Computer Arithmetic and Enclosure Methods
L. Atanassova and J. Herzberger (Editors)
Elsevier Science Publishers B.V. (North-Holland)

An Inclusion Algorithm for Global Optimization in a Portable PASCAL–XSC Implementation

Dietmar Ratz
Institut für Angewandte Mathematik
Universität Karlsruhe
Kaiserstraße 12, W-7500 Karlsruhe

Abstract

An algorithm for computing inclusions for all global minimizers of a function $f : \mathbb{R}^n \to \mathbb{R}$ with $f \in C^2(\mathbb{R}^n)$ in a compact interval vector and its implementation in PASCAL–XSC are presented. The algorithm is based on the method of E. Hansen using branch-and-bound techniques and interval arithmetic.

First, some basic concepts, the essential parts of the algorithm, and some modifications are described. Subsequently, the comfortable programming of the algorithm in PASCAL–XSC and the easy use of the optimization program is demonstrated. Due to the C-based implementation of PASCAL–XSC the compiler as well as the optimization algorithm is portable. Thus, numerical results and performance tests of different computer types are presented for some optimization problems.

1 Introduction

We present an implementation of an algorithm for computing guaranteed bounds for all solutions of the global unconstrained optimization problem

$$\min f(x) \qquad \text{subject to } x \in X, \tag{1}$$

where $X \subseteq \mathbb{R}^n$ is a compact interval vector (called *box*) and $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable in X. The algorithm is based on the branch-and-bound method of Hansen [2]. It computes a list of boxes, enclosing the global minimizers of (1). The whole area X is searched by branching in several boxes. Depending on the value of f over a special subbox, it can be decided whether to continue the processing of this box or to delete it. The main features of the algorithm and its implementation are

- interval arithmetic and differentiation arithmetic,
- subdivision (bisection) method,
- box-discarding tests (midpoint test, monotonicity test, concavity test),
- interval Newton step, and
- boundary treatment.

The algorithm is implemented in PASCAL–XSC [6]. The compiler is portable, and the optimization program runs on different computers. We now describe the essential parts of the implemented algorithm and give a short survey of the used PASCAL–XSC features to demonstrate their comfortable programming. After explaining the use of our program, we present test results for some examples. At last, we mention some planned modifications.

1.1 Interval Arithmetic

The basic tool for handling continua and for computing guaranteed bounds for the global minimizers is interval arithmetic (Moore [8], Alefeld/Herzberger [1]) with the property

$$x \in X \Longrightarrow f(x) \in f(X) \subseteq F(X)$$

(*inclusion isotonicity*), with $x \in I\!R^n$, $X \subseteq I\!R^n$, $f : I\!R^n \to I\!R$, and $F : IR^n \to IR$ the machine interval evaluation of f.

The machine interval arithmetic used in our implementation guarantees highest accuracy for the elementary operations and function evaluations due to their definition by semimorphism (Kulisch [7]).

In addition to the general rules for the interval operations, our algorithm deals with cases in which an expression $a - A/B$ occurs, with $a \in I\!R$, $A, B \subseteq I\!R$, and $0 \in B$. For this special application, an extension of the interval arithmetic operations is introduced (Kaucher [4], Moore [8], Hansen [2]). Let $A = [\underline{a}, \overline{a}]$, $0 \in B = [\underline{b}, \overline{b}]$, and $a, b \in I\!R$, then we define

$$A/B = \begin{cases} (-\infty, \infty) & \text{if } \underline{b} = \overline{b} = 0 \\ (-\infty, \infty) & \text{if } \underline{a} < 0 < \overline{a} \\ [\overline{a}/\underline{b}, \infty) & \text{if } \overline{a} \le 0 \ \wedge \ \underline{b} < \overline{b} = 0 \\ (-\infty, \overline{a}/\overline{b}] \cup [\overline{a}/\underline{b}, \infty) & \text{if } \overline{a} \le 0 \ \wedge \ \underline{b} < 0 < \overline{b} \\ (-\infty, \overline{a}/\overline{b}] & \text{if } \overline{a} \le 0 \ \wedge \ 0 = \underline{b} < \overline{b} \\ (-\infty, \underline{a}/\underline{b}] & \text{if } \underline{a} \ge 0 \ \wedge \ \underline{b} < \overline{b} = 0 \\ (-\infty, \underline{a}/\underline{b}] \cup [\underline{a}/\overline{b}, \infty) & \text{if } \underline{a} \ge 0 \ \wedge \ \underline{b} < 0 < \overline{b} \\ [\underline{a}/\overline{b}, \infty) & \text{if } \underline{a} \ge 0 \ \wedge \ 0 = \underline{b} < \overline{b} \end{cases} \quad \text{and} \quad \begin{aligned} a - [b, \infty) &= (-\infty, a - b] \\ a - (-\infty, b] &= [a - b, \infty) \\ a - (-\infty, \infty) &= (-\infty, \infty). \end{aligned}$$

The *width* of an interval $A = [\underline{a}, \overline{a}] \subseteq I\!R$ or an interval vector $X = [\underline{x}, \overline{x}] \subseteq I\!R^n$ is defined by $w(A) = \overline{a} - \underline{a}$ and $w(X) = \max\limits_i(w(X_i))$.

1.2 Differentiation Arithmetic

To avoid explicit programming or entering of formulas for the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_1})$ and the Hessian $\nabla^2 f = (\frac{\partial_2 f}{\partial x_i \partial x_j})_{\ i,j=1,\ldots,n}$ of the objective function f, a differentiation arithmetic (Rall [10]) based on interval arithmetic is used. Thus, we can calculate the value of the gradient and the Hessian matrix automatically when calculating the function value. Also, we can improve the function evaluation by using centered forms (Ratschek/Rokne [11]).

2 The Algorithm

2.1 Subdivision

The basic subdivision algorithm of Hansen [2] can also be applied to non-differentiable functions. For the starting box X and the objective function f, it is given by

1. $Y := X$.
2. Calculate $F(Y)$ and $ify := \inf F(Y)$.
3. Initialize $L := (Y, ify)$.
4. $(Y, ify) := \text{head\_of}\,(L)$.
5. Choose $k \in \{i : w(Y_i) = w(Y)\}$.
6. Bisection: $(V_1, V_2) := \text{bisect}\,(Y, k)$, that is: $Y = V_1 \cup V_2$.
7. Calculate $F(V_1)$, $F(V_2)$, $ifv_1 := \inf F(V_1)$, and $ifv_2 := \inf F(V_2)$.
8. $L := L + (V_1, ifv_1) + (V_2, ifv_2)$ (adding at the end of L).
9. $L := L - (Y, ify)$ (eliminating).
10. If termination criteria does not hold, then goto 4.

The convergence of this method is arbitrarily slow (Ratschek/Rokne [12]). Accelerating devices like *midpoint test, monotonicity test, concavity test*, and *interval Newton step* are necessary.

2.2 Midpoint Test

The midpoint test is used to reduce the number of intervals in the list L. We choose the pair $(\widetilde{Y}, \widetilde{ify})$ out of the list L which satisfies $\widetilde{ify} \leq ifz$ for all pairs (Z, ifz) of L. Then, we compute $\widetilde{f} = \sup F(c)$, with $c = \text{mid}\,\widetilde{Y}$. Now, all pairs (Z, ifz) satisfying $\widetilde{f} < ifz$ can be discarded from the list L. The test remains valid, if an arbitrary $c' \in \widetilde{Y}$ is used instead of $c = \text{mid}\,\widetilde{Y}$. Also, a new pair (W, ifw) must only be entered in the list L if $\widetilde{f} \geq ifw$ is satisfied.

2.3 Monotonicity Test

The monotonicity test is used to figure out whether the function f is *strictly monotone* in a whole subbox $Y \subseteq X$. