\ 0.003417968750, & 0.003612312397] \\ [\ 0.003417968750, & 0.003612576659] \end{array}.$$

GEO1 ($n = 3$, The Problem from the Geodesy)

$$f(x) = \left(\sqrt{x_2^2 + x_3^2 - 2c_1x_2x_3} - s_1\right)^2 + \left(\sqrt{x_3^2 + x_1^2 - 2c_2x_3x_1} - s_2\right)^2 + \left(\sqrt{x_1^2 + x_2^2 - 2c_3x_1x_2} - s_3\right)^2$$

with $[x] = [10^{-13}, 3600] \times [10^{-13}, 3520]^2$, $\varepsilon = 10^{-6}$.

$$\mathrm{c} = \begin{pmatrix} 0.846735205 \\ 0.928981803 \\ 0.912299033 \end{pmatrix} \qquad \text{and} \qquad \mathrm{s} = \begin{pmatrix} 1871.1 \\ 1592.4 \\ 1471.9 \end{pmatrix}.$$

The global minimum $f^* \in [0.000000000000, 4.064659879814 \cdot 10^{-8}]$.

The unique verified local minimizers are enclosed in

$$\begin{array}{ll} [2.292480245974\text{E}{+}003, & 2.292480532764\text{E}{+}003] \\ [3.225046974853\text{E}{+}003, & 3.225047033776\text{E}{+}003] \\ [3.477180122515\text{E}{+}003, & 3.477180155240\text{E}{+}003] \\ [3.575365805357\text{E}{+}003, & 3.575366350973\text{E}{+}003] \\ [3.412155596113\text{E}{+}003, & 3.412155809792\text{E}{+}003] \\ [2.435715337047\text{E}{+}003, & 2.435715899727\text{E}{+}003] \end{array} .$$

GEO2 ($n = 3$, The problem from the Geodesy)
The same as GEO1 with $[x] = [10^{-13}, 8.68] \times [10^{-13}, 9.24] \times [10^{-13}, 8.68]$, $\varepsilon = 10^{-6}$.

$$\mathrm{c} = \begin{pmatrix} 0.740824038 \\ 0.817119474 \\ 0.737253644 \end{pmatrix} \qquad \text{and} \qquad \mathrm{s} = \begin{pmatrix} 6.2 \\ 5.0 \\ 6.3 \end{pmatrix}.$$

The global minimum $f^* \in [0.000000000000, 1.623435961783 \cdot 10^{-10}]$.
The unique verified local minimizers are enclosed in

$$\begin{array}{ll} [8.369812220149, & 8.369839576033] \\ [8.947975268228, & 8.947982535122] \\ [8.150609828868, & 8.150627639759] \\ [8.284519260955, & 8.284618895430] \\ [8.999435873302, & 8.999467348600] \\ [5.288929330144, & 5.289080719743] \end{array} , \quad \begin{array}{ll} [4.873871645490, & 4.874651656288] \\ [8.964325384629, & 8.964372850539] \\ [8.118603200621, & 8.118671024373] \\ [8.311368014175, & 8.311422879085] \\ [3.271289310727, & 3.271387415143] \\ [8.221033235082, & 8.221046574887] \end{array} .$$

GEO3 ($n = 3$, The problem from the Geodesy)
The same as GEO1 with $[x] = [10^{-13}, 8.0]^3$, $\varepsilon = 10^{-6}$.

$$\mathrm{c} = \begin{pmatrix} 0.766044443 \\ 0.766044443 \\ 0.766044443 \end{pmatrix} \qquad \text{and} \qquad \mathrm{s} = \begin{pmatrix} 5.0 \\ 5.0 \\ 5.0 \end{pmatrix}.$$

The global minimum $f^* \in [0.000000000000, 1.435824800184 \cdot 10^{-9}]$.
The unique verified local minimizers are enclosed in

$$\begin{array}{ll} [7.309465389835, & 7.309556607226] \\ [7.309480041821, & 7.309541955399] \\ [7.309493137513, & 7.309528859554] \\ [3.889135969558, & 3.889481326344] \\ [7.309473015841, & 7.309548531135] \\ [7.309482573959, & 7.309538972089] \end{array} , \quad \begin{array}{ll} [7.309455164529, & 7.309566441765] \\ [3.889171586871, & 3.889445949503] \\ [7.309484162818, & 7.309537442613] \\ [7.309463050123, & 7.309558754954] \\ [7.309472945285, & 7.309548860249] \\ [3.889221015008, & 3.889397330209] \end{array} .$$

JS ($n = 2$, Jennrich-Sampson Problem)

$$f(x) = \sum_{i=1}^{10} \left(2 + 2i - \left(e^{ix_1} + e^{ix_2}\right)\right)^2$$

with $[x] = [-1, 1]^2$, $\varepsilon = 10^{-6}$. The global minimum
$f^* \in [124.362182355353, 124.362182355877]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.257825213670, & 0.257825213671] \\ [0.257825213670, & 0.257825213671] \end{array} .$$

