STUDIES IN COMPUTATIONAL MATHEMATICS 5

editors: C. BREZINSKI and L. WUYTACK

TOPICS IN VALIDATED COMPUTATIONS

J. HERZBERGER
Editor

NORTH-HOLLAND

TOPICS IN VALIDATED COMPUTATIONS

Proceedings of the IMACS-GAMM International Workshop on Validated Computation, Oldenburg, Germany, 30 August-3 September 1993

edited by

Jürgen HERZBERGER
Fachbereich Mathematik
Universität Oldenburg
Germany

1994

ELSEVIER
Amsterdam – Lausanne – New York – Oxford – Shannon – Tokyo

ELSEVIER SCIENCE B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

Library of Congress Cataloging-in-Publication Data

IMACS-GAMM International Workshop on Validated Computations (1993 : Oldenburg, Germany)
Topics in validated computations : proceedings of IMACS-GAMM International Workshop on Validated Computations, Oldenburg, Germany, 30 August - 3 Septmeber, 1993 / edited by Jürgen Herzberger.
p. cm. -- (Studies in computational mathematics ; 5)
Includes bibliographical references.
ISBN 0-444-81685-2
1. Interval analysis (Mathematics)--Congresses. 2. Numerical calculations--Verification--Congresses. 3. Algorithms--Congresses. I. Herzberger, Jürgen. II. Title. III. Series.
QA297.75.I43 1993
512'.5--dc20 94-36669
CIP

ISBN: 0 444 81685 2

This book is printed on acid-free paper.

Printed in the Netherlands

Topics in Validated Computations
J. Herzberger (Editor)
1994 Elsevier Science B.V.

On Self-Validating Methods for Optimization Problems

C. Jansson
Technische Informatik III, Technische Universität Hamburg-Harburg, Sitz: Eißendorfer Straße 38, 21071 Hamburg, Germany

0. Introduction

Optimization deals with the problem of minimizing or maximizing a function of several variables subject to inequality and/or equality constraints. Optimization problems are central in many applications in applied mathematics, operations research, management science and engineering. This wide applicability, the rich mathematical theory underlying many models in optimization, and the methods to solve these models have been driving forces behind the rapid and continuing evolution of this subject.

Usually, the methods and algorithms devoloped in optimization theory are performed on digital computers. When solving a problem on a computer, three major sources of errors are present. First, in many practical applications the input data are uncertain, that is experimental, measurement, and modelling errors may be present. Secondly, during the execution of an algorithm rounding errors and cancellation occur. Finally, even if some input data are known exactly, conversion errors may occur.

Therefore, a central question is: How reliable are the results calculated on a computer? There are two well-known approaches. The first approach is the backward error analysis, a technique in which errors made in executing an algorithm are thrown back on the original input data. Backward error analysis and their widespread use is closely connected with Wilkinson's work [86], [87]. The second approach is forward error analysis, that is to give explicit bounds for the error of the computed solution, where all sources of errors are taken into consideration. An error analysis may be very important; there are simple examples in optimization (cf. Section 2) where well-known algorithms produce approximations of an optimal solution which are completely wrong in the sense of backward and forward error analysis.

Interval arithmetic provides (i) a tool for forward error analysis for many problems, (ii) for estimating and controlling these errors automatically and rigorously, and (iii) for proving existence and uniqueness of solutions. In this context, the term *self-validating numerical methods, inclusion methods, or verification methods* is in use. Instead of approximating a real value or an uncertain real value by a machine number, in interval arithmetic this value is replaced by a machine interval. The disadvantage of a straightforward application of

interval arithmetic is that due to data dependencies dramatic overestimations may occur. Therefore, algorithms using interval arithmetic must be designed very carefully and should avoid this phenomenon. A number of books describe the tools of interval arithmetic, and are recommended to readers who desire additional information on this topic. These include Alefeld and Herzberger 1974 [3] and 1983 [4], Bauch et al 1987 [6], Moore 1979 [53], and Neumaier 1990 [59]. A summarizing representation of a new computer arithmetic taking into consideration interval arithmetic together with optimal scalar products is given in Kulisch and Miranker 1981 [44].

Many present-day methods that use the tools of interval arithmetic are not aiming to replace conventional numerical methods. On the contrary, they incorporate conventional numerical methods in an extensive way, and they are very much in the spirit of Wilkinson 1971 [88]:

> *"In general it is the best in algebraic computations to leave the use of interval arithmetic as late as possible so that it effectively becomes an a posteriori weapon."*

The major purpose of this paper is to present such self-validating methods for linear programming problems and global optimization problems.

In Section 1 self-validating methods for linear programming problems are presented. Linear programming problems involve the optimization of a linear function subject to linear constraints. There are numerous excellent treatments of this subject. We mention here only the fundamental work of Dantzig 1963 [17]. Linear programming problems have the main properties that (i) an optimal solution (if it exists) occurs at a vertex (i.e. extreme point), (ii) the set of optimal points (if it is bounded) is the convex hull of the optimal vertices, (iii) there is only a finite number of vertices, and (iv) each local optimal point is optimal (in the following "optimal" means always "global optimal"). The interesting output data of linear programming problems are the optimal value of the objective function and the set of optimal vertices.

In Section 1.1 basic results and notations for linear programming problems are repeated. In Section 1.2 a method that computes error bounds for all interesting output data is described. There, it is assumed that some or all input data of the system matrix, the right hand side, and the objective function vary independently between given lower and upper bounds, and that this interval problem is basisstable; that is, all real problems contained in the interval problem have the same optimal basis. In Section 1.3 the basisinstable case is discussed. In Section 1.4 a case study that is reported by Wardle 1965 [85] is discussed where we show how these methods work. Section 1.5 contains some additional remarks.

In Section 2 a self-validating method for differentiable and non-differentiable global optimization problems is presented. In contrast to linear programming problems we have

no assumptions like linearity or convexity, and in general many local minima may exist. Murty and Kabadi 1987 [56] proved that the global optimization problem is NP-hard, even if the objective function is quadratic. In Section 2.1 a new branch and bound algorithm for the global optimization problem with bound constraints is described which is designed for problems where derivatives are not available or very costly. Roughly speaking, this algorithm consists of a repeated application of a bisection strategy in connection with local optimization algorithms. The bounds calculated for the global minimum are proved to be correct, all rounding errors are rigorously estimated, and in most cases the bounds calculated for the global minimum points are rough compared to the bounds for the global minimum value. Section 2.2 contains some convergence statements, and in Section 2.3 numerical results for some well-known test problems are given. In Section 2.4 we introduce a new scheme for proving existence and uniqueness of a stationary point. For this scheme it is necessary that inclusion functions of the gradient and the Hessian are available. Then, in Section 2.5 it is shown how this scheme can be incorporated in the branch and bound method of Section 2.1. This modified method leads in many cases to an acceleration, and very sharp bounds for the global minimum value **and** the global minimum points are calculated. Some numerical experiments of this modified method are given in Section 2.6. Section 2.7 contains some additional remarks.

1. Linear Programming

Deputizing for the opinion of many scientist about linear programming we quote Chvatal 1983 [12], page 148:

> "In linear programming problems arising from applications, the numerical data often represent only rough estimates of quantities that are inherently difficult to measure or predict. Market prices may fluctuate, supplies of raw materials and demands for finished products are often unknown in advance, and production may be affected by a variety of accidental events such as machine breakdowns. *In such cases, solving the initial LP problem is only the starting point for further analysis of the situation.*
>
> By replacing the original data by more pessimistic or optimistic estimates of the unknown quantities, we may create a number of variations on the original theme. Since each of these new LP problems might possibly represent the actual situation, *it is useful to find out how the optimal solutions vary with the changes in data.* It may be, for example, that the optimal solution is particularly sensitive to changes in only a small set of parameters; if possible, these parameters should then be estimated with greater accuracy."

In this section, self-validating methods for linear programming problems with interval input data are described. A linear programming problem with interval input data can

be viewed as a linear parametric optimization problem where the coefficients vary independently between the given lower and upper interval bounds. For such problems error bounds for the variation of the objective function and the variation of the optimal vertices are calculated. The bounds are proved to be correct, all rounding errors are rigorously estimated, and a sensitivity analysis is given.

We assume that the reader is familiar with the basic concepts of interval arithmetic, and we use throughout this paper the following definitions and notations. By $\mathbb{R}, \mathbb{R}^n, \mathbb{R}^{m\times n}$ we denote the set of real numbers, real vectors, and real $m \times n$ matrices, respectively. By $\mathbb{IR}, \mathbb{IR}^n, \mathbb{IR}^{m\times n}$ we denote the set of real compact intervals $[a] = [\underline{a}, \overline{a}]$, real interval vectors $[x] = [\underline{x}, \overline{x}] = ([\underline{x}_i, \overline{x}_i])$, and real $m \times n$ interval matrices $[A] = [\underline{A}, \overline{A}] = ([\underline{a}_{i,j}, \overline{a}_{i,j}])$, respectively. For interval quantities $[x]$ the absolute value $|[x]|$, comparisons $<, \leq$, the midpoint $m([x]) := (\underline{x} + \overline{x})/2$, and the width $w([x]) := (\underline{x} - \overline{x})$ are to be understood componentwise. Inclusion functions (or natural interval evaluations) of real functions $f(x)$ have the property that for a given box $[x]$ an inclusion function contains always the range of f on $[x]$. Therefore, we use for the inclusion function of f the same symbol, and write $f([x])$.

1.1. Basic Concepts of Linear Programming

A linear programming problem (cf. [12], [17], [49], [55]) is an optimization problem where the objective function is linear in the variables, and the constraints consist of linear equalities and/or linear inequalities. Linear programming problems can be expressed in different forms. The *standard form* is

$$\text{Max}\{ f(x) \mid x \in X \},\ f(x) := c^t x,\ X := \{ x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0 \}, \tag{1.1}$$

where $A = (a_1, \ldots, a_n)$ is a real $m \times n$ matrix with column vectors $a_1, \ldots, a_n \in \mathbb{R}^m$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $m < n$. All different forms in linear programming are equivalent, and can be put into standard form by simple transformations.

The function $f(x) := c^t x$ is called the *objective function*, X is called the set of *feasible solutions* or *feasible points*, and a point $x^* \in X$ with $c^t x^* \geq c^t x$ for all $x \in X$ is called (global) *optimal*. If an optimal solution $x^* \in X$ exists, then the value $f^* := f(x^*)$ is called the (global) *optimum value*.

Associated with the standard form (1.1) is the corresponding *dual linear programming problem*

$$\text{Min}\{ g(y) \mid y \in Y \},\ g(y) := b^t y,\ Y := \{ y \in \mathbb{R}^m \mid A^t y \geq c \}. \tag{1.2}$$

Y is called the set of *dual feasible solutions*, and an optimal solution of (1.2) is also called *dual optimal*. For $x \in X$, $y \in Y$ we have

$$f(x) = c^t x \leq (A^t y)^t x = y^t (Ax) = b^t y = g(y),$$

and the main part of the well-known Duality theorem of linear programming states that, if X and Y are non-empty, then

$$\text{Max}\{ f(x) \mid x \in X \} = \text{Min}\{ g(y) \mid y \in Y \}. \tag{1.3}$$

The duality theory is a very important and powerful concept, and there is a close relationship to sensitivity analysis. Moreover, the dual problem has important economic interpretations.

The input data of (1.1) and (1.2) are given by the triple $p = (A, b, c)$. The interesting output data are the optimal value f^* and the set of optimal solutions of (1.1) and (1.2). The sets of optimal solutions are the convex hull of their corresponding optimal vertices provided that these sets are bounded. Therefore the interesting output data are the *optimal value* f^*, the *set of optimal vertices* X^* of (1.1), and the *set of optimal vertices* Y^* of (1.2).

The vertices of the convex polyhedron X are algebraically characterized by the property that at most m components are not equal zero. Solutions of $Ax = b$, where at least $n - m$ variables are equal to zero, can be described by partitioning the matrix $A = (a_1, \ldots, a_n)$ into two submatrices $A_B := (a_{\beta_1}, \ldots, a_{\beta_m})$, $A_N := (a_{\gamma_1}, \ldots, a_{\gamma_{n-m}})$ with corresponding sets of indices $B := \{\beta_1, \ldots, \beta_m\} \subseteq \{1, \ldots, n\}$ and $N := \{\gamma_1, \ldots, \gamma_{n-m}\} := \{1, \ldots, n\} \setminus B$. If the $m \times m$ submatrix A_B is regular then B is called a *basic-index-set*. N is called the set of *nonbasic indices*. For a given partitioning B, N the equation $Ax = b$ can be expressed in the form

$$A_B x_B + A_N x_N = b \tag{1.4}$$

where $x^t := (x_B^t, x_N^t)$ is partitioned analogously. By multiplying equation (1.4) with the inverse A_B^{-1} it follows that

$$x_B = A_B^{-1} b - S_N x_N \tag{1.5}$$

where $S_N := A_B^{-1} A_N$. With $c^t := (c_B^t, c_N^t)$ the objective function can be expressed in the form

$$c^t x = c_B^t x_B + c_N^t x_N = c_B^t (A_B^{-1} b - S_N x_N) + c_N^t x_N = c_B^t A_B^{-1} b - (c_B^t S_N - c_N^t) x_N,$$

and therefore

$$c^t x = c_B^t A_B^{-1} b - d_N^t x_N \tag{1.6}$$

with

$$d_N := S_N^t c_B - c_N = A_N^t (A_B^t)^{-1} c_B - c_N. \tag{1.7}$$

Notice that the objective function $f(x) = c^t x$ and the variables x_B are determined by the $n - m$ variables x_N. The solution

$$x(B) := \begin{pmatrix} x_B \\ x_N \end{pmatrix} \quad \text{with } x_B := A_B^{-1} b,\ x_N := 0 \tag{1.8}$$

is called a *basic solution* (corresponding to the basic-index-set B) of $Ax = b$. In addition, if $x_B := A_B^{-1} b \geq 0$, then $x(B) \in X$ and $x(B)$ is called a *feasible basic solution.* The feasible basic solutions correspond to the vertices of the convex polyhedron X.

The so-called vector of *reduced costs* d_N determines the dependency of the objective function $f(x)$ on the non-basic variables x_N. If $d_N \geq 0$ then, using (1.7), the vector

$$y(B) := (A_B^t)^{-1} c_B \tag{1.9}$$

satisfies $A_N^t y(B) - c_N \geq 0$ which yields $y(B) \in Y$. If $d_N \geq 0$ then $y(B)$ is called a *dual feasible basic solution.* The dual feasible basic solutions correspond to the vertices of the convex polyhedron Y.

If $x_B := A_B^{-1} b \geq 0$ and $d_N := A_N^t y(B) - c_N \geq 0$ (that is $x(B) \in X$ and $y(B) \in Y$), then the basic-index-set B is called *optimal.* In this case, the equality

$$c^t x(B) = c_B^t x_B = c_B^t A_B^{-1} b = b^t (A_B^t)^{-1} c_B = b^t y(B) \tag{1.10}$$

holds and by the Duality theorem (1.3) it follows that $x(B)$ and $y(B)$ are optimal vertices. The set of optimal basic-index-sets is denoted by V^*.

Using the well-known Complementary slackness theorem, it follows immediately that $x(B)$ and $y(B)$ are the unique optimal solutions of (1.1) and (1.2) provided $x_B > 0$, $d_N > 0$. In this case $V^* = \{B\}$. A linear programming problem is called *basisstable* if there exists a basic-index-set B with $x_B > 0$ and $d_N > 0$. Linear programming problems which have an optimal solution, and which are not basisstable are called *basisinstable.*

Basisinstable linear programming problems are characterized by the property that there exist optimal vertices of (1.1) and/or (1.2) which are degenerated. Former, degeneracy seemed to be an accident but yet, degeneracy is frequently arising in practical applications (cf. Chvatal 1983 [12], page 30, Kotiah and Steinberg 1977 [41] and 1978 [42]).

1.2. Basisstable linear programming problems with interval input data

To perform an error and sensitivity analysis for linear programming problems we describe the input data by the triple

$$[p] = ([A], [b], [c]) \tag{1.11}$$

where $[A]$ is an $m \times n$ interval matrix, $[b]$ and $[c]$ are interval vectors with m and n components, respectively, and $m < n$. The triple $[p]$ defines a set of linear programming problems with real input data $p \in [p]$.

We introduce the following notation: for a quantity Q which is defined in Section 1.1 $Q(p)$ denotes the dependency of this quantity on the parameter $p \in [p]$, and

$$Q([p]) := \{ Q(p) \mid p \in [p] \} \tag{1.12}$$

is called the corresponding *solution set.*

Therefore, $f^*(p)$ is the optimal value of the linear programming problem with real input data $p \in [p]$, $x_B(p) = A_B^{-1}b$ where $p = (A_B, A_N, b, c) \in [p]$, and the quantities $V^*(p)$, $X^*(p)$, $Y^*(p)$, $d_N(p)$, $x(B;p)$, $y(B;p)$ are defined analogously. Obviously the corresponding solution sets are given by

$$\begin{aligned}
&f^*([p]) = \{ f^*(p) \in \mathbb{R} \mid p \in [p] \}, \\
&x_B([p]) = \{ x_B(p) \in \mathbb{R}^m \mid A_B x_B(p) = b,\ p = (A_B, A_N, b, c) \in [p] \}, \\
&y(B;[p]) = \{ y(B;p) \in \mathbb{R}^m \mid A_B^t y(B;p) = c_B, p = (A_B, A_N, b, c,) \in [p] \}, \\
&d_N([p]) = \left\{ d_N(p) \in \mathbb{R}^{n-m} \;\middle|\; \begin{array}{l} d_N(p) := A_N^t (A_b^t)^{-1} c_B - c_N, \\ p = (A_B, A_N, b, c) \in [p] \end{array} \right\}, \\
&V^*([p]) = \{ B \mid p \in [p], B \in V^*(p) \},
\end{aligned} \tag{1.13}$$

and $X^*([p])$, $Y^*([p])$, $x(B;[p])$ etc. are defined analogously.

The interesting output data of the linear programming problems where p varies in $[p]$ are the sets $f^*([p])$, $X^*([p])$, $Y^*([p])$, and $V^*([p])$. We are interested in reliable and very sharp lower and upper bounds for the output data. In this section we describe a method for the basisstable case.

If B is a basic-index-set for all $p \in [p]$, then B is called a *basic-index-set of* $[p]$; that is A_B is regular for all $A_B \in [A_B]$. A linear programming problem with interval input data $[p]$ is called *basisstable* if a basic-index-set B of $[p]$ exists such that $x_B([p]) > 0$ and $d_N([p]) > 0$. Linear programming problems which have optimal solutions for some $p \in [p]$, and which are not basisstable are called *basisinstable.*

The following corollary is an immediate consequence of the Duality theorem, the Complementary slackness theorem, and our definitions.

Corollary 1.1: Assume that the linear programming problem with interval input data $[p]$ is basisstable with $x_B([p]) > 0$ and $d_N([p]) > 0$. Then

$$V^*([p]) = \{B\},\ X^*([p]) = x(B;[p]),\ Y^*([p]) = y(B;[p]), \tag{1.14}$$

and $f^*([p]) \subseteq [c_B]^t \cdot x_B([p]) \cap [b]^t \cdot y(B;[p])$.

The self-validating method for basisstable linear programming problems which we describe in this section is an immediate consequence of Corollary 1.1. The main idea is to calculate an approximation of an optimal solution together with the corresponding basic-index-set B with the simplex method, and then, a posteriori, to compute bounds for the solution sets $x_B([p])$, $y(B;[p])$, and $d_N([p])$ by using a self-validating method for linear interval systems. Several methods for solving linear interval systems have been proposed; see for example Alefeld and Herzberger 1983 [4], Alefeld 1994 [1], Moore 1979 [53], Neumaier 1990 [59], Rump 1983 [73], and Rump 1994 [72].

algorithm BASISSTABLE LP$([p], B, [x(B)], [y(B)], [f^*])$

begin

call the simplex method for the real input data $m([p])$; (1.15)

if the simplex method terminates without calculating (1.16)
an approximation of an optimal solution
then STOP with WARNING (1);

call a self-validating method that calculates
inclusion vectors $[x_B]$ and $[y(B)]$ of the
solution sets for the linear interval systems
$A_B x_B(p) = b,\ A_B^t y(B;p) = c_B,\ p \in [p]$ (1.17)
where B is the optimal basic-index-set calculated
by the simplex method;

if no inclusions $[x_B]$, $[y(B)]$ can be calculated (1.18)
then STOP with WARNING (2);

Set $[x(B)] := \binom{[x_B]}{x_N}$ with $x_N := 0$, $[d_N] := [A_N]^t[y(B)] - [c_N]$ (1.19)
and $[f^*] := [c_B]^t[x_B] \cap [b]^t[y(B)]$;

if the inequalities $[x_B] > 0$ and $[d_N] > 0$ are (1.20)
not satisfied **then WARNING (3)**;

end;

Theorem 1.2: Suppose that algorithm BASISSTABLE LP terminates without giving any WARNING. Then the linear programming problem with interval input data $[p]$ is ba-

sisstable, each problem with input data $p \in [p]$ has an optimal solution, and the following inclusion conditions are satisfied:

$$V^*([p]) = \{B\},\ X^*([p]) \subseteq [x(B)],\ Y^*([p]) \subseteq [y(B)],\ \text{and}\ f^*([p]) \subseteq [f^*].$$

Proof. The proof follows from Corollary 1.1, and noticing that $x(B;[p]) \subseteq [x(B)]$, $y(B;[p]) \subseteq [y(B)]$, and $[c_B]^t x_B([p]) \cap [b]^t y(B;[p]) \subseteq [f^*]$. ■

Note that this algorithm is in the spirit of Wilkinson. First, in step (1.15) an approximation of the optimal solution for the midpoint problem is calculated, and then a posteriori in step (1.17) interval arithmetic is used.

A termination with WARNING (1) indicates that some or all point problems $p \in [p]$ have no optimal solutions. If the algorithm terminates with WARNING (2), then in many cases a matrix $A_B \in [A_B]$ is singular or close to singularity. Both cases point to an instable behaviour of the linear programming problem with interval input data $[p]$, and therefore, the algorithm which aims to prove stability and solvability for all point problems with $p \in [p]$ is stopped.

If the algorithm terminates with WARNING (3), then basisstability is not verified. This termination often occurs for interval input data and is very unsatisfactory. Because in this case it is not known whether all point problems with $p \in [p]$ have optimal solutions or not. However, if the algorithm terminates with WARNING (2) or WARNING (3) the calculated approximation (cf. (1.15)) is given to the user. This is just in the spirit of Wilkinson. In the next section a method is considered which gives a satisfactory answer to this central question.

1.3. Basisinstable linear programming problems with interval input data

The following method which calculates inclusions of the output data $V^*([p])$, $X^*([p])$, $Y^*([p])$, and $f^*([p])$ for basisinstable linear programming problems can be viewed as a graph-search method (cf. Papadimitriou and Steiglitz 1982 [60]) applied to the graph

$$G^*([p]) := \Big(V^*([p]),\ E^*([p])\Big) \tag{1.21}$$

with the set of *nodes* $V^*([p])$, and the set of *edges*

$$E^*([p]) := \left\{ \{B, B'\} \;\middle|\; \begin{array}{l} B, B' \text{ differ in exactly one index and} \\ \text{there exists a } p \in [p] \text{ with } B, B' \in V^*(p) \end{array} \right\}. \tag{1.22}$$

We call $G^*([p])$ the *representation graph* of the linear programming problem with interval input data $[p]$. Two nodes B, B' are called to be *adjacent* if $\{B, B'\} \in E^*([p])$. For

$B \in V^*([p])$ the set $N(B)$ denotes the set of all nodes B' of $G^*([p])$ which are adjacent to B.

The graph-search method is based on the following idea: First, a starting node $B \in V^*([p])$ is calculated. This is done by calling algorithm BASISSTABLE LP for some point problem with $p \in [p]$. Following all nodes of $N(B)$ are calculated, and then all nodes of $N(B')$ with $B' \in N(B)$ are calculated, and so on. Obviously, the graph-search method terminates by calculating all nodes of $G^*([p])$ which are connected to the starting node B in $O\Big(|E^*([p])|\Big)$ time.

The graph $G^*([p])$ is only given implicitly by the linear programming problem. The following four theorems are important not only in its own right, but also as a means of establishing the graph-search method on this implicitly defined graph. The proofs of these theorems are far too lengthly to be included in these notes.

Theorem 1.3 states that the optimal vertices of the standard form (1.1) and the dual problem (1.2) can be generated by the node set $V^*([p])$.

Theorem 1.3: For any $p = (A, b, c)$ the graph $G^*(p)$ is connected and the following equations hold:

$$X^*(p) = \{\, x(B;p) \mid B \in V^*(p) \,\}, \tag{1.23}$$

$$Y^*(p) = \{\, y(B;p) \mid B \in V^*(p) \,\}. \tag{1.24}$$

Proof. See [25]. ■

Theorem 1.4 shows that the representation graph $G^*([p])$ is connected provided some weak assumptions are fulfilled. Therefore, the graph-search method will generate the whole graph $G^*([p])$.

Theorem 1.4: Assume that the sets of optimal solutions of the standard form (1.1) and the dual problem (1.2) are nonempty and bounded for all $p \in [p]$. Suppose further that all nodes $B \in V^*([p])$ are basic-index-sets of $[p]$. Then $G^*([p])$ is connected and the optimal value function $f^*(p) : [p] \to \mathbb{R}$ is continuous.

Proof. See [25]. ■

Simple examples (cf. [25], page 100) can be constructed which demonstrate that $G^*([p])$ is not connected provided that one of the assumptions of Theorem 1.4 is not satisfied. In the next theorem we examine how an inclusion $\widetilde{N}(B)$ of $N(B)$ can be calculated for a

node $B \in V^*([p])$.

Theorem 1.5. Let $B \in V^*([p])$ be a basic-index-set of $[p]$, and assume further that

(a) $[x_B]$, $[y(B)]$, and $[s_\gamma]$ for $\gamma \in N$ are inclusions for the solution sets of the linear interval systems

$$[A_B]x_B = [b],\ [A_B]^t y(B) = [c_B],\ [A_B]s_\gamma = [a_\gamma], \tag{1.25}$$

and

$$[d_N] := [A_N]^t[y(B)] - [c_N], \text{ respectively;} \tag{1.26}$$

(b) $\widetilde{N}(B)$ is the set of all index-sets

$$B' = \big(B\backslash\{\beta\}\big) \cup \{\gamma\},\ \beta \in B,\ \gamma \in N \tag{1.27}$$

that satisfy one of the following two conditions

$$0 \in [d_\gamma],\ \overline{s}_{\beta\gamma} > 0 \text{ ,and } \frac{\underline{x}_\beta}{\overline{s}_{\beta\gamma}} \leq \text{Min}\left\{ \frac{\overline{x}_{\beta'}}{\underline{s}_{\beta'\gamma}} \mid \underline{s}_{\beta'\gamma} > 0,\ \beta' \in B \right\}, \tag{1.28}$$

$$0 \in [x_\beta],\ \underline{s}_{\beta\gamma} < 0 \text{ ,and } \frac{\underline{d}_\gamma}{\underline{s}_{\beta\gamma}} \geq \text{Max}\left\{ \frac{\overline{d}_{\gamma'}}{\overline{s}_{\beta\gamma'}} \mid \overline{s}_{\beta\gamma'} < 0,\ \gamma' \in N \right\}. \tag{1.29}$$

Then the following results hold:

(i) $\widetilde{N}(B) \supseteq N(B)$;
(ii) if $p = [p]$, and the inclusions in (a) coincide with the corresponding exact solutions then $\widetilde{N}(B) = N(B)$.

Proof. See [25]. ■

A major goal of our method is to prove a posteriori that all linear programming problems within the tolerances are well-posed. In analogy to the definition of well-posed equations, a linear programming problem with real input data p is called *well-posed* if there exists an open ball $U_\delta(p)$, $\delta > 0$ such that the sets of optimal solutions of the linear programming problems with input data $\widetilde{p} \in U_\delta(p)$ are nonempty, and the optimal value function f^* is continuous on $U_\delta(p)$. Otherwise, the linear programming problem is called *ill-posed.* The following theorem characterizes well-posed linear programming problems.

Theorem 1.6. For a linear programming problem with real input data $p = (A, b, c)$ the following statements are equivalent:

(i) the problem is well-posed;
(ii) the optimal solution sets of the standard form (1.1) and the dual problem (1.2) are

nonempty and bounded;

(iii) Rank $(A) = m$ and the systems of inequalities

$$Ax = b,\ x > 0, \text{ and } A^t y > c \tag{1.30}$$

have solutions $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$;

(iv) There exists a ball $U_\delta(p)$, $\delta > 0$ such that $X(\tilde{p}) \neq \emptyset$ and $Y(\tilde{p}) \neq \emptyset$ for all $\tilde{p} \in U_\delta(p)$.

(v) Rank $(A) = m$, $X(p) \neq \emptyset$, $Y(p) \neq \emptyset$, and for each $B \in V^*(p)$ the following two conditions are satisfied:

$$\forall\ \gamma' \in N \text{ with } d_{\gamma'} = 0\ \exists\ \beta' \in B \text{ such that } s_{\beta'\gamma'} > 0, \tag{1.31}$$

$$\forall\ \beta' \in B \text{ with } x_{\beta'} = 0\ \exists\ \gamma' \in N \text{ such that } s_{\beta'\gamma'} < 0. \tag{1.32}$$

Proof. See [25]. ■

Because in the basisstable case the optimal solutions are unique, from Theorem 1.6 (ii) it follows that basisstable linear programming problems are well-posed. Moreover, we see that linear programming problems with Rank $(A) < m$ are ill-posed. Because of (iv), for ill-posed linear programming problems with input data p there exist sequences $(\tilde{p}^{(i)})_{i \in \mathbb{N}}$ with $\tilde{p}^{(i)} \to p$ such that the set of feasible solutions $X(\tilde{p}^{(i)})$ and/or $Y(\tilde{p}^{(i)})$ are empty. Obviously, such problems cannot be solved in the usual way. Ill-posed linear programming problems must be solved with special regularization techniques where a priori information about the problem is needed in addition (cf. Tikhonov and Arsenin 1977 [83], Engl and Groetsch 1987 [19]).

An important characterization of well-posed problems for the basisinstable case is statement (v) which says that, by passing only through the optimal basic-index-sets, we can decide whether the problem is well-posed or not. This is a fundamental property which is used in the following method BASISINSTABLE LP for verifying a posteriori that all problems with $p \in [p]$ are well-posed.

algorithm BASISINSTABLE LP $([p], S(V^*), S(X^*), S(Y^*), S(f^*))$
begin

select a point problem $p \in [p]$; (1.33)

call BASISSTABLE LP $(p, B, [x(B)], [y(B)], [f^*])$; (1.34)

if BASISSTABLE LP terminates with a WARNING (i) (i = 1,2,3) (1.35)

then STOP with this WARNING;

initialize lists $W := \{B\}$, $S(V^*) := \emptyset$, $S(X^*) := \emptyset$, $S(Y^*) := \emptyset, S(f^*) := \emptyset$; (1.36)

while $W \neq \emptyset$ **do** (1.37)

begin

choose $B \in W$; $W = W \backslash \{B\}$; (1.38)

call a self-validating method that calculates inclusions (1.39)

$[x_B]$, $[y(B)]$, $[s_\gamma]$ with $\gamma \in \mathbb{N}$, and $[d_N]$ of the
systems (1.25), (1.26);
if some inclusions in step (1.39) cannot be calculated (1.40)
then STOP with WARNING (4);

if $\overline{x}_B \geq 0$ **and** $\overline{d}_N \geq 0$ **then** (1.41)
begin
if one of the following two conditions (1.42)
(i) $0 \in [x_\beta]$, $\beta \in B \Rightarrow \exists\, \gamma \in N$ such that $[s_{\beta\gamma}] < 0$
(ii) $0 \in [d_\gamma]$, $\gamma \in N \Rightarrow \exists\, \beta \in B$ such that $[s_{\beta\gamma}] > 0$
is not valid **then STOP with WARNING(5)**;
set $[x(B)] := \binom{[x_B]}{x_N}$ with $x_N := 0$; (1.43)
set $[f^*(B)] := [c_B]^t[x_B] \cap [b]^t[y(B)]$; (1.44)
set $S(V^*) := S(V^*) \cup \{B\}$, $S(X^*) \cup \{[x(B)]\}$, (1.45)
$S(Y^*) := S(Y^*) \cup \{[y(B)]\}$, $S(f^*) := S(f^*) \cup \{[f^*(B)]\}$;
calculate $\widetilde{N}(B)$ by using Theorem 1.5; (1.46)
set $S(V^*) := S(V^*) \cup \{B\}$; set $W := W \cup \{\widetilde{N}\ (\mathrm{B}) \setminus S(V^*)\}$; (1.47)
end;
end;
end;

First, in steps (1.33) and (1.34) of algorithm BASISINSTABLE LP, for a selected point problem $p \in [p]$ a starting node $B \in V^*([p])$ is calculated by calling algorithm BASISSTABLE LP. If BASISSTABLE LP gives a WARNING, then we stop our algorithm. As mentioned in Section 1.2, WARNING(1) and WARNING(2) indicate an instable behaviour of our problem. In the case of WARNING(3), the selected point problem p is basisinstable or close to basisinstability. Because in each neighbourhood of a basisinstable point problem there exists basisstable point problems, we may call BASISSTABLE LP for another slightly perturbated point problem, and proceed in our algorithm.

In (1.36) the "working list" W contains our starting node B, and four lists $S(V^*)$, $S(X^*)$, $S(Y^*)$, $S(f^*)$ are set empty. These lists will contain inclusions of the output data. The while-loop (1.37) to (1.47) implements the graph-search method for the implicitly defined graph $G^*([p])$. The main steps are (1.38) where a node B is selected, then (1.46) where the adjacent nodes are calculated, and (1.47) where all nodes of $N(B)$ which are not contained in $S(V^*)$ are stored on list W.

Note that for each $B \in V^*([p])$ there exists a $p \in [p]$ with $x_B(p) \geq 0$ and $d_N(p) \geq 0$. Therefore, a necessary condition for $B \in V^*([p])$ is (1.41). Obviously, B is only admitted to $S(V^*)$ provided that this necessary condition is fulfilled.

WARNING(4) points out that there exists a singular matrix $A_B \in [A_B]$, or such a matrix may be close to singularity. Therefore an assumption of Theorem 1.4 is not satisfied, and the graph $G^*([p])$ may be not connected. The conditions in step (1.42) are an interval

version of the characterization of well-posed problems (cf. Theorem 1.6, (v)). Hence, WARNING(5) indicates that there may exist a problem with $p \in [p]$ which is ill-posed or close to an ill-posed problem.

The following theorem shows that algorithm BASISINSTABLE LP verifies a posteriori that all problems with $p \in [p]$ are well-posed, and gives inclusions of the output data.

Theorem 1.7: Suppose that algorithm BASISINSTABLE LP terminates without giving any WARNING. Then

(i) Each problem with $p \in [p]$ is well-posed;
(ii) the optimal value function $f^*(p) : [p] \to \mathbb{R}$ is continuous;
(iii) the representation graph $G^*([p])$ is connected;
(iv) the following inclusion conditions hold:

$$V^*([p]) \subseteq S(V^*), \tag{1.48}$$
$$X^*([p]) \subseteq \bigcup \{[x(B)] \in S(X^*)\}, \tag{1.49}$$
$$Y^*([p]) \subseteq \bigcup \{[y(B)] \in S(Y^*)\}, \tag{1.50}$$
$$f^*([p]) \subseteq \bigcup \{[f^*(B)] \in S(f^*)\}. \tag{1.51}$$

Proof. See [25]. ■

1.4. Management of New Forest

This case study is a real world problem reported by Wardle 1965 [85] and Chvatal 1983 [12]. They discuss a felling program for New Forest, a park of 145 square miles situated in Hampshire, England. We follow the description of Chvatal. The management of New Forest had to choose a felling program for an area of about 8500 acres consisting of six different crop types listed in Table 1.4.1.

Crop type	Description	Acres	Volume if felled (h.ft./acre)
1	High-volume hardwoods	2754	2000
2	Medium-volume hardwoods	850	1200
3	Low-volume hardwoods	855	700
4	Conifer high forest	1598	4000
5	Mixed high forest	405	2500
6	Bare land	1761	

Table 1.4.1: Crop Types

The volume of wood is measured in Hoppus foot, abbreviated h.ft. This is the volume of a board 1 foot square and 1 inch thick. The hardwood areas are classified w.r.t. their undergrowth as follows:

	Complete undergrowth	Partial undergrowth	No undergrowth	Total
High-volume hardwoods	357	500	1897	2754
Medium-volume hardwoods	197	130	523	850
Low-volume hardwoods	39	170	646	855

Table 1.4.2: Classification of Hardwood Areas

For any number of acres of any crop type two basic treatments can be choosen:

1.1) Fell and plant conifer.
1.2) Fell and plant hardwood.

For bare land these treatments become

1.1) Plant conifer.
1.2) Plant hardwood.

Additionally to these basic treatments, the management has the following options:

2. For hardwood areas with complete undergrowth: fell and retain the undergrowth.
3. For hardwood areas with partial undergrowth: fell and enrich the undergrowth.
4. Postpone treatments alltogether for any number of acres of any crop type.

The estimate of the net discounted revenue over the next ten years varies w.r.t. the chosen treatments and crop types. The corresponding Table 1.4.3 with coefficients measured in pounds per acre is given below.

	Treatment				
Crop type	1.1	1.2	2	3	4
1	287	215	228	292	204
2	207	135	148	212	148
3	157	85	98	162	112
4	487	415	—	—	371
5	337	265	—	—	264
6	87	15	—	—	61

Table 1.4.3: Net discounted revenue (£/ acre)

Because of some visual requirements and a limited labor capacity, the following four conditions must be satisfied.

a) At most 5000 acres can be treated.
b) The resulting conifer area must not exceed 3845 acres.
c) At most 2.44 million h.ft. of hardwood are allowed to be felled.
d) At most 4.16 million h.ft. of conifer and mixed high forest can be felled.
e) At least 500 acres must be planted with hardwood.

To model this problem, let us first look at Table 1.4.3. Note that for each of the six crop types the difference between treatments 1.1 and 1.2 is exactly £72. This is because the net discounted revenue is equal to the sum of two components:

Felling:	£1 per h.ft.
Planting:	£87 if conifer is planted, and
	£15 if hardwood is planted.

Therefore, in the case of only felling, we need not to distinguish between treatments 1.1 und 1.2. When we concern only planting it follows immediately: The crop types formerly occupying the forest are irrelevant, the essential fact are the total acreages of newly planted conifer and newly planted hardwood. Hence, in the following we refer to both treatments 1.1 and 1.2 as treatment 1.

For any crop type and any treatment we introduce a variable x_{ij} that defines the acres of crop type i receiving treatment j. By examining Table 1.4.3 we get the following variables:

$x_{11}, x_{12}, x_{13}, x_{14}$	for crop type 1	receiving treatment $j = 1, \ldots, 4$.
$x_{21}, x_{22}, x_{23}, x_{24}$	for crop type 2	receiving treatment $j = 1, \ldots, 4$.
$x_{31}, x_{32}, x_{33}, x_{34}$	for crop type 3	receiving treatment $j = 1, \ldots, 4$.
x_{41}, x_{44}	for crop type 4	receiving treatment $j = 1, 4$.
x_{51}, x_{54}	for crop type 5	receiving treatment $j = 1, 4$.
x_{61}, x_{64}	for crop type 6	receiving treatment $j = 1, 4$.

By defining x_0 as the acres which are planted with conifer, it follows that the acres which are planted with hardwood equals

$$x_{11} + x_{12} + x_{31} + x_{41} + x_{51} + x_{61} - x_0.$$

The resulting objective function is

$$\begin{aligned} f(x) \; &= 215x_{11} + 228x_{12} + 292x_{13} + 204x_{14} + 135x_{21} + 148x_{22} + 212x_{23} + 148x_{24} \\ &\quad + 85x_{31} + 98x_{32} + 162x_{33} + 112x_{34} + 415x_{41} + 371x_{44} + 265x_{51} + 264x_{54} \\ &\quad + 15x_{61} + 61x_{64} + 72x_0. \end{aligned}$$

Because we do not distinguish between treatments 1.1 and 1.2, the term $72x_0$ must be added.

From our discussion above it follows that we wish to maximize $f(x)$ subject to the constraints

$$
\begin{array}{rcrcrcrcl}
x_{11} & + & x_{12} & + & x_{13} & + & x_{14} & = & 2754 \\
x_{21} & + & x_{22} & + & x_{23} & + & x_{24} & = & 850 \\
x_{31} & + & x_{32} & + & x_{33} & + & x_{34} & = & 855 \\
x_{12} & & & & & & & \leq & 357 \\
x_{13} & & & & & & & \leq & 500 \\
x_{22} & & & & & & & \leq & 197 \\
x_{23} & & & & & & & \leq & 130 \\
x_{32} & & & & & & & \leq & 39 \\
x_{33} & & & & & & & \leq & 170 \\
x_{41} & + & x_{44} & & & & & = & 1598 \\
x_{51} & + & x_{54} & & & & & = & 405 \\
x_{61} & + & x_{64} & & & & & = & 1761
\end{array}
$$

$$
\begin{aligned}
& x_0, x_{ij} \geq 0 \\
& \sum_{i,j} x_{ij} \leq 5000 \\
& x_0 - x_{41} \leq 2247 \\
& 2(x_{11} + x_{12} + x_{13}) + 1.2(x_{21} + x_{22} + x_{23}) + 0.7(x_{31} + x_{32} + x_{33}) \leq 2440 \\
& 4x_{41} + 2.5x_{51} \leq 4160 \\
& x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{61} - x_0 \geq 500 \\
& x_{ij} \geq 0 \text{ for all } i, j.
\end{aligned}
$$

This is a linear programming problem and the simplex method finds the (rounded) optimal solution

$$
\begin{aligned}
& x_{11}^* = 365,\ x_{13}^* = 500,\ x_{23}^* = 130,\ x_{33}^* = 170,\ x_{31}^* = 621,\ x_{41}^* = 1040, \\
& x_{61}^* = 1761,\ x_0^* = 3287.
\end{aligned}
$$

All remaining variables are equal zero. This felling program yields a total net discounted revenue of £1 840 000.

Most of the input data for the forest problem are, due to measurements, not known exactly. But, depending on the measuring method, more or less sharp bounds for these input data can be given. The question is, how reliable are the predictions that are calculated by the simplex method. In this context the most important question for managing New Forest is:

(i) How changes the total net discounted revenue w.r.t. the uncertain input data?

An answer to this question gives the sensitivity of the objective function. Regarding that many subjective and visual aspects appear in managing this forest, we may ask:

(ii) How does the optimal basic solution change, and are there alternatively other basic solutions which are optimal for some $p \in [p]$?

A very precise measurement of some input data may be very costly or unpossible. Hence, a third important question is:

(iii) How sensitive is the problem for different variations of some coefficients of the input data?

To perform a sensitivity and error analysis we applied the algorithm described in Section 1.3, where the input data are intervals that are given in the following form:

The coefficients of the system matrix A that describe the volumes of the crop types are defined by

$$[a_{ij}] := [a_{ij} - \frac{1}{2}w_{\text{rel}}(A)|a_{ij}|,\ a_{ij} + \frac{1}{2}w_{\text{rel}}(A)|a_{ij}|].$$

Therefore, these coefficients are relatively perturbated such that the relative width of the intervals equals $100w_{\text{rel}}(A)$ % w.r.t. the midpoint a_{ij}.

The coefficients of the right hand side b are relatively perturbated by

$$[b_i] := [b_i - \frac{1}{2}w_{\text{rel}}(b)|b_i|,\ b_i + \frac{1}{2}w_{\text{rel}}(b)|b_i|],$$

and analogously the coefficients that define our objective function (cf. Table 1.4.3) are given by

$$[c_j] := [c_j - \frac{1}{2}w_{\text{rel}}(c)|c_j|,\ c_j + \frac{1}{2}w_{\text{rel}}(c)|c_i|].$$

By performing the algorithm of Section 1.3 for the interval problem $[p] = ([A], [b], [c])$ with $w_{\text{rel}}(A) = 0.001$, $w_{\text{rel}}(b) = 0.004$, and $w_{\text{rel}}(c) = 0.02$ (thus, the coefficients of A, b, and c are relatively perturbated by 0.1 %, 0.4 %, 2 %, respectively), we get the (rounded) results that are given in Table 1.4.4 and Table 1.4.5:

	Solution $[x^{(1)}]$	Solution $[x^{(2)}]$	Solution $[x^{(3)}]$	Solution $[x^{(4)}]$	Solution $[x^{(5)}]$
x_0	[3286, 3288]	[3286, 3288]	[3286, 3288]	[3286, 3288]	[3286, 3288]
x_{11}	[360, 370]	[340, 345]	0	0	[82, 94]
x_{12}	0	0	0	0	0
x_{13}	[499, 501]	[499, 501]	[499, 501]	[499, 501]	[499, 501]
x_{14}	[1879, 1899]	[1904, 1918]	[2248, 2260]	[2248, 2260]	[2155, 2176]
x_{21}	0	0	[567, 576]	[719, 721]	[719, 721]
x_{22}	0	0	0	0	0
x_{23}	[129.8, 130.2]	[129.8, 130.2]	[129.8, 130.2]	[129.8, 130.2]	[129.8, 130.2]
x_{24}	[719, 721]	[719, 721]	[143, 155]	0	0
x_{31}	[612, 630]	[683, 687]	[683, 687]	[421, 439]	[168, 188]
x_{32}	0	0	0	0	0
x_{33}	[169.7, 170.3]	[169.7, 170.3]	[169.7, 170.3]	[169.7, 170.3]	[169.7, 170.3]
x_{34}	[53, 75]	0	0	[245, 265]	[495, 519]
x_{41}	[1039, 1041]	[1039, 1041]	[1039, 1041]	[1039, 1041]	[1039, 1041]
x_{44}	[554, 562]	[554, 562]	[554, 562]	[554, 562]	[554, 562]
x_{51}	0	0	0	0	0
x_{54}	[404, 406]	[404, 406]	[404, 406]	[404, 406]	[404, 406]
x_{61}	[1758, 1764]	[1716, 1723]	[1486, 1495]	[1589, 1605]	[1758, 1764]
x_{64}	0	[35, 49]	[262, 278]	[153, 175]	0

Table 1.4.4: Bounds, calculated for $X^*([P])$

	Solution $[x^{(1)}]$	Solution $[x^{(2)}]$	Solution $[x^{(3)}]$	Solution $[x^{(4)}]$	Solution $[x^{(5)}]$
net discounted revenue	[1813327, 1866980]	[1815832, 1864302]	[1814581, 1864194]	[1813893, 1865025]	[1812447, 1866984]

Table 1.4.5: Bounds, calculated for $f^*\big([P]\big)$

We can draw several conclusions from these results. Some of them are mentioned here:

From Table 1.4.5 it follows that $f^*([P]) \subseteq [1812447, 1866984]$. The relative diameter of the intervals that are calculated for the objective function as well as for the components of the 5 interval vectors $[x^{(i)}]$ is very small w.r.t. the perturbations of $[p]$. Hence, it is proved that the problem is very stable for the objective function, but it is not basisstable.

The interval vectors $[x^{(i)}]$ represent 5 different basic strategies how to manage New Forest. The components of these vectors can be partioned in the following way:

I. Components which are equal to zero for all interval vectors $(x_{12}, x_{22}, x_{32}, x_{51})$. Therefore, treatment 2 is never applied.

II. Components which are equal for all strategies ($x_0, x_{13}, x_{23}, x_{33}, x_{41}, x_{44}, x_{54}$)

III. Components which vary w.r.t. our different solutions $[x^{(i)}]$ (x_{11}, x_{14}, x_{21}, x_{24}, x_{31}, x_{34}, x_{61}, x_{64}). These components may serve to take additional visual aspects into consideration.

$r(A)$	$r(b)$	$r(c)$	Inclusion of $f^*([P])$	$r(f^*)$	$\lvert S(V^*)\rvert$
0.0%	0.0%	0.2%	[1838229, 1841909]	0.2%	1
0.0%	0.0%	0.4%	[1835947, 1843749]	0.4%	2
0.0%	0.0%	0.6%	[1833805, 1845589]	0.6%	5
0.0%	0.0%	2.0%	[1820930, 1858470]	2.0%	5
0.0%	0.0%	3.0%	[1811733, 1867670]	3.0%	5
0.0%	0.0%	4.0%	[1791915, 1876871]	4.6%	9
0.0%	0.4%	0.0%	[1836836, 1843303]	0.3%	1
0.0%	1.0%	0.0%	[1831986, 1848153]	0.9%	1
0.0%	2.0%	0.0%	[1823902, 1856236]	1.7%	1
0.0%	4.0%	0.0%	[1807735, 1872516]	3.5%	2
0.2%	0.0%	0.0%	[1839400, 1840738]	0.05%	1
0.4%	0.0%	0.0%	[1838729, 1841410]	0.2%	1
1.0%	0.0%	0.0%	[1836695, 1843444]	0.4%	1
0.2%	0.2%	0.2%	[1832330, 1847154]	0.8%	2
0.2%	0.2%	0.6%	[1828661, 1850844]	1.2%	5
0.2%	0.2%	2.0%	[1815822, 1863761]	2.6%	5
0.2%	0.2%	4.0%	[1785036, 1882214]	5.3%	11
0.2%	0.4%	0.2%	[1828923, 1850347]	1.1%	2
0.2%	0.4%	0.4%	[1827093, 1852195]	1.4%	2
0.2%	0.4%	0.6%	[1825262, 1854044]	1.6%	5
0.2%	0.4%	2.0%	[1812447, 1866983]	3.0%	5
0.2%	0.4%	4.0%	[1780811, 1886677]	5.7%	11
1.0%	1.0%	0.2%	[1820745, 1858556]	2.0%	2
1.0%	1.0%	0.4%	[1813631, 1865704]	2.8%	5
1.0%	1.0%	1.0%	[1803063, 1876465]	4.0%	5
1.0%	1.0%	2.0%	[1794002, 1885800]	5.0%	5
1.0%	1.0%	4.0%	[1756540, 1913886]	8.6%	12
1.0%	2.0%	0.2%	[1812689, 1866662]	3.0%	2
1.0%	2.0%	0.4%	[1805591, 1873827]	3.8%	5
1.0%	2.0%	2.0%	[1777012, 1903133]	6.9%	5

Table 1.4.6: Behaviour for different perturbations

Much more conclusions can be drawn by looking at the inclusions of the dual solutions. But we finish this case study with Table 1.4.6 that describes the behaviour for some

different perturbations and gives an answer to question (iii). The coefficients are relatively perturbated with $r(A) := 100 \cdot w_{\mathrm{rel}}(A)\%$, $r(b) = 100 \cdot w_{\mathrm{rel}}(b)\%$, $r(c) = 100 \cdot w_{\mathrm{rel}}(c)\%$. $r(f^*)$ denotes the relative width of the inclusion of $f^*([p])$ in percent.

Thus, it is proved that this problem is stable for various perturbations. Moreover, for small perturbations of A and b it is basisstable whereas small perturbations in c yield the basisinstable case. I thank Mrs. U. Maichle who has implemented the algorithm of Section (1.3) using PASCAL-SC.

1.5. Remarks

The main idea of our approach for the basisstable case is due to Krawczyk 1975 [43]. But in contrast to Krawczyk, we incorporate the dual linear programming problem. For other treatments of the basisstable case and related questions, the reader is referred to Machost 1970 [46], Beeck 1978 [8], Steuer 1981 [82], Rump 1983 [73], Rohn 1984 [68], and Rohn 1993 [70]. Algorithms for computing the supremum of the objective function for linear programming problems with interval input data are proposed by Rohn 1984 [68] and Mraz 1990 [54].

The theoretical results of Section 1.3 and the method for basisinstable linear programming problems with interval input data was developed by the author 1985 [25]; see also Jansson 1988 [26], Jansson and Rump 1991 [34]. For a modification of this method in the case of linear programming problems with lower and upper bounds, we refer the reader to Maichle 1988 [47].

The main effort of the methods described in Section 1.2 and 1.3 is to compute guaranteed bounds for the solution sets of linear interval systems. Rohn and Kreinovich 1993 [71] have shown that the calculation of exact bounds for the solution set of a linear interval system is NP-hard. Nevertheless, many methods (cf. [4], [6], [53], [59], [73], and [76]) for calculating very sharp bounds of the solution set were developed. Most of them need to compute an approximate inverse, and they have the complexity $O(n^3)$. Moreover, for some of these methods the componentwise overestimation can rigorously be estimated; that is, guaranteed componentwise inner and outer bounds of the solution set can be calculated (cf. Neumaier 1987 [57] and 1989 [58], and Rump 1990 [74]).

The first algorithm for computing exact bounds for the solution set of a linear interval system was proposed by Rohn 1984 [68], and 1989 [69], and some additional results are given by Neumaier 1990 [59].

Until now, it was an important unsolved problem how guaranteed bounds for the solution set of sparse linear interval systems can be computed without calculating an approximate inverse. Since the inverse of a sparse matrix is in general full, the known self-validating methods have a very limited size for large linear systems. In special situations, for example in the case of M-matrices, the computation of an approximate inverse can be avoided

(cf. Alefeld and Platzöder 1983 [5], Schwandt 1984 [79] and 1985 [80], and Rump 1992 [75]). Now, in a recent paper of Rump 1993 [77], a self-validated direct sparse solver for general real matrices has been developed. Examples with up to 1.000.000 unknowns were presented. Such a direct sparse solver offers the possibility to develop self-validating methods for sparse linear programming problems.

In Sections 1.2 and 1.3 it is assumed that the input data vary independently between their given lower and upper bounds. But in some cases dependencies in the coefficients of the objective function, the right hand side or the system matrix occur. Self-validating methods which take into consideration linear dependencies of interval input data in the case of square linear systems are given in [28] and [29], and Rump 1994 [72] in this volume. For linear programming problems with dependencies in the input data, these solvers can be used to calculate bounds for the linear interval systems which correspond to optimal basic-index-sets.

2. Global Optimization Methods Using Interval Arithmetic

In order to motivate the calculation of guaranteed bounds for nonlinear optimization problems, we discuss the following little application described by Becker and Wittmer 1983 [7]. This example was used as an exercise in a course on nonlinear optimization that the author of this paper gave last summer for students studying electrical engineering. In this course we used MATLAB [62] because of its user-friendly design and its collection of toolboxes.

We were surprised that commercial optimization routines may produce completely wrong approximations for the following simple problem. The problem is to minimize the costs of a voltage stabilizer such that the ripple factor is not too large. A voltage stabilizer (see Figure 2.1) transforms rectified alternating voltage into direct current voltage.

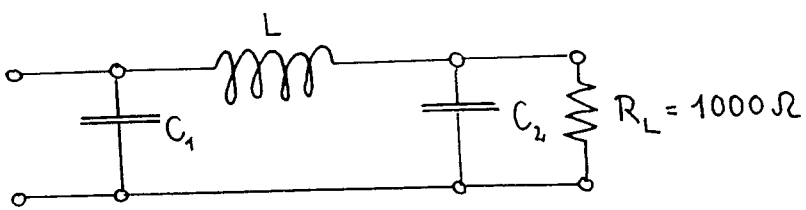


Figure 2.1

The ripple factor is given by

$$r = \frac{3450}{R_L C_1 C_2 L},$$

and r should not exceed 10^{-2}. By using the equations $r = 10^{-2}$, and $R_L = 10^3\Omega$ we get the equation

$$L = \frac{345\Omega^{-1}}{C_1 \cdot C_2}.$$

The costs of the components of the voltage stabilizer are determined from list prices (dated about 1980). The costs for the capacities are approximated by linear interpolation

$$\left(0.25 + 2.5 \cdot 10^{-4} \quad \frac{C}{\mu F}\right) \text{ DM}$$

and the costs for the inductivity are approximated by quadratical interpolation

$$\left(1.0 + 5.0 \cdot 10^{-5} \quad \frac{L^2}{H^2}\right) \text{ DM}$$

(F: Farad, H: Henry, DM: German Mark). Summing up the costs yields the problem

$$\text{minimize } f(C_1, C_2) = 1.5 + 2.5 \cdot 10^{-4}(C_1 + C_2) + \frac{59.5}{C_1^2 \cdot C_2^2}$$

$$\text{subject to } C_1 > 0,\ C_2 > 0.$$

This simple well-conditioned problem is not a typical global optimization problem. It is strictly convex and has therefore one local minimum which is global. With an appropriate self-validating method it easy to verify the following guaranteed bounds for the optimal solution:

$$C_1^* \in [13.6391843461,\ 13.6391843724]\ \mu F,$$

$$C_2^* \in [13.6391843461,\ 13.6391843724]\ \mu F,$$

$$f^* \in [1.50852449021,\ 1.50852449024]\ \text{DM}.$$

For solving this problem we use the MATLAB routine *fminu*. This routine is a BFGS-quasi-Newton method. All the runs use double precision.

Our first starting point is $C_1 = 100\mu F$, $C_2 = 100\mu F$. In each iteration step MATLAB gives a warning: "`Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 8.455280e-22`". After about 250 iteration steps we stopped *fminu*. The last condition number is `RCOND = 4.810885e-250`. *fminu* shows the

same behaviour for starting points which are close to the optimal solution, like $C_1 = 25\mu F$, $C_2 = 25\mu F$ or $C_1 = 19\mu F$, $C_2 = 19\mu F$ or $C_1 = 20\mu F$, $C_2 = 25\mu F$.

Our next starting point is $C_1 = 0.1\mu F$, $C_2 = 0.1\mu F$. The routine *fminu* returns the solution $C_1 = 4608.9\mu F$, $C_2 = 4608.9\mu F$ with a function value of 3.8044 DM.

Obviously, it seems that smaller starting points give better results in terms of solvability. Next, we choose $C_1 = 0.01\mu F$, $C_2 = 0.01\mu F$. Then we get the results $C_1 = 4.6088 \cdot 10^8 \mu F$ and $C_2 = 4.6088 \cdot 10^8 \mu F$ with a function value of 230440 DM.

The starting point $C_1 = 0.1\mu F$, $C_2 = 0.5\mu F$ gives the approximation $C_1 = 184.48\mu F$, $C_2 = 37.377\mu F$ with the function value of 1.5055 DM.

Using the starting point $C_1 = 10\mu F$, $C_2 = 10\mu F$ MATLAB gives the approximation $C_1 = 13.6622\mu F$, $C_2 = 13.6622\mu F$ with a function value of 1.5085 DM. This result is close to the solution. By comparing it with our inclusion we see that only three decimal digits are correct, although the calculations are performed in double precision ($\sim$ 17 decimals).

Summarizing, *fminu* gives warnings that a well-conditioned problem is ill-conditioned, and without any warning *fminu* produces approximations of the solution which are completely wrong in the sense of forward and backward error analysis. We also used the routine E04JAF of the NAG-library. The results are better, but, in principle, this BFGS-quasi-Newton method has the same behaviour.

Obviously, the BFGS-quasi-Newton method, one of the best algorithms, has sometimes problems with computing accurate results, even for very simple applications with practical (not special constructed) models and input data. Moreover, we see that in such simple, strict convex situations the attraction region may be small.

In the following, our major goal is on self-validating methods for differentiable and non-differentiable global optimization problems that use local optimization methods but additionally give guaranteed bounds for the global minimum. These bounds are proved to be correct, all rounding errors are rigorously estimated.

2.1. A Branch and Bound Method Using Local Optimization Techniques

In this section we consider a branch and bound method for the global optimization problem

$$\text{Min}\{ f(x) \mid x \in X \},\ X := [x] := \{ x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x} \} \tag{2.1}$$

where $f : X \to \mathbb{R}$, and $\underline{x}, \overline{x} \in \mathbb{R}^n$ with $\underline{x} \leq \overline{x}$, and the set of *feasible points* X is a *box* or *interval vector* $[x]$. We denote the *global minimum* (if it exists) by f^* and the set of *global minimum points* by X^*.

Branch and bound techniques are frequently used for solving global optimization problems. Usually, during the branching process the feasible domain is successively partitioned and bounds for the global minimum are calculated. Branch and bound methods differ in the way they (i) partition the feasible domain X into subregions $[y]$, (ii) calculate bounds for the range of f on those subregions and (iii) discard subregions for avoiding exhaustive search.

In our branch and bound method *interval evaluations (inclusion functions)* are used for bounding the range of f over subregions. Obviously, a subregion $[y]$ contains no global minimum point and can be discarded if a lower bound of f on $[y]$ is greater than $f(x)$ for some $x \in X$. Therefore, the early knowledge of an approximation $\widetilde{x}$ of a global minimum point or at least of a point $\widetilde{x}$ such that $f(\widetilde{x})$ is close to f^* has important influence upon the efficiency. We incorporate local optimization algorithms to compute such points.

The most important difficulties by using local optimization algorithms for global optimization problems are:

(i) Local optimization algorithms require a starting point in a region of attraction to converge to a global minimum point or a stationary point. Frequently, these attraction regions are small, and it is hard to give a suitable starting point.
(ii) It is a difficult task when to call a local optimization algorithm in a global optimization method. Ideally, the local optimization algorithm should only be called when an approximation of a global minimum point will actually be computed.

Our scheme attempts to avoid these difficulties by using inclusion functions for improving starting points and by incorporating a special scheme for calling a local optimization algorithm. Especially, it becomes apparent that local optimization methods and interval methods benefit from each other.

For the remainder of this section, we assume that an interval evaluation $f([x])$ of f is given, and $\underline{f}^*$, $\overline{f}^*$ denote a lower and upper bound of the global minimum value f^*. Moreover, let $p : \{1, \ldots, n\} \to \{1, \ldots n\}$ be a permutation and $n_{it}, n_d \in \mathbb{N} \setminus \{0\}$.

In the following, the quantities $f([x])$, p, n_{it}, and n_d are global quantities, and they will not be mentioned explicitely in the lists of parameters of our algorithms.

Our method consists of three algorithms. The first algorithm called BRANCH AND BOUND manages the partitioning of the feasible domain which is stored on a list S, and calls the main part of our method: procedure SUBDIVISION. Additionally, BRANCH AND BOUND contains some termination criteria, and updates the lower bound $\underline{f}^*$.

Procedure SUBDIVISION is responsable for discarding subregions which do not contain a global minimum point. Furthermore, SUBDIVISION improves starting points for the local optimization algorithm. One main idea here is to bisect always the box with the smaller

lower bound which is calculated with an inclusion function, and to store the remaining boxes on a "working list" W. This is done until we reach a certain number ($k_{\max} = n \cdot n_d$) of bisections. Bisecting in this way yields better starting points. Then SUBDIVISION calls the third algorithm SEARCH. If some conditions are satisfied, then in SEARCH a local optimization algorithm is started where the starting point is the midpoint of the box bisected last, the calculated approximation is stored on list A, and the upper bound $\overline{f}^*$ is updated. After finishing SEARCH, SUBDIVISION continues with bisecting boxes from list W, and discards those boxes $[y]$ whose corresponding lower bound $\underline{f}([y])$ is greater than the update of $\overline{f}^*$. All other boxes are stored on a list L. Then list L is returned to algorithm BRANCH AND BOUND and added to list S. Hence, boxes which contain global minimum points are never deleted.

First,the problem is initialized in (2.2) and (2.3) of algorithm BRANCH AND BOUND where the bounds $\underline{f}^*$ and $\overline{f}^*$ are set to $-\infty$ and ∞, respectively. List S contains the original box X, and list A (which will contain the approximations calculated by the local optimization algorithm) is empty. By passing through steps (2.4) to (2.11) we see that at most n_{it} iteration steps are executed. In each iteration step for all pairs of S' procedure SUBDIVISION is applied, in (2.10) the lower bound $\underline{f}^*$ is updated, and in (2.11) some additional termination criteria may be added.

algorithm BRANCH AND BOUND $(X, S, A, \underline{f}^*, \overline{f}^*)$
begin

$[y]^{(1)} := X, \quad \underline{f}^* := -\infty, \quad \overline{f}^* := \infty, \quad f([y]^{(1)}) := [-\infty, \infty];$ (2.2)

initialize lists $S := \left\{ ([y]^{(1)},\ f([y]^{(1)})) \right\}, \quad A := \emptyset;$ (2.3)

for $i = 1, \ldots, n_{it}$ **do** (2.4)

begin

select $S' \subseteq S$ such that the pair $([y], f([y])) \in S$ (2.5)
which has the smallest lower bound $\underline{f}([y])$ is contained in S';

set $S := S \setminus S'$; (2.6)

for all pairs $\left([y]^{(j)}, f([y]^{(j)})\right) \in S'$ **do** (2.7)

begin

call SUBDIVISION $\left([y]^{(j)}, f([y]^{(j)}), L, \overline{f}^*, A\right)$; (2.8)

append list L at the end of list S; (2.9)

end;

$\underline{f}^* := \mathrm{Max}\left\{ \underline{f}^*, \mathrm{Min}\{ \underline{f}([y]) \mid ([y], f([y])) \in S \} \right\};$ (2.10)

if some termination criteria hold **then** STOP; (2.11)

end;

end;

In (2.5) there are two extreme strategies how to define S'. First, we may choose always $S' = S$. This version corresponds to a *breadth-first search* of the branching process.

Another strategy is to choose only the pair $\Big([y], f([y])\Big) \in S$ which has the smallest lower bound $\underline{f}([y])$. This version corresponds to a *depth-first search.*

Because only n_{it} iteration steps are executed, the method terminates after a finite number of bisections. There are several possibilities to add some other termination criteria. We prefer a criterion that uses the relative error of the lower und upper bound, that is we may terminate our method if the conditions

$$\overline{f}^* - \underline{f}^* < \varepsilon_0, \quad \text{if} \quad 0 \in [\underline{f}^*, \overline{f}^*], \tag{2.12}$$

$$\left|(\overline{f}^* - \underline{f}^*) \,/\, \overline{f}^*\right| < \varepsilon_1, \quad \text{if} \quad 0 \notin [\underline{f}^*, \overline{f}^*] \tag{2.13}$$

hold where $\varepsilon_0, \varepsilon_1 > 0$.

In BRANCH AND BOUND no bisections are performed and no subregions are discarded. This will be done in SUBDIVISION. Here, we use a cyclic bisection rule which works as follows: first the coordinate with index $p(1)$ is bisected, then the coordinate with index $p(2)$ is bisected, ..., then the coordinate with index $p(n)$ is bisected; and this process is repeated n_d times. Hence, every box which is not discarded in SUBDIVISION results from at most $k_{\max} = n \cdot n_d$ bisections from the initial box $[y]$. A consequence is that all boxes which are contained in list L have a width equal to $w([y]) \,/\, 2^{n_d}$.

procedure SUBDIVISION $([y], f([y])), L, \overline{f}^*, A)$

begin

$k_{\max} := n \cdot n_d$; (2.14)

$k([y]) := 0$; (2.15)

initialize list $W := \Big\{([y], f([y]), k([y]))\Big\}$ and list $L := \emptyset$; (2.16)

while $W \neq \emptyset$ **do** (2.17)

begin

remove the last triple $\Big([y], f([y]), k([y])\Big)$ from list W; (2.18)

for $k = \Big(k([y]) + 1\Big), \ldots, k_{\max}$ **do** (2.19)

begin

if $\underline{f}([y]) > \overline{f}^*$ **then exit for loop**; (2.20)

$s := (k \bmod n) + 1$; (2.21)

bisect $[y]$ normal to direction $p(s)$ getting (2.22)

two boxes $[y]^{(1)}$, $[y]^{(2)}$ with $[y]^{(1)} \cup [y]^{(2)} = [y]$;

calculate $f([y]^{(1)})$, $f([y]^{(2)})$; (2.23)

if $\underline{f}([y]^{(1)}) > \underline{f}([y]^{(2)})$ **then** (2.24)

exchange the indices of $\Big([y]^{(1)}, f([y]^{(1)})\Big)$, $\Big([y]^{(2)}, f([y]^{(2)})\Big)$; (2.25)

$[y] := [y]^{(1)}$; $f([y]) := f([y]^{(1)})$; (2.26)

if $k < k_{\max}$ **and** $\underline{f}([y]^{(2)}) \leq \overline{f}^*$ **then** (2.27)

append the triple $\left([y]^{(2)}, f([y]^{(2)}), k\right)$ to list W; (2.28)

if $k = k_{\max}$ **and** $L = \emptyset$ **and** $\underline{f}([y]) \leq \overline{f}^*$ **then** (2.29)

call SEARCH $([y], A, \overline{f}^*)$; (2.30)

for $j = 1, 2$ **do**

begin

if $k = k_{\max}$ **and** $\underline{f}([y]^{(j)}) \leq \overline{f}^*$ **then** (2.31)

append $\left([y]^{(j)}, f([y]^{(j)})\right)$ to list L; (2.32)

end;

end;

end;

end;

In (2.16) a "working list" W is initialized which stores box $[y]$, bounds $f([y])$, and $k([y])$. The integer $k([y])$ gives the index of the last bisected coordinate of $[y]$.

At the beginning $k([y]) = 0$, which means no coordinate of $[y]$ has been bisected. Later on, the index of the coordinate which must be bisected is given by (2.21).

By passing through steps (2.17) to (2.32) we see that only in the case $\underline{f}([y]) > \overline{f}^*$ a pair $\left([y], f([y])\right)$ is discarded. Otherwise, this pair is entered in list L or list W (cf. (2.20),(2.27), (2.28), (2.31), (2.32)). Obviously, list W contains at most $k_{\max}$ triples. Therefore, in our numerical experiments we display the maximal length of the lists S, L, and A.

Moreover, looking at steps (2.19), (2.24), (2.25), (2.26), it follows that we always bisect the box with the smaller lower bound in order to move to an attraction region. Then in (2.29) and (2.30) procedure SEARCH is called. It is easy to see that list W contains at most $k_{\max}$ triples. Notice, that because L is empty, SEARCH is only called for that box which is at first bisected $k_{\max}$ times. All other boxes which are bisected $k_{\max}$ times are discarded or entered in list L (cf. (2.32)).

Steps (2.24), (2.25), and (2.26) form a rule which selects the box with the smaller lower bound. In the rare case where both bounds are equal, the algorithm proceeds with the first box. This case can be improved by inserting the following step after (2.23):

> **if** $\underline{f}([y]^{(1)}) = \underline{f}([y]^{(2)})$ then bisect $[y]$ normal to direction $p(s)$ yielding two boxes $[y]^{(1)}$, $[y]^{(2)}$ with $[y]^{(1)} \cup [y]^{(2)} = [y]$ such that $w([y_{p(s)}]^{(1)}) = 0.49 \cdot w([y_{p(s)}]^{(2)})$, then calculate $f([y]^{(1)}), f([y]^{(2)})$;

This rule serves to accelerate our method especially, if a global minimum is close to or on the common boundary of $[y]^{(1)}$ and $[y]^{(2)}$.

One central problem is to decide of when to call a local optimization algorithm. If each call of procedure SEARCH would imply a call of the local optimization algorithm the computational costs would grow dramatically. Therefore, in SEARCH some decision rules are incorporated. Additionally, we assume here that parameters $\alpha, \beta, \gamma, \delta \geq 0$ are given.

procedure SEARCH $([y], A, \overline{f}^*)$
begin
  $\widetilde{y} := m([y])$; (2.33)
  $\widetilde{f} := f(\widetilde{y})$; (2.34)
  if $((\widetilde{f} < \overline{f}^*)$ **or** $\Big(([y] \cap v(\widetilde{x})) = \emptyset$ for each already calculated (2.35)
    approximate local or global minimum point $\widetilde{x}$ stored
    in list A **and** $\widetilde{f} < \overline{f}^* + \delta \cdot |\overline{f}^*|)\Big)$ **then**
  begin
    call a local optimization algorithm with starting point $\widetilde{y}$ (2.36)
    calculating an approximation $\widetilde{x}$;
    if $\widetilde{x} \notin [x]$ **then** (2.37)
      begin
        if $\widetilde{x}_i < \underline{x}_i$ **then** $\widetilde{x}_i := \underline{x}_i$; (2.38)
        if $\widetilde{x}_i > \overline{x}_i$ **then** $\widetilde{x}_i := \overline{x}_i$; (2.39)
      end;
    $\widetilde{f} := f(\widetilde{x})$; (2.40)
    $\overline{f}^* := \mathrm{Min}\{\widetilde{f}, \overline{f}^*\}$; (2.41)
    $v(\widetilde{x}) := \widetilde{x} \pm \alpha \cdot \mathrm{Max}\{|\widetilde{x} - \widetilde{y}|, \beta|\widetilde{x}|, \gamma\}$; (2.42)
    append the triple $(\widetilde{x}, \widetilde{f}, v(\widetilde{x}))$ at the end of list A; (2.43)
  end;
end;

In (2.33), (2.34) the function value of the midpoint of box $[y]$ is calculated. In (2.35) two cases occur, both of which imply the call of a local optimization algorithm. In the first case when $\widetilde{f} < \overline{f}^*$, this call gives, in general, a better approximation of f^*. In our experience, using only this decision rule has the disadvantage that many bisections have to be performed in situations where the first call of the local optimization algorithm delivers only a stationary point or a local minimum $\widetilde{x}$. This is because the next call of the local optimization algorithm needs a box such that the function value of the midpoint is smaller than the function value of the previously calculated local minimum. Therefore, the local optimization algorithm is also called in the case where $\widetilde{f} < \overline{f}^* + \delta|\overline{f}^*|$ and box $[y]$ has an empty intersection with boxes $v(\widetilde{x})$ of previously calculated stationary points or local minima $\widetilde{x}$. Notice that by choosing $\delta := 0$ the local optimization algorithm is called only if $\widetilde{f} < \overline{f}^*$. In (2.42) the box $v(\widetilde{x})$ is defined by means of an expansion around $\widetilde{x}$. Obviously, this expansion depends on the difference between $\widetilde{x}$ and the starting point $\widetilde{y}$, on the absolute value of $\widetilde{x}$, and on a constant γ giving an expansion in situations where

some of the components of $\tilde{x}$ are close to zero. We emphasize that, in a test set of over 50 test problems (cf. Jansson and Knüppel 1992 [32]), in almost all situations the number of calls of the local optimization algorithm varies between the number of global minimum points and twice this number.

In rare cases the approximation $\tilde{x}$ is not feasible. Therefore, in steps (2.37), (2.38), and (2.39) the infeasible coordinates are projected onto the bounds of the set of feasible solutions X. Then, in (2.41), we update the upper bound $\overline{f}^*$, and in (2.43) the calculated approximation $\tilde{x}$ together with the corresponding function value $\tilde{f}$ and box $v(\tilde{x})$ are stored on list A.

Finally, one point for the implementation should be mentioned. So far we have ignored the effect of rounding errors. On a computer where they are present it is important to calculate a guaranteed upper bound of $f(\tilde{x})$ in (2.40). Then we undoubtly know that, due to rounding errors, we cannot loose any global minimum point. All the other calculations in SEARCH are executed on a computer by using floating point arithmetic.

An immediate consequence of our previous discussion is the following theorem:

Theorem 2.1. Let $f : X \to \mathbb{R}$, and assume further that

a) $f([x])$ is an inclusion function such that $f([y])$ contains the range $\{ f(y) \mid y \in [y] \}$ for each box $[y] \subseteq X$ which is representable on a computer;
b) $n_{it}, n_d \in \mathbb{N} \setminus \{0\}$ and $\alpha, \beta, \gamma, \delta, \varepsilon_0, \varepsilon_1 \geq 0$;
c) the local optimization algorithm terminates after a finite number of steps.

Then the following results hold:

(i) Algorithm BRANCH AND BOUND terminates after a finite number of steps;
(ii) $\underline{f}^* \leq f^* \leq \overline{f}^*$;
(iii) List A contains an approximation $\tilde{x}$ such that $f^* \leq f(\tilde{x}) \leq \overline{f}^*$;
(iv) $X^* \subseteq \bigcup \{ [y] \mid \big([y], f([y])\big) \in S \}$;

Proof. Since the parameter n_{it}, n_d bound the number of bisections, assumption c) yields (i).

The assertions (ii), (iv) follow by formulae (2.10), (2.41), and the fact that boxes which contain a global minimum point are not discarded.

Using (2.40) to (2.43) and noticing that $\tilde{x} \in X$, assertion (iii) follows. ■

2.2. Convergence

In the following we state and prove some convergence results of the method described in Section 2.1.

We assume that BRANCH AND BOUND is executed by using the real number system; that is, the effect of rounding errors is ignored for the following analysis.

The quantities that are defined in Section (2.1) depend mainly on the iteration steps $i = 1, \ldots, n_{it}$. In the following we write $\underline{f}^*(i)$, $\overline{f}^*(i)$, $S(i)$, $A(i)$ to indicate this dependence on i. Moreover, let

$$U(i) := \bigcup \left\{ [y]^{(j)} \mid \left([y]^{(j)}, f([y]^{(j)})\right) \in S(i) \right\}.$$

Theorem 2.2. Let $f : X \to \mathbb{R}$ be continuous, and suppose further that

a) $f([x])$ is an inclusion function with the following property: for each $x \in X$ and for each sequence $([z]^{(k)})$ with $[z]^{(k)} \to x$ and $[z]^{(k)} \subseteq X$, it follows that $\underline{f}([z]^{(k)}) \to f(x)$;
b) the local optimization algorithm is locally convergent for all $x^* \in \overline{X}^*$; that is, there is a neighbourhood $N(x^*)$ such that this algorithm converges to x^* for each starting point $x \in N(x^*)$, and calculates x^*;
c) $n_d \in \mathbb{N} / \{0\}$ and $n_{it} := \infty$ (that is, we want to study the asymptotic behaviour of our method for $i \to \infty$);
d) $\varepsilon_0 := 0$, $\varepsilon_1 := 0$ (that is, step (2.11) contains no additional termination criteria);
e) $\alpha, \beta, \gamma, \delta \geq 0$;
f) In step (2.5) set $S' := S$;

Then the following results hold:

(i) $\underline{f}^*(i)$ is monotonically increasing, and $\overline{f}^*(i)$ is monotonically decreasing for increasing i;
(ii) there exists an $i_0 \in \mathbb{N}$ such that list $A(i)$ contains a global minimum point $x^* \in X^*$ and $\overline{f}^*(i) = f^*$ for all $i \geq i_0$;
(iii) let $f^* \neq 0$, $\delta > 0$ and $\alpha = \beta = \gamma = 0$. If the global minimum points are isolated then there exists an $i_1 \in \mathbb{N}$ such that all global minimum points $x^* \in X$ are contained in all lists $A(i)$ for $i \geq i_1$;
(iv) $\lim_{i \to \infty} \underline{f}^*(i) = f^*$;
(v) $X^* = \bigcap_{i=1}^{\infty} U(i)$.

Proof.

(i) Formulae (2.10) and (2.41) imply that

$$\underline{f}^*(i+1) = \mathrm{Max}\Big\{\underline{f}^*(i),\ \mathrm{Min}\big\{\,\underline{f}([y]) \mid ([y], f([y])) \in S(i)\,\big\}\Big\}$$
$$\overline{f}^*(i+1) = \mathrm{Min}\Big\{\overline{f}^*(i), \widetilde{f}\Big\}.$$

Hence, these bounds monotonically increase and decrease respectively.

(ii) Because all boxes which are contained in a list $L(i;[y])$ have width $w([y])/2^{n_d}$ from assumption f) it follows that the width of the boxes contained in list $S(i)$ converge to zero as $i \to \infty$.

Since only boxes which contain no global minimum points are discarded, for each global minimum point $x^* \in X^*$ there exists boxes $[y]^{(j_1)}$ such that

$$x^* \in [y]^{(j_1)} \subseteq N(x^*),\ j_1 \in \mathbb{N},\ \Big([y]^{(j_1)}, f\big([y]^{(j_1)}\big)\Big) \in S(j_1).$$

To prove by contradiction that there exists an $i_0 \in \mathbb{N}$ such that $\overline{f}^*(i_0) = f^*$, we assume the contrary. By $(x^{(k)})$ we denote the finite or infinite sequence of points that are calculated by the local optimization algorithm and stored in $A(i)$. Two cases occur both of which imply a contradiction:

Case 1: There exists an $\varepsilon > 0$ such that $f(x^{(k)}) \geq f^* + \varepsilon$ for all k. Then $\overline{f}^*(i) \geq f^* + \varepsilon$ for all i.

Let $x^* \in X^*$. Since f is continuous and the width of the boxes contained in $S(i)$ converge to zero, there exists a box $[y]^{(j_2)}$, $j_2 \geq j_1$ such that

$$x^* \in [y]^{(j_2)} \subseteq [y]^{(j_1)}, ([y]^{(j_2)}, f([y]^{(j_2)})) \subseteq S(j_2), \text{ and}$$
$$f\Big(m\big([y]^{(j_2)}\big)\Big) < f^* + \varepsilon \leq \overline{f}^*(i) \text{ for all } i.$$

Thus, condition (2.35) is satisfied, and the local optimization algorithm is called with a starting point that is contained in $[y]^{(j_2)} \subseteq N(x^*)$. Now using assumption b) and setting $i_0 := j_2 + 1$ it follows that x^* is calculated by the local optimization algorithm and is contained in list $A(i_0)$. This gives the desired contradiction and shows that $\overline{f}^*(i_0) = f^*$.

Case 2: $f(x^{(k)}) > f^*$ for all k and $f(x^{(k)}) \to f^*$ as $k \to \infty$. Because the sequence $(x^{(k)})$ is bounded by X, there exists an accumulation point $\overline{x} \in X$ with $x^{(k_j)} \to \overline{x}$. Because f is continuous $f(\overline{x}) = f^*$, and therefore $\overline{x} \in X^*$. By assumption b) there is a neighbourhood $N(\overline{x})$ such that the local optimization algorithm converges for each starting point in $N(\overline{x})$. This contradicts the fact that the local optimization algorithm has calculated the points $x^{(k_j)}$ and $x^{(k_j)} \to \overline{x}$. Hence $\overline{f}(i_0) = f^*$ for some $i_0 \in \mathbb{N}$.

(iii) $\alpha = \beta = \gamma = 0$ yields $v(\tilde{x}) = \{\tilde{x}\}$ for all $\tilde{x}$ that are calculated by the local optimization algorithm. Let $x^* \in X^*$ be not contained in list $A(i_0)$. With the same arguments as in (ii) it follows that there exists a $[y]^{(i_1)}$ with the following properties:

$$x^* \in [y]^{(i_1)}, \ \left([y]^{(i_1)}, f([y]^{(i_1)})\right) \in S(i_1), \ [y]^{(i_1)} \subseteq N(x^*),$$

and because x^* is isolated (that is, there exists a neighbourhood of x^* such that x^* is the unique local and global optimum point in this neighbourhood)

$$[y]^{(i_1)} \cap v(\tilde{x}) = \emptyset$$

for all $\tilde{x}$ contained in $A(i_1)$, and since $\delta \cdot |f^*| > 0$

$$f\left(\mathrm{mid}([y]^{(i_1)})\right) < \overline{f}^* + \delta \cdot |f^*|.$$

Therefore, condition (2.35) is satisfied, and the local optimization algorithm calculates x^*.

(iv) Assume that $\lim\limits_{i\to\infty} \underline{f}^*(i) \neq f^*$. Because $(\underline{f}^*(i))$ is monotonically increasing and bounded by f^*, it follows that

$$\lim_{i\to\infty} \underline{f}^*(i) = \underline{f}^* < f^*.$$

Then (2.10) implies that there exists a infinite sequence of boxes $([y]^{(i)})$ with $\underline{f}([y]^{(i)}) \leq \underline{f}^*$ and $\left([y]^{(i)}, f([y]^{(i)})\right) \in S(i)$. Because this sequence of boxes is bounded by X and the width $w([y]^{(i)}) \to 0$ there exists a subsequence $([y]^{(i_j)})$ with $[y]^{(i_j)} \to y \in X$. It follows from assumption a) that $\underline{f}([y]^{(i_j)}) \to f(y) > \underline{f}^*$. This gives the desired contradiction.

(v) Boxes which contain a global minimum point are not discarded. Therefore $X^* \subseteq \bigcap\limits_{i=1}^{\infty} U(i)$. Let $\hat{x} \in \bigcap\limits_{i=1}^{\infty} U(i)$. Then there is a sequence $[y]^{(i)} \subseteq U(i)$ with $\hat{x} \in [y]^{(i)}$ and $w([y]^{(i)}) \to 0$ with $i \to \infty$. Now using assumption a) yields $\underline{f}([y]^{(i)}) \to f(\hat{x}) \geq f^*$. Since by (ii), we have $\overline{f}^*(i) = f^*$ for $i \geq i_0$, and because $[y]^{(i)}$ is not discarded, it follows that $\underline{f}([y]^{(i)}) \leq f^*$. Hence, $f(\hat{x}) = f^*$ and $\hat{x} \in X^*$. ■

Statement (ii) of Theorem 2.2 shows that, theoretically, after a *finite number* of steps the global minimum value f^* and at least one global minimum point x^* are calculated. As far we know, a similar finite convergence property using only such weak assumptions is not proved for other branch and bound methods. Notice that we have no assumptions like Lipschitz continuity or differentiability. From the proof it follows that no assumptions about the quality of the inclusion function are used. This is because the local optimization

algorithm is called by using only function evaluations at real points. Our method typically computes f^* and a global minimum point x^* at the very beginning. This is demonstrated by many test problems (cf. [32]). At a first glance, assumption b) used for proving statement (ii) seems to be strong. But almost all local optimization algorithms are locally convergent for a wide class of problems, and many Newton-type methods show locally superlinear or quadratic convergence. Thus these methods compute approximations of x^* very fast and accurate, provided the starting point is in the corresponding attraction region. For many classes of problems a rough approximation may be improved by a few Newton-type steps, such that the accuracy is almost equal to the machine accuracy, that is we just ignore the effect of rounding errors in this analysis. Hence, at least from a theoretical point of view, we can assume that the approximations calculated on a computer are identical with stationary points or global minimizers.

Statement (iii) of Theorem 2.2 shows that after a *finite number of steps* all global minimum points are calculated. Because an addition of an arbitrary constant to the objective function does not change the global minimum points, the assumption $f^* \neq 0$ is not restrictive. Moreover, from the proof it follows that this result holds also in the case where $\alpha, \beta, \gamma > 0$ provided that the global optimum points are sufficiently separated. In allmost all cases, there exists exactly one global minimum point, and therefore this result seems to be more of theoretical interest. But it is also interesting from a practical point of view: our method may calculate some approximations of local minimizers $\widetilde{x}$ which satisfy $f(\widetilde{x}) < f^* + \delta \cdot |f^*|$. That is, local minimum points which are almost global minimal may be calculated, and are at the disposal of the user.

Statements (iv) and (v) show that the lower bound converges to f^* and the bounds given by the boxes stored in list $S(i)$ converges to X^*. Our method is originally designed for nondifferentiable problems or problems where derivatives are not available. In our experience the convergence to X^* is very slow and undesirable clusters of boxes around each global minimum point occur provided that no interval derivatives are calculated. This cluster problem seems to be typical for branch and bound methods, and is studied in the univariate case by Kearfott and Du 1993 [35].

In the literature it is pointed out that many practical problems are design problems (cf. Törn and Žilinskas 1989 [84], pages 7, 12). For those problems it is sufficient to calculate only a point $\widetilde{x}$ with $f(\widetilde{x}) \approx f^*$, and sharp bounds for X^* are not necessary. Thus the method may be terminated before such clusters appear.

Our method can be used in an interactive way. If the precision is not good enough we can successively increase n_{it}. And also the parameter n_d, α, β, γ, δ, may be changed interactively. This offers additionally a great flexibility in applying this method. Mainly, the parameter n_d is responsable for calculating starting points in a region of attraction. For many problems values of n_d between 2 and 4 are satisfactory. Higher values of n_d are suited, if the region of attraction is very small.

2.3. Numerical experiments

In this section some numerical experiments are described. Even though they are not very extensive, they give a good guide in answering such questions as how accurate are the bounds for different parameters n_{it}, n_d, or questions related to the computational costs and the storage requirements.

The algorithm described in Section 2.1 is implemented in PROFIL/BIAS [38], [39], a C++ class library supporting interval arithmetic. PROFIL/BIAS is available for SUN Sparc-Stations, IBM RISC Stations, HP 9000/700 series, and PC's. All the runs use IEEE double precision ($\sim$ 17 decimals) on a SUN Sparc-Station 1.

We perform our experiments on a set of 9 test problems. Problems 1–7 are well-known differentiable examples. Problems 8 and 9 are non-differentiable. We emphasize that for all examples the same set of parameters has been chosen:

$$\alpha := 0.2,\ \beta := 0.1,\ \gamma := 10^{-3},\ \delta := 0.2.$$

The parameters are not optimized w.r.t. this set of problems. For some problems we obtain better results by changing some of the parameters. In all examples the permutation p is choosen such that

$$w([x_{p(i)}]) \geq w([x_{p(j)}]) \text{ for } i < j,$$

where $[x] = X$ is the feasible domain. Hence, first the box is bisected normal to the direction with the largest width, then normal to the direction with the second largest width, and so on.

Because we want to show the behaviour of our method for increasing parameters n_{it}, n_d, we set $\varepsilon_0 := 0$ and $\varepsilon_1 := 0$, that is we disable the additional termination criteria (2.11). In (2.5) we choose the breadth-first search $S' := S$. In examples 1 to 8 interval arithmetic evaluations are used. In example 9 a variant of Lohner's method [45] is used for the computation of an inclusion function. In examples 1 to 7 and 9 we used the local optimization algorithm of Brent 1973 [10] which is a modification of Powell's algorithm. In example 8 we used a SQP method.

In the tables of this section, we use the following abbreviations:

n_M denotes the number of global minima found by the algorithm, where a dash means, that only a local minimum has been found;

n_L denotes the number of calls of the local optimization method (cf. (2.36));

l_s denotes the maximum of the maximal lengths of the lists S, L, A;

n_{rf} is the total number of real function calls;

n_{if} is the total number of inclusion function calls;

t_{STU} is the machine independent Standard Unit Time. The unit for STU is the time needed to perform 1000 calls of the Shekel Function No. 5 at (4,4,4,4). On a SUN Sparc-Station 1 one unit in standard time is about 0.25 s.

We emphasize that for all following test problems the computed approximations $\tilde{x}$ and $f(\tilde{x})$ of the global minimum agree with the global minimum x^* and f^* within at least six decimal digits. Moreover, the upper bound $\overline{f}^*$ also agrees with $f(\tilde{x})$ and f^* in at least six decimal digits. Therefore, we display in our tables only $\overline{f}^*$ rounded to six decimal digits; x^* is given in the description of the test problems.

Example 1: Branin function [84].

$$f_{BR}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10$$

$$-5 \le x_1 \le 10, \quad 0 \le x_2 \le 15.$$

This function has three global minimum points.

$$f^* = 0.397887, \quad x^* = \begin{cases} (-3.14159, & 12.27500) \\ (3.14159, & 2.27500) \\ (9.42478, & 2.47500) \end{cases}$$

n_{it}	n_d	$\underline{f}^*$	$\overline{f}^*$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
2	2	0.397887	0.397887	1	1	7	127	79	0.600
3	2	0.397887	0.397887	3	3	9	271	139	1.267
4	2	0.397887	0.397887	3	3	9	276	201	1.467
2	3	0.397887	0.397887	3	3	10	247	143	1.200
3	3	0.397887	0.397887	3	3	10	251	231	1.400
4	3	0.397887	0.397887	3	3	10	254	309	1.600
2	4	0.397887	0.397887	3	3	9	233	209	1.267

Example 2: Rosenbrock [23].

$$f_{RB}(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

$$-5 \le x_i \le 5, \quad i = 1,2$$
$$f^* = 0, \quad x^* = (1,1)$$

n_{it}	n_d	f^*	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
2	2	0	$1.36492 \cdot 10^{-22}$	1	1	4	101	31	0.267
3	2	0	$1.36492 \cdot 10^{-22}$	1	1	4	102	55	0.333
4	2	0	$1.36492 \cdot 10^{-22}$	1	1	4	104	77	0.400
2	3	0	$1.14764 \cdot 10^{-22}$	1	1	4	122	57	0.400
3	3	0	$1.14764 \cdot 10^{-22}$	1	1	4	124	89	0.467
2	4	0	$2.29523 \cdot 10^{-21}$	1	1	3	119	83	0.400

Example 3: Shekel functions [84].

$$f_{Sm}(x) = -\sum_{i=1}^{m} \frac{1}{(x-a_i)^T(x-a_i)+c_i}$$

$$0 \le x_j \le 10, \quad j = 1,\ldots,4$$

For $m = 10$ the minimum is:

$$f^* = -10.5364, \quad x^* = (4.00075,\ 4.00059,\ 3.99966,\ 3.99951)$$

n_{it}	n_d	f^*	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
2	2	−10.8593	−10.5364	1	1	1	171	39	0.933
3	2	−10.5918	−10.5364	1	1	1	172	55	1.067
4	2	−10.5501	−10.5364	1	1	1	173	93	1.267
2	3	−10.5918	−10.5364	1	1	1	144	55	0.800
3	3	−10.5447	−10.5364	1	1	15	145	123	1.267
4	3	−10.5376	−10.5364	1	1	92	148	803	5.667
2	4	−10.5501	−10.5364	1	1	1	86	93	0.933

Example 4: Levy No. 12, $n = 10$ [23].

$$f_{\text{levy1}}(x) = \sin^2\pi y_1 + \sum_{i=1}^{n-1}(y_i-1)^2(1+10\sin^2\pi y_{i+1}) + (y_n-1)^2$$

$$\text{with } y_i = 1 + (x_i-1)/4, \quad -10 \le x_i \le 10, \quad i = 1,\ldots,n$$

This function has about 10^{10} feasible local minima.

$$f^* = 0, \quad x^* = (1, \ldots, 1)$$

n_{it}	n_d	$\underline{f^*}$	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
2	2	0	$3.41336 \cdot 10^{-16}$	1	1	1	379	81	2.200
3	2	0	$3.41336 \cdot 10^{-16}$	1	1	1	380	121	2.600
4	2	0	$3.41336 \cdot 10^{-16}$	1	1	1	381	161	2.933
2	3	0	$1.49337 \cdot 10^{-12}$	1	1	1	146	121	1.667
3	3	0	$1.49337 \cdot 10^{-12}$	1	1	1	147	181	2.133
2	4	0	$4.93934 \cdot 10^{-21}$	1	1	1	345	161	2.933

Example 5: Griewank function [84], $n = 10$.

$$f_G(x) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos\frac{x_i}{\sqrt{i}} + 1$$

$$-600 \leq x_i \leq 600, \quad d = 4000,$$
$$f^* = 0, \quad x^* = (0, \ldots, 0)$$

n_{it}	n_d	$\underline{f^*}$	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
2	8	0	$1.31006 \cdot 10^{-14}$	1	1	1	135	341	2.667
2	10	0	$1.80300 \cdot 10^{-13}$	1	1	1	417	421	4.600
2	12	0	$3.25184 \cdot 10^{-13}$	1	1	1	400	501	5.133

Only for a few real methods results are known for this function. In Törn and Žilinskas [84] the results of two methods are given:

Method	n_M	n_{rf}	t_{STU}
Griewank (1981)	–	6600	–
Snyman, Fatti (1987)	1	23399	90

Both methods give no guarantee, and Griewank's method has calculated only a local minimum.

Example 6: Same as example 5, but $n = 50$.

n_{it}	n_d	$\underline{f^*}$	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
1	10	0	$1.14087 \cdot 10^{-12}$	1	1	1	7644	1101	110.333
2	10	0	$1.14087 \cdot 10^{-12}$	1	1	1	7645	2101	140.333
1	15	0	$2.25375 \cdot 10^{-14}$	1	1	1	743	1601	48.067

Example 7:

This example is taken from Moore, Hansen, and Leclerc [50]. The authors write:

"Our second example concerns a *real world* problem which arises in the field of chemistry. More specifically, chemists performing *photoelectron spectroscopy* collide photons with atoms or molecules. These collisions result in the ejection of photoelectrons. The chemist is left with a *photoelectron spectrum* which is a plot of the number of photoelectrons ejected as a function of the kinetic energy of the photoelectron. A typical spectrum consists of a number of overlapping peaks of various shapes and intensities. The chemist desires to resolve the individual peaks.

One method for isolating each peak attempts to 'fit' the spectrum as the sum of *peak functions*. Peak functions are functions of variables which convey information regarding the peak's *position, intensity, width, function type, and tail characteristics*. Various types of functions have been used for this purpose, but the most common are Gaussian and/or Lorentzian."

In [50] the authors construct a spectral curve as the sum of two Gaussian functions given by 81 points (x_i, y_i):

$$x_i = 4.0 + 0.1(i+1),\ i = 1, 2, \ldots, 81$$

$$y_i = a_1 e^{-(\frac{x_i - u_1}{s_1})^2} + a_2 e^{-(\frac{x_i - u_2}{s_2})^2}$$

$$a_1 = 130.89 \quad a_2 = 52.6$$

$$u_1 = 6.73 \quad u_2 = 9.342$$

$$s_1 = 1.2 \quad s_2 = 0.97$$

The goal was to recover the six parameters a_1, a_2, u_1, u_2, s_1, and s_2 by minimizing the function

$$f(a_1, a_2, u_1, u_2, s_1, s_2) = \sum_{i=1}^{81} \left(a_1 e^{-(\frac{x_i - u_1}{s_1})^2} + a_2 e^{-(\frac{x_i - u_2}{s_2})^2} - y_i \right)^2,$$

where range of the six parameters was defined as follows:

$$[a_1] = [130,\ 135], \quad [a_2] = [50,\ 55], \quad [u_1] = [6,\ 8], \quad [u_2] = [8,\ 10],$$
$$[s_1] = [1,\ 2], \quad [s_2] = [0.5,\ 1]$$

The method described in [50] uses interval derivatives in an extensive way. They obtained

the following guaranteed bounds for the global minimum point

$$
\begin{aligned}
a_1 &= [130.889999624668920,\ 130.890000237423440] \\
a_2 &= [52.5999994426222910,\ 52.6000003353821410] \\
u_1 &= [6.72999999580056230,\ 6.73000000523584680] \\
u_2 &= [9.34199999170696670,\ 9.34200000792551850] \\
s_1 &= [1.19999999502502950,\ 1.20000000672384770] \\
s_2 &= [0.96999998507893725,\ 0.97000001469388031]
\end{aligned}
$$

and for the global minimum value

$$[\underline{f^*}, \overline{f^*}] = [6.3015390640982946 \cdot 10^{-13},\ 9.9696829305332294 \cdot 10^{-11}].$$

For the guaranteed bounds of f^* given in [50] supposely there is a misprint, and the lower bound possibly should be negative.

The number of guaranteed decimal digits for the global minimum point varies between 7 and 9 and the time needed on a SUN SparcStation is reported as 109240 s.

Our method gives the following results:

n_{it}	n_d	$\underline{f^*}$	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
1	1	0	$2.72132 \cdot 10^{-20}$	1	1	8	256	53	12.4
2	1	0	$2.72132 \cdot 10^{-20}$	1	1	44	264	469	57.8
1	2	0	$8.92363 \cdot 10^{-17}$	1	1	44	208	469	52.8

The number of correct decimal digits of the approximation, calculated for $n_{it} = n_d = 1$, varies between 12 and 15:

$$
\begin{aligned}
\widetilde{a}_1 &= 130.8900000000426 \\
\widetilde{a}_2 &= 52.59999999998969 \\
\widetilde{u}_1 &= 6.730000000000008 \\
\widetilde{u}_2 &= 9.342000000000636 \\
\widetilde{s}_1 &= 1.199999999999803 \\
\widetilde{s}_2 &= 0.9700000000003587
\end{aligned}
$$

For other values of n_{it} and n_d the approximations are similar. For example in case $n_{it} = n_d = 1$, we see that the algorithm has calculated an approximation such that

$$f(\widetilde{a}_1, \ldots, \widetilde{s}_2) \in [\underline{f^*}, \overline{f^*}] = [0, 2.72132 \cdot 10^{-20}],$$

and the time needed on a SUN SparcStation 1 is about 3 seconds. The bounds calculated for the global minimum point are rough. Corresponding test results for a variant of this

method which additionally calculates sharp bounds for the global minimum points are given in Section 2.6.

This problem is a nonlinear least square problem. Such problems can be found in many applications in engineering science. Typically, a model (here the Gaussian functions) and some experimental data (here the y_i) are given. The problem is to determine some parameters such that the model fits the data. Because of experimental and measuring errors the values y_i are uncertain. Other uncertainties do not occur. Obviously for this type of problems it is important to verify that the calculated approximation $\widetilde{a}_1, \ldots, \widetilde{s}_2$ fit exactly the data $\widetilde{y}_i$ where the $\widetilde{y}_i$ are relatively perturbated, and the perturbation is inside the measuring error. That is, the approximation is the exact solution of this problem with slightly perturbated data $\widetilde{y}_i$. For the above approximation the least square error is

$$\overline{f^*} = 2.72132 \cdot 10^{-20},$$

the maximal absolute error is

$$\max_{i=1}^{81} \left| \widetilde{a}_1 e^{-(\frac{x_i - \widetilde{u}_1}{\widetilde{s}_1})^2} + \widetilde{a}_2 e^{-(\frac{x_i - \widetilde{u}_2}{\widetilde{s}_2})^2} - y_i \right| \leq 4.26610 \cdot 10^{-11},$$

and the maximal relative error is

$$\max_{i=1}^{81} \frac{\left| \widetilde{a}_1 e^{-(\frac{x_i - \widetilde{u}_1}{\widetilde{s}_1})^2} + \widetilde{a}_2 e^{-(\frac{x_i - \widetilde{u}_2}{\widetilde{s}_2})^2} - y_i \right|}{|y_i|} \leq 1.00880 \cdot 10^{-11}.$$

That is, for example, if the relative measuring error is greater than $1.00880 \cdot 10^{-11}$, then the approximation $\widetilde{a}_1, \ldots, \widetilde{s}_2$ fit exactly the perturbated $\widetilde{y}_i$ where the perturbation is inside the measuring error. In practical applications the measuring errors are significantly larger than $1.00880 \cdot 10^{-11}$. In our opinion, the calculation of sharp bounds for the global minimum point for this type of problems is therefore not necessary.

The original range of the six parameters seems to be small for this problem. Enlarging the range to

$$[a_1] = [0,\ 200],\ [a_2] = [0,\ 100],\ [u_1] = [0,\ 10],\ [u_2] = [0,\ 20],$$
$$[s_1] = [0.1,\ 5],\ [s_2] = [0.1,\ 5]$$

our method yields the following results:

n_{it}	n_d	$\underline{f^*}$	$\overline{f^*}$	n_M	n_L	l_s	n_{rf}	n_{if}	t_{STU}
1	1	0	$4.07219 \cdot 10^{-18}$	1	1	32	542	155	26.8
2	1	0	$4.07219 \cdot 10^{-18}$	1	1	215	554	1549	164.1

Example 8: This example has been taken from Charalambous and Bandler [11], where an approximation of a fourth-order system using a second-order model is searched. The fourth-order system has the transfer function

$$G(s) = \frac{(s+4)}{(s+1)(s^2+4s+8)(s+5)}.$$

The second-order model's transfer function is

$$H(s) = \frac{x_3}{(s+x_1)^2 + x_2^2},$$

where x_1, x_2, x_3 are the parameters of the model with $x_1, x_3 \in [0,1]$ and $x_2 \in [0.1, 1]$.

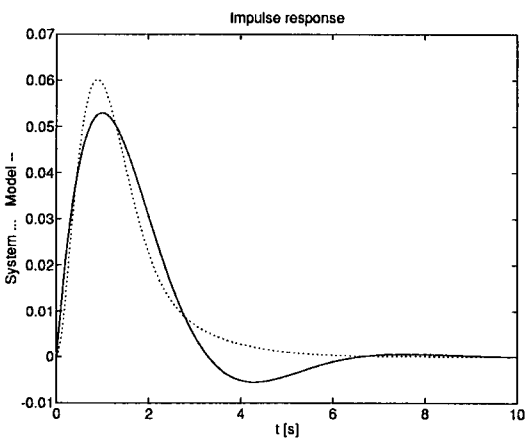
The impulse responses for the system and the model are:

$$\begin{aligned} s(t) &= \frac{3}{20}e^{-t} + \frac{1}{52}e^{-5t} - \frac{1}{65}e^{-2t}(3\sin 2t + 11\cos 2t) \\ h(x,t) &= \frac{x_3}{x_2}e^{-x_1 t}\sin x_2 t \end{aligned}$$

The impulse responses are compared at 51 equidistant time points t_i, $i = 0, \ldots, 50$ in the time from 0 to 10 s.

The aim is to find a set of the three parameters of the model such that the maximal error $f(x) := \max_i |s(t_i) - h(x,t_i)|$ is minimal.

The solution is $f^* = 0.00794706$ at $x_1^* = 0.684418$, $x_2^* = 0.954093$, and $x_3^* = 0.122864$. Plots of $s(t)$ and $h(x^*,t)$ are shown below, where the solid line is the impulse response of the model and the dotted line is the system's response.



The results by using our method are displayed in the following table.

n_{it}	n_{d}	$\underline{f^*}$	$\overline{f^*}$	n_{M}	n_{L}	l_{s}	n_{rf}	n_{if}	t_{STU}
2	2	0	0.00794706	1	2	283	236	1286	82.200
3	2	0.00631710	0.00794706	1	2	283	264	3836	224.267
4	2	0.00756954	0.00794706	1	2	283	293	7214	410.800
2	3	0.00631710	0.00794706	1	2	744	332	6046	332.800
3	3	0.00776569	0.00794706	1	2	744	344	11052	611.533
4	3	0.00791567	0.00794706	1	2	744	358	16232	899.333
2	4	0.00756954	0.00794706	1	2	471	346	7348	410.200

Example 9: In system analysis, a problem is to minimize the maximal real part of the eigenvalues of a matrix in order to get a maximal stable system. The systems discussed here consist of a matrix $M(x) \in \mathbb{R}^{m\times m}$ with parameters $x \in \mathbb{R}^2$. If $\lambda_i(x)$ are the eigenvalues of $M(x)$ and $\sigma(x) := \max_i \Re\{\lambda_i(x)\}$, then the aim is

$$\min_{x\in X} \sigma(x).$$

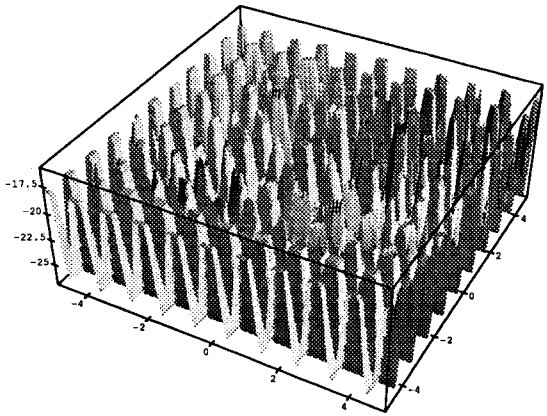
We consider the matrix

$$M(x) = \begin{pmatrix} d_1(x_1,x_2) & k\sin x_1 & k\sin x_2 & k\cos x_1 & k\cos x_2 \\ k\sin 2x_1 & d_2(x_1,x_2) & kx_1 & kx_2 & kx_1x_2 \\ k\sin 2x_2 & k(x_1+x_2) & d_3(x_1,x_2) & kx_1^2 & kx_2^2 \\ k\cos 2x_1 & k(x_1-x_2) & k(x_1+x_2)^2 & d_4(x_1,x_2) & k\sin x_1x_2 \\ k\cos 2x_2 & kx_1x_2^2 & k4x_2^2 & k\sin(x_1+x_2) & d_5(x_1,x_2) \end{pmatrix}$$

with

$$\begin{aligned} d_1(x_1,x_2) &= 17.5 - 2e^{-500\left((x_1+4)^2+(x_2+4)^2\right)} - \frac{x_1+x_2}{20} \\ d_2(x_1,x_2) &= 20 - \frac{x_1^2+x_2^2}{7} - (x_2+5)\cos\frac{\pi}{2}\left(x_1^2+x_2^2\right) \\ d_3(x_1,x_2) &= 20 - 6\cos 2\pi x_1 \\ d_4(x_1,x_2) &= 18 - \frac{x_1^4+x_2^4}{128} + \frac{1}{2}\cos 6\pi x_1x_2 \\ d_5(x_1,x_2) &= 20 - 6\cos 2\pi x_2 \\ k &= 10^{-3} \\ x_1,x_2 &\in [-5,5] \end{aligned}$$

As it can be seen in the plot of $-\sigma(x)$ (we turned it upside down to let the global minimum be visible as global maximum), the function $\sigma(x)$ contains lots of local minima and maxima.



The unique global minimum is

$$f^* = 15.9, \quad x^* = (-3.99997, -3.99997)$$

Applying our method by using a variant of Lohner's method [45] for calculating bounds for the eigenvalues, we obtain the results

n_{it}	n_{d}	$\underline{f^*}$	$\overline{f^*}$	n_{M}	n_{L}	l_{s}	n_{rf}	n_{if}	t_{STU}
2	2	15.8686	17.1890	–	2	44	815	114	144.3
2	4	15.8974	15.9000	1	1	1	133	33	21.5
3	4	15.8999	15.9000	1	1	1	134	49	28.5

2.4. An Expansion Scheme

Here, we introduce a new scheme which (i) proves existence and uniqueness of a stationary point in a small box that contains an approximation $\widetilde{x}$ calculated by a local optimization algorithm, and (ii) tries to verify uniqueness of this stationary point in a large box by expansion. This scheme yields an additional criterion for discarding subboxes. In the next section, it will be used to accelerate our branch and bound method, and to calculate very sharp bounds for the global minimum value **and** the global minimum points. We assume that inclusion functions of the first and second dervatives of f are available.

This approach uses the information of a stationary point or global minimum calculated with floating-point arithmetic, and is very much in the spirit of Wilkinson [88]. It increases the width of boxes where uniqueness can be verified, and it can be viewed contrary to the interval Newton method, which tries to decrease the width of the boxes and uses no approximations.

Theorem 2.3. Let $\sigma, \tau \in \mathbb{R}$ be positive, $\widetilde{x} \in \mathbb{R}^n$, $R \in \mathbb{R}^{n \times n}$, and let $w \in \mathbb{R}^n$ be defined componentwise by

$$w_i := \begin{cases} |\widetilde{x}_i| & \text{if} \quad |\widetilde{x}_i| \geq \tau \\ 1 & \text{otherwise} \end{cases} \tag{2.44}$$

for $i = 1, \ldots, n$. Moreover, assume that f is twice continuously differentiable, and let

$$z(\sigma) := |R \cdot f'(\widetilde{x})| + |I - R \cdot f''(\widetilde{x} + \sigma \cdot [-w, w])| \cdot \sigma \cdot w, \tag{2.45}$$

where f'' is an inclusion function of the Hessian of f.

If $z(\sigma) < \sigma \cdot w$, then R is nonsingular, and f contains exactly one stationary point $\widehat{x}$ with $f'(\widehat{x}) = 0$ in the box $\widetilde{x} + \sigma \cdot [-w, w]$. Moreover, $\widehat{x} \in \widetilde{x} + [-z(\sigma), z(\sigma)]$.

Proof. Let $W := \sigma \cdot [-w, w]$. Then $0 \in W$, and it follows from $z(\sigma) < \sigma \cdot w$ that

$$-R \cdot f'(\widetilde{x}) + \big(I - R \cdot f''(\widetilde{x} + W)\big) \cdot W \subsetneqq W.$$

Now, Theorem 2.3 is proved by using Theorem 7.4 of Rump (cf. [73], page 82), where f has to be replaced by f'. ■

Theorem 2.3 proves existence and uniqueness of a stationary point in a box symmetric to an approximation $\widetilde{x}$. Usually, R is an approximate inverse of the midpoint of $f''(\widetilde{x} + \sigma \cdot [-w, w])$. We mention that R may also be determined as an optimal preconditioner w.r.t. the inclusion function of the Hessian (cf. Kearfott 1990 [36]). Notice that no assumptions about the quality of $\widetilde{x}$ and R are required.

Our expansion scheme EXPAND $\big(\widetilde{x}, [e(\widetilde{x})], [u(\widetilde{x})]\big)$ depends on the parameters σ_0, σ_1, τ, and *expansion* which are not mentioned explicitly in the list of parameters of EXPAND. An appropriate choice is:

$$\sigma_0 = 10^{-5};\ \sigma_1 = 10^{-1};\ \tau = 10^{-3};\ \mathit{expansion} = 2.$$

Now, we use Theorem 2.3 in the following way. First, in steps (2.46) to (2.48) for $\sigma := \sigma_0$ we test the inequality $z(\sigma) < \sigma \cdot w$. If this inequality is not true, then we stop procedure EXPAND with a WARNING: existence and uniqueness cannot be verified. Otherwise, in the small box $[e(\widetilde{x})]$ (see (2.49)) around the approximation $\widetilde{x}$ existence and uniqueness of a stationary point is proved. Secondly, in steps (2.50) to (2.62) a large box is calculated where uniqueness of this stationary point is verified. We start with the larger box $\widetilde{x} + \sigma_1[-w, w]$. Then two cases occur:

In the first case (cf. (2.52) to (2.56)), uniqueness of the stationary point is verified, and we expand the box by multiplying σ with factor *expansion* (see (2.55)). This is done until $z(\sigma) \not< \sigma \cdot w$ — uniqueness cannot be proved —, or the set of feasible points $X = [x]$ is equal to $[u(\widetilde{x})]$. In the latter case, f has exactly one stationary point in $[x]$.

In the second case (cf. (2.57) to (2.61)), uniqueness of the stationary point is not proved, and we shrink the box by dividing σ with *expansion*. This is done until uniqueness can be proved, or $[u(\widetilde{x})]$ is contained in $[e(\widetilde{x})]$. In the latter situation (cf. (2.62)) both boxes coincide.

procedure EXPAND $(\widetilde{x}, [e(\widetilde{x})], [u(\widetilde{x})])$

begin

$\sigma := \sigma_0$; $[e(\widetilde{x})] := \emptyset$; $[u(\widetilde{x})] := \emptyset$; (2.46)

calculate w and $z(\sigma)$ according to (2.44) and (2.45); (2.47)

if $z(\sigma) \not< \sigma \cdot w$ **then STOP with WARNING** (2.48)

$[e(\widetilde{x})] := \widetilde{x} + [-z(\sigma), z(\sigma)]$; (2.49)

$\sigma := \sigma_1$; (2.50)

calculate $z(\sigma)$; (2.51)

if $z(\sigma) < \sigma \cdot w$ **then** (2.52)

while $z(\sigma) < \sigma \cdot w$ **and** $[u(\widetilde{x})] \subsetneqq [x]$ **do begin** (2.53)

$[u(\widetilde{x})] := (\widetilde{x} + \sigma \cdot [-w, w]) \cap [x]$; (2.54)

$\sigma := \sigma \cdot expansion$; (2.55)

calculate $z(\sigma)$; (2.56)

end;

else (2.57)

while $z(\sigma) \not< \sigma \cdot w$ **and** $[u(\widetilde{x})] \not\subseteq [e(\widetilde{x})]$ **do begin** (2.58)

$\sigma := \sigma / expansion$; (2.59)

calculate $z(\sigma)$; (2.60)

$[u(\widetilde{x})] := \widetilde{x} + \sigma \cdot [-w, w]$; (2.61)

end;

end;

if $[u(\widetilde{x})] \subseteq [e(\widetilde{x})]$ **then** $[u(\widetilde{x})] := [e(\widetilde{x})]$; (2.62)

end;

Summarizing, procedure EXPAND calculates a small box $[e(\widetilde{x})]$ where uniqueness and existence of a stationary point is guaranteed, and an expanded box $[u(\widetilde{x})]$. The latter box

$[u(\widetilde{x})]$ has the property that uniqueness cannot be proved (provided Theorem 2.3 is used) for the box which results from $[u(\widetilde{x})]$ by multiplying the width with factor *expansion*.

This procedure assumes that inclusion functions of the gradient and the Hessian are available. One way to calculate these inclusion functions is to perform the algebra of differentiation, for example with a program like MAPLE or MACSYMA, and then to calculate the corresponding natural interval evaluations. Another approach is *automatic differentiation* (cf. Rall 1981 [63], Griewank 1991 [20]). An important property of this technique is that gradients and Hessians can be obtained very cheaply.

There are various ways how to improve EXPAND. Here we mention only some of them:

• If EXPAND stops with a WARNING, then we may try the interval Newton method or the iteration scheme of Rump (see [73], Page 85).

• At a local or global unconstrained minimum of a twice differentiable function the Hessian must be positive semidefinite. A necessary condition for being positive semidefinite is that the diagonal elements of the Hessian are nonnegative. Therefore, if one of the diagonal elements of $f''(\widetilde{x}+\sigma\cdot[-w,w])$ is negative, then f is not convex in this box and EXPAND may be stopped. This may save some evaluations of the second derivative.

• Theorem 2.3 may be replaced by inclusion theorems for eigenvalue problems (cf. for example [2], [9] [45], [48], [78]) to prove that the Hessian is positive definite over a box around a calculated approximation. Then, f is strictly convex in the corresponding region, and the stationary point is guaranteed to be at least a local minimum.

• Theorems, similar to Theorem 2.3, may be used in procedure EXPAND where slopes (cf. Rump 1994 [72]) or Hansen's inclusion functions (cf. Hansen 1992 [23]) of the Hessian are involved. Moreover, in Rump 1994 [72] it is shown how regularity of an interval matrix can be proved by a special iteration. This iteration can be used to prove uniqueness of the expanded box.

At present, these improvements are not implemented in our code, and all test results given in Section 2.6 are obtained by using prodedure EXPAND in the simplified form (2.46) to (2.62).

2.5. A Branch and Bound Method Using Inclusion Functions of Derivatives

The branch and bound method described in Section 2.1 uses no derivatives, and subboxes $[y]$ of X are only discarded if a lower bound of f on $[y]$ is greater than $\overline{f}^*$. If interval evaluations of the first and second derivatives are available, in general, branch and bound schemes can be accelerated and the accuracy can be improved. The proposed techniques for some branch and bound methods are the monotonicity test, the nonconvexity test, and the interval Newton method (cf. Hansen 1992 [23], Ratschek and Rokne 1988 [65]). We do not use these techniques, and in the following we show how the expansion scheme

of Section 2.4 can be incorporated in our branch and bound scheme.

We have to modify the three procedures BRANCH AND BOUND, SUBDIVISION, and SEARCH. The most modifications are in procedure SEARCH. Therefore, the modified SEARCH is given completely below.

procedure SEARCH $([y], A, \overline{f}^*)$
begin
$\widetilde{y} := m([y])$; $\widetilde{f} := f(\widetilde{y})$; (2.63)
if $(\widetilde{f} < \overline{f}^*)$ **or** $(([y]) \cap [v(\widetilde{x})^{(i)}] = \emptyset$ **and** $[y] \cap [u(\widetilde{x}^{(i)})] = \emptyset$ for each (2.64)
already calculated approximate local or global minimum point $\widetilde{x}^{(i)}$
stored in list A **and** $\widetilde{f} < \overline{f}^* + \delta \cdot |\overline{f}^*|)$ **then begin**
call a local optimization algorithm with starting point $\widetilde{y}$ (2.65)
calculating an approximation $\widetilde{x}$;
if $(\widetilde{x} \in [x]$ **and** $\widetilde{x} \notin [u(\widetilde{x}^{(i)})]$ for each already calculated approximate
local or global minimum point $\widetilde{x}^{(i)}$ stored in list A)
then call EXPAND $(\widetilde{x}, [e(\widetilde{x})], [u(\widetilde{x})])$; (2.66)
else begin
if $\widetilde{x}_i < \underline{x}_i$ then $\widetilde{x}_i := \underline{x}_i$; (2.67)
if $\widetilde{x}_i > \overline{x}_i$ then $\widetilde{x}_i := \overline{x}_i$; (2.68)
$[e(\widetilde{x})] := \emptyset$; $[u(\widetilde{x})] := \emptyset$; (2.69)
end;
$\widetilde{f} := \overline{F}(\widetilde{x})$; $\overline{f}^* := \mathrm{Min}\{\widetilde{f}, \overline{f}^*\}$; (2.70)
$[v(\widetilde{x})] := \widetilde{x} + \alpha \cdot \mathrm{Max}\{|\widetilde{x} - \widetilde{y}|, \beta|\widetilde{x}|, \gamma\}$; (2.71)
append the tuple $(\widetilde{x}, \widetilde{f}, [v(\widetilde{x})], [u(\widetilde{x})], [e(\widetilde{x})])$ at (2.72)
the end of list A;
end;
end;

Obviously, the main modifications are in step (2.64) where the call of the local optimization algorithm is changed accordingly to the boxes $[u(\widetilde{x})]$, and in step (2.66) the call of EXPAND.

The most important point is that with the boxes $[u(\widetilde{x})]$ calculated by EXPAND we have an additional criterion to discard subboxes. If a subbox $[y]$ is contained in some expanded box $[u(\widetilde{x})]$, then $[y]$ can be discarded independently of the bound of f on $[y]$. This is, because $[u(\widetilde{x})]$ contains exactly one stationary point that is enclosed by $[e(\widetilde{x})]$ and these boxes are stored on list A. It follows that during execution of our algorithm list A contains very sharp inclusion boxes $[e(\widetilde{x})]$ of local or global minima, list S contains boxes which cannot be discarded by both criteria, and the set of global minimizers satisfies

$$X^* \subseteq \bigcup\{\,[y] \mid ([y], f([y])) \in S\,\} \cup \bigcup\{\,[e(\widetilde{x})] \mid [e(\widetilde{x})] \text{ is contained in } A\,\}. \tag{2.73}$$

Hence, if list S is empty, then X^* is contained in the union of the boxes $[e(\widetilde{x})]$, and

algorithm BRANCH AND BOUND can be stopped. Therefore, step (2.5) is replaced by

if $S = \emptyset$ **then begin**
$\underline{f}^* = \text{Min}\{\, \underline{f}([e(\tilde{x})] \mid [e(\tilde{x})] \text{ is contained in} A \,\}$;
STOP algorithm BRANCH AND BOUND;
end;
else select $S' \subseteq S$ such that the pair $([y], f([y])) \in S$,
which has the smallest lower bound $\underline{f}([y])$, is contained in S';

Since only SUBDIVISION is responsable for discarding subregions, we have the following modifications:

Step (2.20) is replaced by

if $(\underline{f}([y]) > \overline{f}^*$ **or** $[y] \subseteq [u(\tilde{x})]$ for an approximation $\tilde{x}$
already calculated and stored in list A) **then exit for loop**;

and steps (2.29), (2.30) are replaced by

if ($k = k_{\max}$ **and** $L = \emptyset$ **and** $\underline{f}([y]) \leq \overline{f}^*$ **and** $[y] \not\subseteq [u(\tilde{x})]$
for an approximation $\tilde{x}$ already calculated and stored in list A) **then**
call SEARCH($[y], A, \overline{f}^*$);

Usually, this algorithm terminates after a finite number of steps with $S = \emptyset$ provided that the Hessians of the global minimizers are nonsingular. Since the main modification is the additional criterion for discarding subboxes in SUBDIVISION together with storing the boxes $[e(\tilde{x})]$ on list A in SEARCH, it follows that the statements (i), (ii), (iii) of Theorem 2.1 are satisfied. Statement (iv) must be replaced by (2.73).

This modified method can be viewed as an accelerated version of the method described in Section 2.1 that additionally calculates very sharp bounds for the global minimum value **and** the global minimum points. If the algorithm finishes with $S = \emptyset$, and the precision of the bounds is not sufficient, then one or two interval Newton steps applied to the boxes $[e(\tilde{x})]$ of list A which satisfy $\underline{f}([e(\tilde{x})]) \leq \overline{f}^*$ may be added.

On the contrary to the well-known interval methods in global optimization which use inclusion functions of derivatives in an extensive way, in our method derivatives are used only in the case where approximations of stationary points are calculated. Thus, it is typical for our method that the number of calculations of derivatives is small compared to other interval methods. This is demonstrated in the report [33] where numerical results for many well-known test problems including the test set of Hansen [23] are described.

For some problems it is a priori known that they have exactly one stationary point which is the global minimum point. For example, strictly convex problems have this property. Obviously, in such cases our method can be stopped if the stationary point is verified by procedure EXPAND. Then mainly the branch part of our method serves to calculate a good starting point in the attraction region of the global minimum point. With this

modification sharp bounds for the introducing example that describes the optimization of the voltage stabilizer can be calculated.

For almost all test problems known to us (cf. [32], [33]) our branch and bound method works very efficiently. Only in three cases we were not so lucky. One of which was Mandel'shtam's problem (cf. [18]) where we had success only up to dimension 4, because we did not find an appropriate inclusion functions.

The efficiency depends very much on choosing an appropriate inclusion function. In many cases natural interval evaluations are sufficient. But we think that it is very important to find other types of inclusion functions for special problems.

2.6. Numerical Experiments

In this section we give numerical test results for the method described in Section 2.5. Here, we use the differentiable problems given in examples 1 to 7 of Section 2.3. We use the same set of parameters for all test problems:

$$\alpha := 0.2,\ \beta := 0.1,\ \gamma := 10^{-3},\ \sigma := 0.2,\ \tau := 10^{-3},\ \sigma_1 := 10^{-5},$$
$$\sigma_2 := 0.1,\ \mathit{expansion} := 2.$$

By adapting the parameters to specific test problems, better results may be obtained. For example, increasing the value of *expansion* reduces the number of calls of the Hessian, but increases the number of calls of the inclusion function.

Additionally to the notations given in Section 2.3, we use n_{ig}, n_{ih} for the total number of calls of the inclusion function for the gradient and the Hessian, respectively. We have chosen $n_{it} = 30$ for all test problems. In each case the algorithm terminates with an empty list S. It follows that for each larger value of n_{it} the algorithm terminates in the same manner.

Example 1: Branin function [84].

$$f^* \subseteq [0.3978873577297382, 0.3978873577297453]$$

$$x^* \subseteq \begin{pmatrix} [-3.141592653589795, -3.141592653589792] \\ [12.274999999999999, 12.27500000000001] \end{pmatrix} \cup \begin{pmatrix} [9.424777960769378, 9.424777960769383] \\ [2.474999999999999, 2.475000000000002] \end{pmatrix} \cup \begin{pmatrix} [3.141592653589793, 3.141592653589794] \\ [2.274999999999998, 2.275000000000001] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
2	3	3	7	233	127	3	18	2.30
3	3	3	7	228	131	3	18	2.20
4	3	3	7	247	150	3	18	2.40

Example 2: Rosenbrock [23].

$$f^* \subseteq [0, 8.799053144448318 \cdot 10^{-27}]$$

$$x^* \subseteq \begin{pmatrix} [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
2	1	1	4	124	150	1	13	1.45
3	1	1	4	132	152	1	13	1.45
4	1	1	3	119	156	1	13	1.50

Example 3: Shekel function, $m = 10$ [84].

$$f^* \subseteq [-10.53640981669206, -10.53640981669203]$$

$$x^* \subseteq \begin{pmatrix} [4.000746531592046, 4.000746531592047] \\ [4.000592934138531, 4.000592934138532] \\ [3.999663398040322, 3.999663398040322] \\ [3.999509800586807, 3.999509800586808] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
2	1	1	1	255	93	1	7	2.85
3	1	1	1	174	93	1	7	2.50
4	1	1	1	107	93	1	7	2.30

Example 4: Levy No. 12, $n = 10$ [23].

$$f^* \subseteq [0, 4.939341111267398 \cdot 10^{-21}]$$

$$x^* \subseteq \begin{pmatrix} [0.9999999999999998, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \\ [0.9999999999999999, 1] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
2	1	1	1	380	141	1	5	4.75
3	1	1	1	146	141	1	5	3.70
4	1	1	1	344	141	1	5	4.50

Example 5: Griewank function [84], $n = 10$.

$$f^* \subseteq [0, 4.551914400963142 \cdot 10^{-15}]$$

$$x^* \subseteq \begin{pmatrix} [-1.752875876867703 \cdot 10^{-15}, 9.10327788226753 \cdot 10^{-16}] \\ [-2.49108923735287 \cdot 10^{-15}, 1.273368348184274 \cdot 10^{-15}] \\ [-3.724949921605188 \cdot 10^{-15}, 8.832483289507194 \cdot 10^{-16}] \\ [-3.68400804229635 \cdot 10^{-15}, 1.634425727320358 \cdot 10^{-15}] \\ [-1.364848010712496 \cdot 10^{-23}, 1.282129949457193 \cdot 10^{-23}] \\ [-2.9778502051909 \cdot 10^{-23}, 2.646977960169689 \cdot 10^{-23}] \\ [-5.156808186946726 \cdot 10^{-15}, 1.868301654538232 \cdot 10^{-15}] \\ [-5.743642108837937 \cdot 10^{-15}, 1.762776141781241 \cdot 10^{-15}] \\ [-6.995533733778533 \cdot 10^{-15}, 9.622619705035564 \cdot 10^{-16}] \\ [-1.77843831698901 \cdot 10^{-23}, 1.613002194478404 \cdot 10^{-23}] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
10	1	1	1	416	261	1	7	9.25
15	1	1	1	163	322	1	7	8.60

Example 6: Same as example 5, but $n = 50$.

$$f^* \subseteq [0, 3.164135620181696 \cdot 10^{-14}]$$

$$x^* \subseteq \begin{pmatrix} [-1.804243928668264 \cdot 10^{-15}, 8.589598111826399 \cdot 10^{-16}] \\ \vdots \\ [-1.513044566938841 \cdot 10^{-14}, 3.251126042831056 \cdot 10^{-15}] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
15	1	1	1	767	1602	1	7	627.85
20	1	1	1	516	2102	1	7	729.15

Example 7: taken from [50].

$$f^* \subseteq [0, 3.678114804829197 \cdot 10^{-21}]$$

$$x^* \subseteq \begin{pmatrix} [130.8899999999997, 130.8900000000002] \\ [52.59999999999987, 52.60000000000012] \\ [6.729999999999998, 6.730000000000003] \\ [9.341999999999995, 9.342000000000006] \\ [1.199999999999996, 1.200000000000004] \\ [0.9699999999999923, 0.9700000000000075] \end{pmatrix}$$

n_d	n_M	n_L	l_s	n_{rf}	n_{if}	n_{ig}	n_{ih}	t_{STU}
1	1	1	153	487	19129	1	14	1688.00
2	1	1	164	270	19129	1	14	1681.65
3	1	1	142	296	19135	1	14	1681.90
4	1	1	151	295	19147	1	14	1687.35

2.7. Remarks

To our knowledge, Moore 1966 [51] was the first to use interval arithmetic for solving global optimization algorithms. We would like to stress that this work was truly pioneering. A combination of a branch and bound strategy with some of Moore's principles was given by Skelboe 1974 [81] and improved by Moore 1976 [52]. Some recent test results for the method of Moore and Skelboe are given in Csendes and Pintér 1993 [16]. A very important branch and bound method using extensively the tools of interval arithmetic like a specialized interval Newton method, monotonicity tests, nonconvexity test, etc. is presented by Hansen 1979, 1980, 1992 [21], [22], [23]. Some recent test results of a modification of Hansen's method are given by Ratz 1992 [66], [67], and a special branch and bound method using preconditioning techniques is given by Kearfott 1992 [37]. A special method for solving Minimax problems by using the tools of interval arithmetic has been proposed by Zuhe et al. 1990 [89]. Methods for solving nonlinear parameter estimation problems are described in Csendes 1988, [13], [14], [15]. A detailed convergence

analysis of some branch and bound methods that use interval arithmetic was first given by Ratschek 1985 [64]. For further details, see Ratschek and Rokne 1988 [65]. This book contains also about 290 references that are related to interval arithmetic, nonlinear systems and global optimization problems.

Other branch and bound schemes which are not closely related to interval arithmetic are described in the well-known textbooks of Horst and Tuy 1990 [24], Pardalos and Rosen 1987 [61], and Törn and Žilinskas 1989 [84].

For a first version of the method described in Section 2.1 see [27], [30]. A convergence analysis together with some numerical results and comparisons with other well-known global optimization methods is treated in [31]. Numerical results for a test set of about 50 problems is given in [32]. In these two publications it is shown that for many problems our algorithm has a better efficiency than the best traditional methods, the latter, however, do not deliver guaranteed results. Numerical results for the modified method described in Section 2.5 are given in [33]. Similar methods for nonlinear systems of equations are treated in Knüppel 1994 [40].

REFERENCES

1. G. Alefeld. Inclusion Methods for Systems of Nonlinear Equations. In J. Herzberger, editor, *Topics in Validated Computations — Studies in Computational Mathematics*, Amsterdam. North-Holland to appear.
2. G. Alefeld. Rigorous Error Bounds for Singular Values of a Matrix Using the Precise Scalar Product. In E. Kaucher, U. Kulisch, and Ch. Ullrich, editors, *Computerarithmetic.* Teubner Stuttgart, 1987.
3. G. Alefeld and J. Herzberger. *Einführung in die Intervallrechnung.* B.I. Wissenschaftsverlag, 1974.
4. G. Alefeld and J. Herzberger. *Introduction to Interval Computations.* Academic Press, New York, 1983.
5. G. Alefeld and L. Platzöder. A Quadratically Convergent Krawczyk-Like Algorithm. *SIAM Numer. Anal.*, 20(1):210–219, 1983.
6. H. Bauch, K.-U. Jahn, D. Oelschlägel, H. Süsse, and V. Wiebigke. *Intervallmathematik, Theorie und Anwendungen*, volume Bd. 72 of *Mathematisch-naturwissenschaftliche Bibliothek.* B.G. Teubner, Leipzig, 1987.
7. H. Becker and R. Wittmer. Parameteroptimierung. In *Texte des Modellversuchs zur mathematischen Weiterbildung*, volume Heft 1. Universität Kaiserslautern, 1983.
8. H. Beeck. Linear Programming with Inexact Data. Technical Report Bericht 7830, Abteilung Mathematik, TU München, 1978.
9. H. Behnke. The Determination of Guaranteed Bounds to Eigenvalues with the Use of Variational Methods II. In Ch. Ullrich, editor, *Computer Arithmetic and Self-Validating Numerical Methods*, pages 155–170. Academic Press, 1990.
10. R.P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1973.

11. C. Charalambous and J.W. Bandler. Non-linear Minimax Optimization as a Sequence of Least pth Optimization with Finite Values of *p. J. Comput. Syst. Sci.*, 7(4):377–391, 1976.
12. V. Chvatal. *Linear Programming.* W.H. Freemann and Company, 1983.
13. T. Csendes. Nonlinear Parameter Estimation by Global Optimization — Efficiency and Reliability. *Acta Cybernetica*, Tom 8(Fasc. 4):361–370, 1988.
14. T. Csendes. An Interval Method for Bounding Level Sets of Parameter Estimation Problems. *Computing*, 41:75–86, 1989.
15. T. Csendes. Interval Method for Bjounding Level Sets: Revisited and Tested with Global Optimization Problems. *BIT*, 30:650–657, 1990.
16. T. Csendes and J. Pintér. The Impact of Accelerating Tools on the Interval Subdivision Algorithm for Global Optimization. *European Journal of Operational Research*, 65:314–320, 1993.
17. G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, 1963.
18. V.F. Dem'yanov and V.N. Malozemov. *Introduction to MINIMAX.* John Wiley & Sons, 1974.
19. H.W. Engl and C.W. Groetsch. *Inverse and Ill-posed Problems.* Academic Press, Orlando, 1987.
20. A. Griewank and G.F. Corliss. *Automatic Differentiation of Algorithms. Theory, Implementation, and Applications.* SIAM, Philadelphia, 1991.
21. E.R. Hansen. Global Optimization Using Interval Analysis — the One Dimensional Case. *J. Optim. Theory Appl. 29*, pages 331–344, 1979.
22. E.R. Hansen. Global Optimization Using Interval Analysis — the Multidimensional Case. *Numer. Math. 34*, pages 247–270, 1980.
23. E.R. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, New York, Basel, Hong Kong, 1992.
24. R. Horst and H. Tuy. *Global Optimization.* Springer-Verlag, Berlin, 1990.
25. C. Jansson. *Zur linearen Optimierung mit unscharfen Daten.* Dissertation, Universität Kaiserslautern, 1985.
26. C. Jansson. A Self-Validating Method for Solving Linear Programming Problems. *Computing, Suppl. 6*, pages 33–46, 1988.
27. C. Jansson. A Global Minimization Method: The One-Dimensional Case. Technical Report 91.2, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg, 1991.
28. C. Jansson. Interval Linear Systems with Symmetric Matrices, Skew-Symmetric Matrices, and Dependencies in the Right Hand Side. *Computing 46*, pages 265–274, 1991.
29. C. Jansson. Rigorous Sensitivity Analysis for Real Symmetric Matrices with Uncertain Data. In E. Kaucher, S.M. Markov, and G. Mayer, editors, *Computer Arithmetic*, IMACS, pages 293–316. J.C. Baltzer AG Scientific Publishing Co, 1991.
30. C. Jansson. A Global Optimization Method Using Interval Arithmetic. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, IMACS, pages 259–267. Elsevier Science Publishers B.V., 1992.
31. C. Jansson and O. Knüppel. A Branch-and-Bound Algorithm for Bound Constrained Optimization Problems without Derivatives. To appear.

32. C. Jansson and O. Knüppel. A Global Minimization Method: The Multi-dimensional case. Technical Report 92.1, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg, 1992.
33. C. Jansson and O. Knüppel. Numerical results for a Self-Validating Global Optimization Method. Technical Report 94.1, Forschungsschwerpunkt Informations- und Kommunikationstechnik, TU Hamburg-Harburg, 1994.
34. C. Jansson and S.M. Rump. Rigorous Solution of Linear Programming Problems with Uncertain Data. *ZOR — Methods and Models of Operations Research 35*, pages 87–111, 1991.
35. B. Kearfott and K. Du. The Cluster Problem in Global Optimization. *Computing Suppl.*, (9):117–127, 1993.
36. R.B. Kearfott. Preconditioners for the Interval-Gauss-Seidel Method. *SIAM J. Numer. Anal.*, 27(3):804–822, 1990.
37. R.B. Kearfott. An Interval Branch and Bound Algorithm for Bound Constrained Optimization Problems. *Journ. of Glob. Opt.*, 2:259–280, 1992.
38. O. Knüppel. BIAS — Basic Interval Arithmetic Subroutines. Technical Report 93.3, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TUHH, 1993.
39. O. Knüppel. PROFIL — Programmer's Runtime Optimized Fast Interval Library. Technical Report 93.4, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, TUHH, 1993.
40. O. Knüppel. *Einschließungsmethoden zur Bestimmung der Nullstellen nichtlinearer Gleichungssysteme und ihre Implementierung.* Dissertation, Technische Universität Hamburg-Harburg, 1994.
41. T.C.T. Kotiah and D.I. Steinberg. Occurrencies of Cycling and other Phenomena Arising in a Class of Linear Programming Models. *Comm. ACM 20*, pages 107–112, 1977.
42. T.C.T. Kotiah and D.I. Steinberg. On the Possibility of Cycling with the Simplex Method. *Operations Research 26*, pages 374–376, 1978.
43. R. Krawczyk. Fehlerabschätzung bei linearer Optimierung. *Interval Mathematics*, 29, 1975.
44. U. Kulisch and W.L. Miranker. *Computer Arithmetic in Theory and Practice.* Academic Press, New York, 1981.
45. R. Lohner. Enclosing all Eigenvalues for Symmetric Matrices. In Ch. Ullrich and J. Wolff von Gudenberg, editors, *Accurate Numerical Algorithms*, pages 87–103, New York, 1989.
46. B. Machost. Numerische Behandlung des Simplexverfahrens mit intervallmathematischen Methoden. *Berichte der GMD Bonn 30*, 1970.
47. U. Maichle. Lineare Intervalloptimierung und Anwendungen. Diplomarbeit, Universität Kaiserslautern, 1988.
48. G. Mayer. Enclosures for Eigenvalues and Eigenvectors. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, IMACS. Elsevier Science Publisher B.V., 1992.
49. M. Minoux. *Mathematical Programming: Theory and Algorithms.* John Wiley & Sons, Chichester, 1986.
50. R. Moore, E. Hansen, and A. Leclerc. Rigorous Methods for Global Optimization. In *In Recent Advances in Global Optimization, Princeton series in computer science*, pages 321–

342, Princeton, New Jersey, 1992. Princeton University Press.
51. R.E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1966.
52. R.E. Moore. On Computing the Range of Values of a Rational Function of n Variables over a Bounded Region. *Computing 16*, pages 1–15, 1976.
53. R.E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.
54. F. Mraz. Solution Function of an Interval Linear Programming Problem. Technical Report 90-03, Danmarks Teknikse Hojskole, 1990.
55. B.A. Murtagh. *Advanced Linear Programming: Computation and Practice.* McGraw-Hill-Book Company, 1981.
56. K.G. Murty and S.N. Kabadi. Some NP-Complete Problems in Quadratic and Nonlinear Programming. *Mathematical Programming 39*, pages 117–130, 1987.
57. A. Neumaier. Overestimation in Linear Interval Equations. *SIAM J. Numer. Anal.*, 24(1):207–214, 1987.
58. A. Neumaier. Rigorous Sensitivity Analysis for Parameter-Dependent Systems of Equations. *J. Math. Anal. Appl. 144*, pages 16–25, 1989.
59. A. Neumaier. *Interval Methods for Systems of Equations, Encyclopedia of Mathematics and its Applications.* Cambridge University Press, 1990.
60. C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, N.J., 1982.
61. P.M. Pardalos and J.B. Rosen. Constrained Global Optimization: Algorithms and Applications. *Springer Lecture Notes Comp. Sci. 268, Berlin*, 1987.
62. PRO-MATLAB User's Guide, Vers. 32-SUN. *The MathWorks Inc.*, 1992.
63. L.B. Rall. Automatic Differentiation: Techniques and Applications. In *Lecture Notes in Computer Science 120.* Springer Verlag, Berlin-Heidelberg-New York, 1981.
64. H. Ratschek. Inclusion Functions and Global Optimization. *Mathematical Programming 33*, pages 300–317, 1985.
65. H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* John Wiley & Sons (Ellis Horwood Limited), New York (Chichester), 1988.
66. D. Ratz. An Inclusion Algorithm for Global Optimization in a Portable PASCAL-XSC Implementation. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, pages 329–339. Elsevier Science Publisher B. V., 1992.
67. D. Ratz. *Automatische Ergebnisverifikation bie globalen Optimierungsproblemen.* Dissertation, Universität Karlsruhe, 1992.
68. J. Rohn. Solving Interval Linear Systems. *Freiburger Intervallberichte 84/7,*, pages 1–58, 1984.
69. J. Rohn. Systems of Linear Interval Equations. *Linear Algebra Appl. 126*, pages 39–78, 1989.
70. J. Rohn. Stability of the Optimal Basis of a Linear Program under Uncertainty. *Operations Research Letters 13*, pages 9–12, 1993.
71. J. Rohn and V. Kreinovich. Computing Exact Componentwise Bounds on Solutions of Perturbated Linear Systems is NP-hard. Utep-cs-93-37, Computer Science Dept., University of Texas at El Paso, June 1993.
72. S.M. Rump. Verification Methods for Dense and Sparse Systems of Equations. In J. Herzberger, editor, *Topics in Validated Computations — Studies in Computational Mathe-*

matics, Amsterdam. North-Holland to appear.

73. S.M. Rump. Solving Algebraic Problems with High Accuracy. Habilitationsschrift. In U.W. Kulisch and W.L. Miranker, editors, *A New Approach to Scientific Computation*, pages 51–120. Academic Press, New York, 1983.
74. S.M. Rump. Rigorous Sensitivity Analysis for Systems of Linear and Nonlinear Equations. *Math. of Comp.*, 54(10):721–736, 1990.
75. S.M. Rump. Inclusion of the Solution for Large Linear Systems with M-Matrix. In L. Atanassova and J. Herzberger, editors, *Computer Arithmetic and Enclosure Methods*, pages 339–350. Elsevier Science Publisher B.V., 1992.
76. S.M. Rump. On the Solution of Interval Linear Systems. *Computing 47*, pages 337–353, 1992.
77. S.M. Rump. Validated Solution of Large Linear Systems. In R. Albrecht, G. Alefeld, and H.J. Stetter, editors, *Computing Supplementum 9, Validation Numerics*, pages 191–212. Springer, 1993.
78. S.M. Rump. Zur Außen- und Inneneinschließung von Eigenwerten bei toleranzbehafteten Matrizen. *ZAMM*, 73(718):T861 – T863, 1993.
79. H. Schwandt. An Interval Arithmetic Approach for the Construction of an almost Globally Convergent Method for the Solution of the Nonlinear Poisson Equation on the Unit Square. *SIAM J. Sci. Stat. Comp.*, 5(2):427–452, 1984.
80. H. Schwandt. Krawczyk-like Algorithms for the Solution of Systems of Nonlinear Equations. *SIAM J. Numer. Anal. 22*, pages 792–810, 1985.
81. S. Skelboe. Computation of Rational Interval Functions. *BIT 14*, pages 87–95, 1974.
82. R.E. Steuer. Algorithms for Linear Programming Problems with Interval Objective Function Coefficients. *Mathematics of Oper. Res. 6*, pages 222–348, 1981.
83. A. Tikhonov and V. Arsenin. *Solutions of Ill-posed Problems.* Wiley, New York, 1977.
84. A. Törn and A. Žilinskas. *Global Optimization.* Springer-Verlag, Berlin, Heidelberg, New York, 1989.
85. P.A. Wardle. Forest Management and Operational Research: A Linear Programming Study. *Management Science 11*, pages B260–B270, 1965.
86. J.H. Wilkinson. *Rounding Errors in Algebraic Processes.* Prentice-Hall Inc., 1963.
87. J.H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford Univerity Press, Oxford, 1969.
88. J.H. Wilkinson. Modern Error Analysis. *SIAM Rev. 13*, pages 548–568, 1971.
89. S. Zuhe, A. Neumaier, and M.C. Eiermann. Solving Minimax Problems by Interval Methods. *BIT 30*, pages 742–751, 1990.