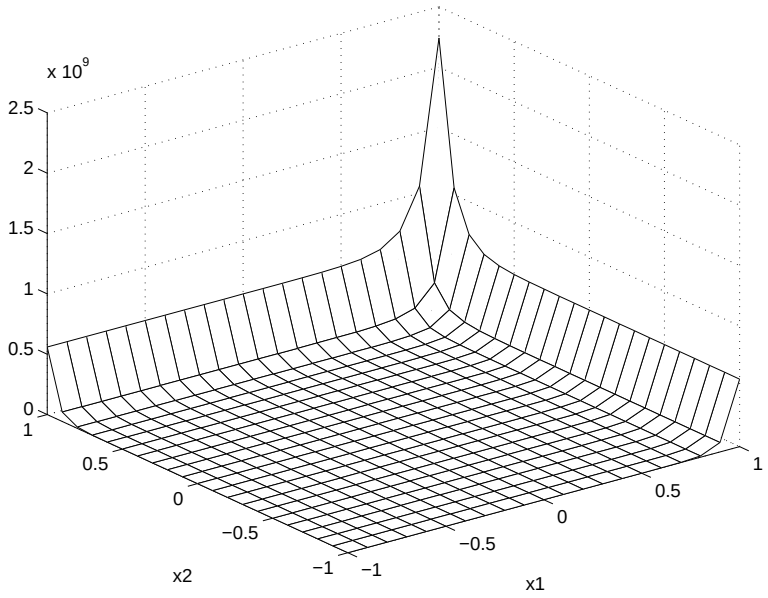


Figure A.6: The plot of Jennrich-Sampson function

A.3 Hard Problems

S2.7 ($n = 3$, Schwefel 2.7)

$$f(x) = \sum_{k=1}^{10} \left(\exp\left(\frac{-kx_1}{10}\right) - \exp\left(\frac{-kx_2}{10}\right) - \left(\exp\left(\frac{-k}{10}\right) - \exp(-k)\right) x_3 \right)^2$$

with $[x] = [0, 5] \times [8, 11] \times [0.5, 3]$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 0.000000132422]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.999760695233, & 1.000238272497] \\ [9.997724926185, & 10.002284236045] \\ [0.999951808024, & 1.000048843040] \end{array} .$$

L3 ($n = 2$, Levy 3)

$$f(x) = \sum_{i=1}^{5} i \cos((i+1)x_1 + i) \sum_{j=1}^{5} j \cos((j+1)x_2 + j)$$

with $[x] = [-10, 10]^2$, $\varepsilon = 10^{-6}$. The global minimum
$f^* \in [-1.867309091505 \cdot 10^2, -1.867309088310 \cdot 10^2]$.
The unique verified local minimizers are enclosed in

$$\begin{array}{llll} [\ 5.482864205738, & 5.482864207677] & [-0.800321100996, & -0.800321099948] \\ [\ 4.858056878362, & 4.858056879357] \ , & [\ 4.858056878605, & 4.858056879115] \ , \\ [-7.708313735502, & -7.708313735496] & [-1.425128428321, & -1.425128428318] \\ [-7.083506407653, & -7.083506407650] \ , & [-7.083506407653, & -7.083506407651] \ , \\ [-7.708313735500, & -7.708313735499] & [-7.708313735500, & -7.708313735499] \\ [\ 5.482864206707, & 5.482864206708] \ , & [-0.800321100472, & -0.800321100471] \ , \end{array}$$

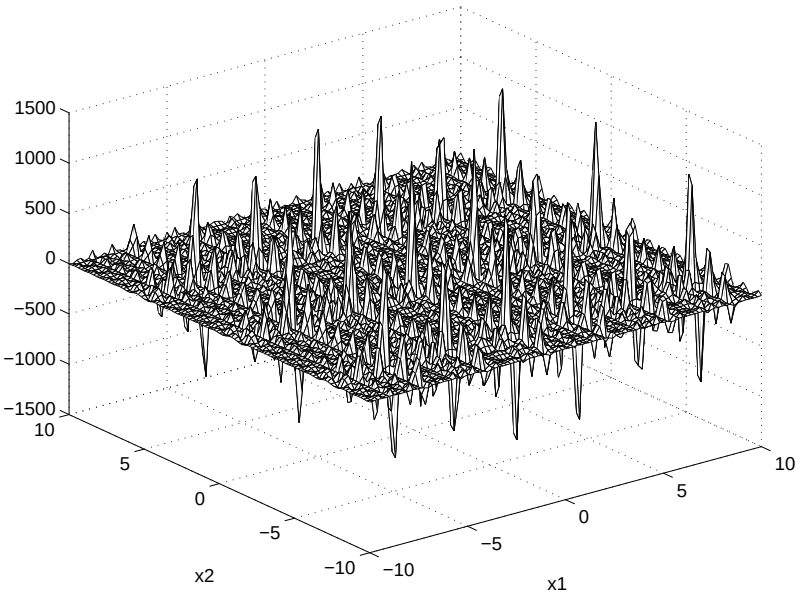


Figure A.7: The plot of Levy's function

[4.858056878859, 4.858056878860] [5.482864206707, 5.482864206708]
[-7.083506407652, -7.083506407651] , [-7.708313735500, -7.708313735499] ,
[-1.425128428320, -1.425128428319] [-1.425128428320, -1.425128428319]
[5.482864206707, 5.482864206708] , [-0.800321100472, -0.800321100471] ,
[-0.800321100472, -0.800321100471] [-7.083506407652, -7.083506407651]
[-7.708313735500, -7.708313735499] , [-7.708313735500, -7.708313735499] ,
[4.858056878859, 4.858056878860] [-7.083506407652, -7.083506407651]
[5.482864206707, 5.482864206708] , [4.858056878859, 4.858056878860] ,
[4.858056878859, 4.858056878860] [-7.083506407652, -7.083506407651]
[-0.800321100472, -0.800321100471] , [-1.425128428320, -1.425128428319] ,
[5.482864206707, 5.482864206708] [-0.800321100472, -0.800321100471]
[-1.425128428320, -1.425128428319] , [-1.425128428320, -1.425128428319] .

R8 ($n = 9$, Ratz 8)

$$f(x) = \left(\sin^2 \left(\pi \frac{x_1 + 3}{4} \right) + \sum_{i=1}^{8} \left(\frac{x_i - 1}{4} \right)^2 \left(1 + 10 \sin^2 \left(\pi \frac{x_{i+1} + 3}{4} \right) \right) \right)^2$$

with $[x] = [-10, 10]^9$, $\varepsilon = 10^{-6}$. The global minimum $f^* \in [0.000000000000, 0.000000455511]$. The unique verified local minimizers are enclosed in

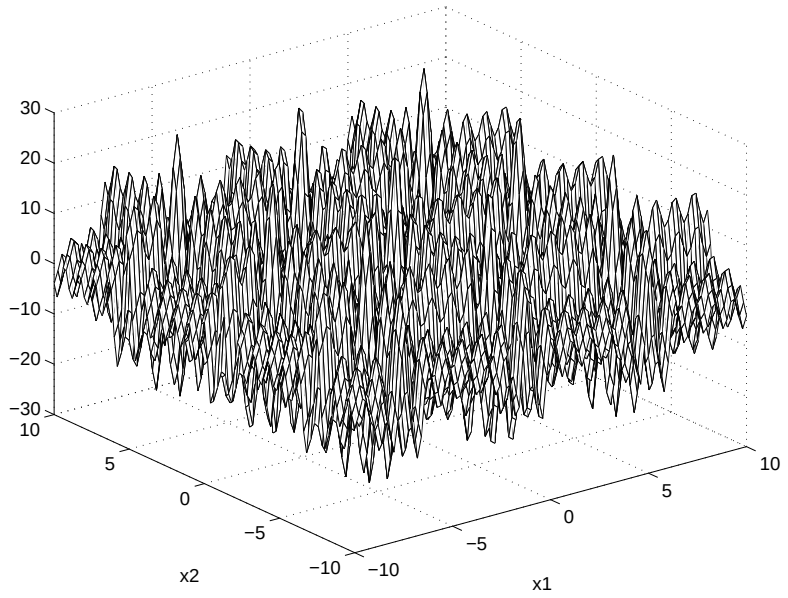


Figure A.8: The plot of Henriksen-Madsen's function

$$\begin{array}{ll}
[\ 0.996093750000, & 1.015625000000] \\
[\ 0.937500000000, & 1.015625000000] \\
[\ 0.976562500000, & 1.015625000000] \\
[\ 0.937500000000, & 1.015625000000] \\
[\ 0.937500000000, & 1.015625000000] \\
[\ 0.976562500000, & 1.015625000000] \\
[\ 0.937500000000, & 1.015625000000] \\
[\ 0.976562500000, & 1.015625000000] \\
[-10.000000000000, & 10.000000000000]
\end{array}.$$

HM3 ($n = 2$, Henriksen and Madsen)

$$f(x) = -\sum_{i=1}^{2}\sum_{j=1}^{5} j\sin((j+1)x_i + j)$$

with $[x] = [-10, 10]^2$, $\varepsilon = 10^{-6}$. The global minimum
$f^* \in [-24.062498884345, -24.062498884330]$.
The unique verified local minimizers are enclosed in

$$\begin{array}{llll}
[-6.774576143440, & -6.774576143438] & [\ 5.791794470920, & 5.791794470921] \\
[-6.774576143440, & -6.774576143438] \ , & [-6.774576143440, & -6.774576143438] \ , \\
[-6.774576143440, & -6.774576143438] & [-0.491390836260, & -0.491390836259] \\
[\ 5.791794470920, & 5.791794470921] \ , & [-6.774576143440, & -6.774576143438] \ , \\
[-6.774576143440, & -6.774576143438] & [\ 5.791794470920, & 5.791794470921] \\
[-0.491390836260, & -0.491390836259] \ , & [\ 5.791794470920, & 5.791794470921] \ , \\
[\ 5.791794470920, & 5.791794470921] & [-0.491390836260, & -0.491390836259] \\
[-0.491390836260, & -0.491390836259] \ , & [\ 5.791794470920, & 5.791794470921] \ , \\
\multicolumn{4}{c}{[-0.491390836260, \quad -0.491390836259]} \\
\multicolumn{4}{c}{[-0.491390836260, \quad -0.491390836259]} \ .
\end{array}$$

HM4 ($n = 3$, Henriksen and Madsen)

$$f(x) = -\sum_{i=1}^{2}\sum_{j=1}^{5} j\sin((j+1)x_i + j)$$

with $[x] = [-5, 5]^3$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [-36.093748326755, -36.093748326248]$.
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [\text{-}0.491390836264, & \text{-}0.491390836254] \\ [\text{-}0.491390836260, & \text{-}0.491390836259] \\ [\text{-}0.491390836264, & \text{-}0.491390836254] \end{array}.$$

KOW ($n = 4$, Kowalik Problem)

$$f(x) = \sum_{i=1}^{11}\left(a_i - x_1\frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4}\right)^2$$

with $[x] = [0, 0.42]^4$, $\varepsilon = 10^{-6}$,

$$\mathrm{c} = \begin{pmatrix} 0.1957 \\ 0.1947 \\ 0.1735 \\ 0.1600 \\ 0.0844 \\ 0.0627 \\ 0.0456 \\ 0.0342 \\ 0.0323 \\ 0.0235 \\ 0.0246 \end{pmatrix} \quad \text{and} \quad \mathrm{s} = \begin{pmatrix} 4 \\ 2 \\ 1 \\ 0.5 \\ 0.25 \\ 1/6 \\ 0.125 \\ 0.1 \\ 1/12 \\ 1/14 \\ 0.0625 \end{pmatrix}.$$

The global minimum $f^* \in [0.000307140870, 0.000307616995]$
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [0.192832591586, & 0.192834390270] \\ [0.190819898956, & 0.190850903478] \\ [0.123112693715, & 0.123121520922] \\ [0.135759694776, & 0.135771333126] \end{array}.$$

WK ($n = 1$, Krämer Problem)

$$f(x) = -\frac{p(x)}{q(x)} = -\frac{\sum_{i=0}^{29} p_i x^i}{\sum_{i=0}^{4} q_i x^i}$$

with $[x] = [0, 64]$, $\varepsilon = 10^{-6}$,

$$q = \begin{pmatrix} 0.588286746328683429346629 9376 \cdot 10^{11} \\ 0.363467493465600874106423 7087 \cdot 10^{9} \\ 0.996353603100060267502727 7824 \cdot 10^{6} \\ 0.146434177625559953978943 5142 \cdot 10^{4} \\ 1 \end{pmatrix}$$

and

$$p = \begin{pmatrix} 7.629394531250000 \cdot 10^{-6} \\ -1.150369644165040 \cdot 10^{-5} \\ 1.372280530631542 \cdot 10^{-5} \\ -6.579421551577981 \cdot 10^{-6} \\ 1.659054419178573 \cdot 10^{-6} \\ -2.521266665667100 \cdot 10^{-7} \\ 2.505680664436719 \cdot 10^{-8} \\ -1.713721655060476 \cdot 10^{-9} \\ 8.330923088212047 \cdot 10^{-11} \\ -2.935023067946825 \cdot 10^{-12} \\ 7.564193999729689 \cdot 10^{-14} \\ -1.426868803614099 \cdot 10^{-15} \\ 1.954186293985405 \cdot 10^{-17} \\ -1.910400220016202 \cdot 10^{-19} \\ 1.299884226135079 \cdot 10{-}21 \\ -5.995712492310049 \cdot 10^{-24} \\ 1.876066147446556 \cdot 10^{-26} \\ -4.291373306373139 \cdot 10^{-29} \\ 7.622481227988642 \cdot 10^{-32} \\ -1.096397325341554 \cdot 10^{-34} \\ 1.315291857866774 \cdot 10^{-37} \\ -1.344664974747858 \cdot 10^{-40} \\ 1.190815536452828 \cdot 10^{-43} \\ -9.253132527171894 \cdot 10^{-47} \\ 6.374582890249432 \cdot 10^{-50} \\ -3.926998956415952 \cdot 10^{-53} \\ 2.170989219023664 \cdot 10^{-56} \\ -1.142719267106732 \cdot 10^{-59} \\ 3.823185031874960 \cdot 10^{-63} \\ -3.691550884472599 \cdot 10^{-66} \end{pmatrix}.$$

The global minimum $f^* \in [-5.129659043375E-016, -4.541716718401E-016]$
The unique verified global minimizer is enclosed in

$$\begin{array}{ll} [34.566830008723, & 34.566830519889] \\ [34.566830635070, & 34.566830764922] \end{array}$$

INF1 $(n = 2)$

$$f(x) = (x_1 - x_2)^2$$

with $[x] = [-2.0, 2.5]^2$, $\varepsilon = 10^{-6}$.
The global minimum $f^* \in [0.000000000000, 1.885928213597E-008]$.
The local minimizers are

$$[x]^* = [x] \cap \{(x_1, x_2) \ : \ x_1 = x_2\}.$$

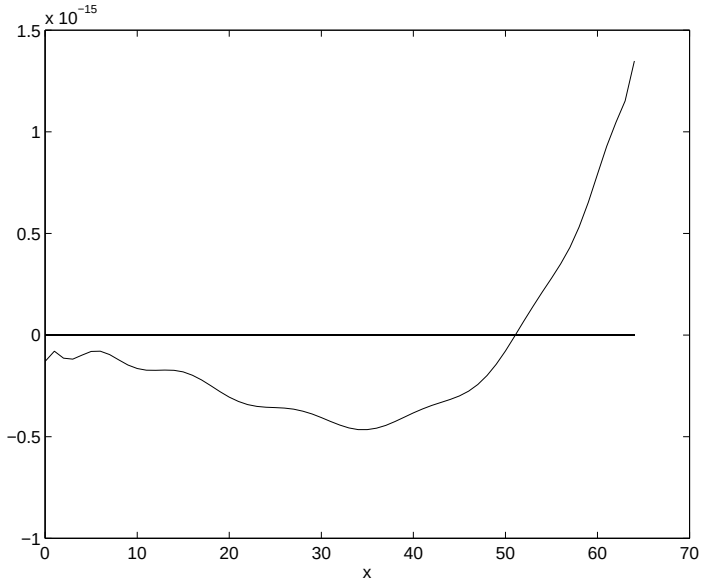


Figure A.9: The plot of Krämer's function

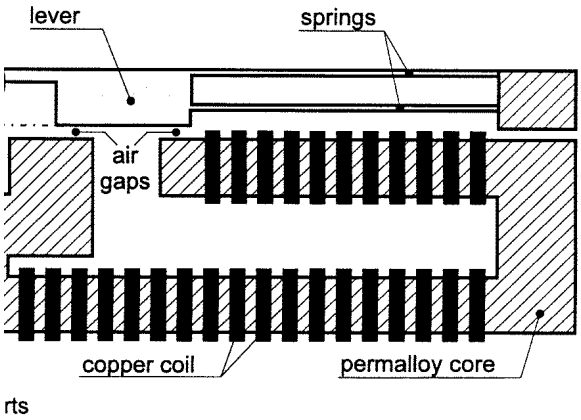


Figure A.10: Schematic of the microrelay

A.4 A Problem from Industry

This problem comes from the design of a microrelay. Our description given here follows the unpublished article of Prof. Dr.-Ing. B. Tibken, the University of Wuppertal. The microrelay has been developed at the Forschungszentrum Karlsruhe in the Institue of Microstructure Technology (see Figure A.10). The actual size of this microdevice is approximately 2mm $\times$ 3mm and the distance between the contacts is only 30μm. Mathematical modeling of the dynamics of this device results in the following differential equation of second order

$$z''(t) + 2 \cdot d \cdot \omega_0 \cdot z'(t) + \omega_0^2 \cdot z(t) + b(z(t)) \cdot I^2(t) = 0 \tag{A.1}$$

Parameter	Physical Interpretation	Value
d	damping constant	0.012
$f_0 = \frac{\omega_0}{2 \cdot \pi}$	base frequency	829.7Hz
A_G	cross-sectional area of air gap	$20000\mu\text{m}^2$
μ_0	magnetic field constant	$1.2566 \cdot 10^{-6}$
N	number of coil windings	44
L_{Fe}	effective length of magnetic core	11.3mm
A_{Fe}	cross-sectional area of magnetic core	$40000\mu\text{m}^s$
δ_0	width of air gap in steady state position	30μm

where $z(t)$ describes the lever deflection from the rest position. The parameters d and $f_0 = \frac{\omega_0}{2 \cdot \pi}$ are the damping constant and the base frequency, respectively, and $I(t)$ is the input current through the copper coil. The nonlinear function $b(z)$ is given by

$$b(z) = \frac{\omega_0^2 \cdot A_G}{c_g \cdot \mu_0} \cdot \left(\frac{N \cdot \mu_0 \cdot \mu_r}{\frac{L_{Fe} \cdot A_G}{A_{Fe}} + 2 \cdot \mu_r \cdot (\delta_0 + z)} \right)^2 \tag{A.2}$$

and is derived from the magnetic interaction. In (A.1) and (A.2) all parameters except μ_r and c_g are well known and have been taken from published data or have been directly measured using a microscope. The corresponding values are given in Table A.4. For the estimation of μ_r and c_g the experimental measurement data are the measured stationary deflections of the lever for different constant currents I. The experimental measurements have been performed for 23 different current values $I_1, \ldots, I_{23}$. From the model equation (A.2) it follows that the stationary deflections satisfy the following cubic equation

$$z^3 + 2 \cdot \gamma \cdot z^2 + \gamma^2 \cdot z + \nu = 0 \tag{A.3}$$

where the abbreviations

$$\gamma(\mu_r) = \frac{L_{Fe} \cdot A_G}{2 \cdot \mu_r \cdot A_{Fe}} + \delta_0,$$
$$\nu(c_g, I) = \frac{N^2 \cdot A_G \cdot \mu_0 \cdot I^2}{4 \cdot c_g}$$

have been introduced. Using the procedure given in [7] to compute the solutions of (A.3) results in three real solutions for the given parameter range. The relevant solution describing the stationary deflection for constant current is given by

$$z(\mu_r, c_g, I(t)) = \frac{2\gamma(\mu_r)}{3} \cdot \left(\cos\left(\frac{\arccos\left(1 - \frac{27\nu(c_g, I(t))}{2\gamma(\mu_r)^3}\right)}{3} \right) - 1 \right)$$

where the dependence from the parameters to be estimated has been made explicit. In this equation, the value of the function z can be measured for the input current I. The measurement equation is given by

$$y = h(\mu_r, c_g),$$

where $h(\mu_r, c_g) = (h_1(\mu_r, c_g), \ldots h_{23}(\mu_r, c_g))^T$ with

$$h_i(\mu_r, c_g) = \frac{2\gamma(\mu_r)}{3} \cdot \left(\cos\left(\frac{\arccos\left(1 - \frac{27\nu(c_g, I_i)}{2\gamma(\mu_r)^3)}\right)}{3} \right) - 1 \right), \quad i = 1, \ldots, 23.$$

The experimentally measured values are

$$
\begin{aligned}
I_1 &= 10.0 \cdot 10^{-3}, & y_1 &= 31.6 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_2 &= 15.0 \cdot 10^{-3}, & y_2 &= 31.38 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_3 &= 20.0 \cdot 10^{-3}, & y_3 &= 30.94 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_4 &= 25.0 \cdot 10^{-3}, & y_4 &= 30.5 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_5 &= 30.0 \cdot 10^{-3}, & y_5 &= 29.84 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_6 &= 32.5 \cdot 10^{-3}, & y_6 &= 29.61 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_7 &= 35.0 \cdot 10^{-3}, & y_7 &= 29.17 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_8 &= 37.5 \cdot 10^{-3}, & y_8 &= 28.73 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_9 &= 40.0 \cdot 10^{-3}, & y_9 &= 28.29 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{10} &= 42.5 \cdot 10^{-3}, & y_{10} &= 27.63 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{11} &= 45.0 \cdot 10^{-3}, & y_{11} &= 27.19 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{12} &= 47.5 \cdot 10^{-3}, & y_{12} &= 26.52 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{13} &= 50.0 \cdot 10^{-3}, & y_{13} &= 24.97 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{14} &= 50.5 \cdot 10^{-3}, & y_{14} &= 24.75 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{15} &= 51.0 \cdot 10^{-3}, & y_{15} &= 24.75 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{16} &= 51.5 \cdot 10^{-3}, & y_{16} &= 24.31 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{17} &= 52.0 \cdot 10^{-3}, & y_{17} &= 24.31 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{18} &= 52.1 \cdot 10^{-3}, & y_{18} &= 23.65 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{19} &= 52.2 \cdot 10^{-3}, & y_{19} &= 23.65 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{20} &= 52.3 \cdot 10^{-3}, & y_{20} &= 23.65 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{21} &= 52.4 \cdot 10^{-3}, & y_{21} &= 23.65 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{22} &= 52.5 \cdot 10^{-3}, & y_{22} &= 23.43 \cdot 10^{-6} - 31.82 \cdot 10^{-6},\\
I_{23} &= 52.6 \cdot 10^{-3}, & y_{23} &= 23.43 \cdot 10^{-6} - 31.82 \cdot 10^{-6}.
\end{aligned}
$$

The estimates for the parameters μ_r and c_g are calculated with the help of the optimization problem

$$\min_{\mu_r, c_g} \|y - h(\mu_r, c_g)\|^2 = \min_{\mu_r, c_g} \sum_{i=1}^{23} \left(y_i - h_i(\mu_r, c_g)\right)^2$$

which has to be solved in order to compute the L_2 optimal solution to the parameter estimation problem. The search regions for μ_r and c_g are $[500, 2500]$ and $[5.2, 6.7]$, respectively. One difficulty of this problem is that the function $\arccos\left(1 - \frac{27\nu(c_g, I_i)}{2\gamma(\mu_r)^3)}\right)$ is not defined everywhere over the search region. It is defined only for

$$c_g > \frac{454.3496}{2 \cdot \left(\frac{282.5}{\mu_r} + 3.182\right)^3}. \tag{A.4}$$

We run the method for verified global optimization (modified for the condition (A.4), see 4.3.7) for the objective function $f = \sum_{i=1}^{23} \left(y_i - h_i(x_1, x_2)\right)^2$ with $[x] = [500, 2500] \times [5.2, 6.7]$. The accuracy is set to $\varepsilon = 10^{-14}$. In Figure A.11 we plotted the solution of this problem.

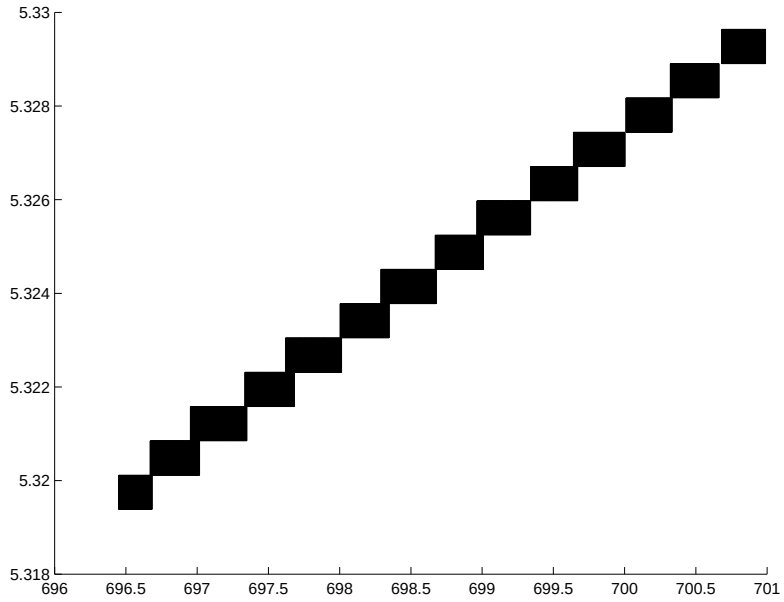


Figure A.11: The plot of the solution for the minimization problem

Literature

[1] Alefeld, G., On the convergence of some interval-arithmetic modifications of Newton's method, In: Stepleman R. et al. (ed) Scientific Computing, IMACS/North-Holland Publishing Company, 223-230, 1983.

[2] Alefeld, G., Über die Divergenzgeschwindigkeit des Intervall-Newton-Verfahrens, Fasciuli Mathematici 19, 9-13, 1989.

[3] Alefeld, G., Über das Divergenzverhalten des Intervall-Newton-Verfahrens, Computing 46, 289-294, 1991.

[4] Alefeld, G., Herzberger, J., Introduction to interval computations, Academic Press, 1983.

[5] Berner, S., Ein paralleles Verfahren zur verifizierten globalen Oprimierung, Dissertation, Universität Wuppertal, Shaker Verlag, Aachen, 1995.

[6] Berner, S., Parallel Methods for Verified Global Optimization: Practice and Theory, Journal of Global Optimization 9, 1-22, 1996.

[7] Bronstein, I. N., Semendjajew, K. A., Taschenbuch der Mathematik, 23, B. G. Teubner, Leipzig, 1987.

[8] Caprani, O., Godthaab, B., Madsen, K., Use of a real-value local minimum in parallel interval global optimization, Interval Computations, 2, 71-82, 1993.

[9] Csallner, A. E., Csendes, T., Markot, M. C., Multisection in Interval Branch-and-Bound Method for Global Optimization, Journal of Global Optimization 1-32, Kluwer Academic Press, 1999.

[10] Csendes, T., Ratz, D., Subdivision direction selection in interval methods for global optimization, SIAM Journal of Numerical Analysis.

[11] Dixon, L. C. W., Szegö, G. P. (ed), Towards Global Optimization, North-Holland Amsterdam, 1975.

[12] Dixon, L. C. W., Szegö, G. P. (ed), Towards Global Optimization 2, North-Holland, Amsterdam, 1978.

[13] Hammer, R., Hocks, M., Kulisch, U., Ratz, D., C++ Toolbox for Verified Computing: Basic Numerical Problems, Springer-Verlag, New York, 1995.

[14] Dixon, L. C. W., Jha, M., Parallel algorithms for global optimization, Journal of Optimization Theory and Applications, 79, 385-395, 1993.

[15] Eriksson, J., Parallel Global Optimization Using Interval Analysis, Dissertaion, University of Umea, Sweden, 1991.

[16] Flynn, M. J., Some computer organizations and their effectiveness, IEEE Trans. Comput., c-21, 1972.

[17] Hammer, R., Hocks, M., Kulisch, U., Ratz, D., Numerical Toolbox for Verified Computing I, Springer-Verlag, Berlin, Heidelberg, 1993.

[18] Hansen, E., Global optimization using interval analysis: The one-dimensional case, Journal of Optimization Theory and Applications, 29, 331-334, 1979.

[19] Hansen, E. R., Global optimization using interval analysis - the multidimensional case, Numerische Mathematik, 34, 247-270, 1980.

[20] Hansen, E. R., Global optimization using interval analysis, Marcel Dekker, Inc, 1992.

[21] Henriksen, T., Madsen, K., Use of a depth-first strategy in parallel global optimization, Tech, Report 92-10, Institute for Numerical Analysis, Technical University of Denmark, Lyngby, 1992.

[22] Herzberger, J., Topics in validated computations, Proceeding of the IMACS-GAMM International Workshop on Validated Computation, Oldenburg, Germany, 30 August - 3 September 1993, Elsevier Science B. V., 1994.

[23] Hord, R. M., Parallel Supercomputing in MIMD Architectures, CRC Press, 1993.

[24] Horst, R., Tuy, H., Global Optimization, Deterministic Approaches, Springer-Verlag, Berlin, Heidelberg, 2, Aufl., 1993.

[25] Ichida, K., Fujii, Y., An interval arithmetic method for global optimization, Computing, 23, 85,97, 1979.

[26] Jájá, J., An Introduction to parallel algorithms, Addison-Wesley Publishing Company, Inc (1992).

[27] Jansson, C., A global minimization method: The one-dimensional case, Tech. Report, Technische Universität Hamburg-Harburg, 1991.

[28] Jansson, C., On self-validated methods for optimization problems, in Herzberger, J. (ed), Topics in 1 Computations, Elsevier, 1994.

[29] Jansson, C., Knüppel, O., A global minimization method: The multidimensional case, Tech. Report, Technische Universität Hamburg-Harburg, 1991.

[30] Krawczyk, R., Nickel, K., Die zentrische Form in der Intervallarithmetik, ihre quadratische Konvergenz und ihre Inklusionsisotonie, Computing, 28, 117-132, 1982.

[31] Kearfott, R. B., Novoa III, M., INTBIS, a portable interval Newton/bisection package, ACM Trans, Math. Software, 16, 152-157, 1990.

[32] Kearfott, R. B., Rigorous Global Search: Continuous Problems, Kluwer Academic Publishers Dordrecht, 1996.

[33] Klatte, R., Kulisch, U., Wiethoff, A., Lawo, C., Rauch, M., C-XSC: A C++ Class Library for Extended Scientific Computing. Springer Verlag.

[34] Knüppel, O., Einschließungsmethoden zur Bestimmung der Nullstellen nichtlinearer Gleichungssysteme und ihre Implementierung, Dissertation, Technische Universität Hamburg-Harburg, 1995.

[35] Lewis, T. G., Introduction to parallel computing, Prentice-Hall, Inc, 1992 .

[36] Message Passing Interface Forum, MPI:A Message-Passing Interface Standard. 1994.

[37] Moore, R. E., Interval Analisys, Prentice-Hall, New Jersey, 1966.

[38] Moore, R. E., Intervallanalyse, Oldenbourg-Verlag, München, 1969.

[39] Moore, R. E., Methods and Applications of Interval Analisys, SIAM, Philadelphia, 1979.

[40] Moore, R. E., Hansen, E., Leclerc, A., Rigorous methods for global optimization, in Floudas, C.A., Pardalos, P.M.(ed), Recent Advances in Global Optimization, Princeton University Press, 1992.

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

[42] Ortega, J. M., Matrix Theory, New York and London, Plenum Press, 1987.

[43] Rall, L. B., Lecture Notes in Computer Science, Springer-Verlag, 1981.

[44] Ratschek, H., Rokne, J., Computer methods for the range of functions, Ellis Horwood, Chichester, 1984.

[45] Ratschek, H., Rokne, J., New Computer Methods for Global Optimization, Ellis Horwood, Chichester, 1988.

[46] Ratschek, H., Rokne, J., Interval methods, in Horst, R.K. Pardalos, P.M.(ed), Handbook of Global Optimization, Kluwer Academic Publishers, Dordrecht, 751-828, 1995.

[47] Ratz, D., Automatische Ergebnisverifikation bei globalen Optimierungsproblemen, Dissertation, Universität Karlsruhe (TH), 1992.

[48] Ratz, D., Csendes, T., On the selection of subdivision directions in interval branch-and-bound methods for global optimization, Journal of Global Optimization.

[49] Rinnoy Kan, A. H. G., Timmer, G. T., Stochastic global optimization methods, Part I: Clustering methods, Mathematical Programming, 39, 27-56, 1987.

[50] Rinnoy Kan, A. H. G., Timmer, G. T.,Stochastic global optimization methods, Part II: Multi level methods, Mathematical Programming, 39, 57-78, 1987.

[51] Skelboe, S., Computation of rational interval functions, BIT, 14, 87-95, 1974.

[52] Schmidt, J. W., Die Regula Falsi für Operatoren in Banachräumen, ZAMM 41, 61-63, 1961.

[53] Schwandt, H., Schnelle fast global konvergente Verfahren für die Fünf-Punkt-Diskretisierung der Poissongleichung mit Dirichletschen Randbedingungen auf Rechteckgebieten, Dissertation, Fachbereich Mathematik der TU Berlin, 1981.

[54] Törn, A., Zilinskas, A., Global Optimization, Springer-Verlag, Berlin, Heidelberg, 1989.

[55] Wiethoff, A., Verifizierte globale Optimierung auf Parallelrechnern, Dissertation, Universität Karlsruhe, Karlsruhe, 1998.

[56] Yair, C., Stavros, A. Z., Parallel Optimization: Theory, Algorithms, and Applications, Oxford University Press, 1997.

[57] Zomaya, A. Y., Parallel and Distributed Computing Handbook, McGraw-Hill, Inc, 1996.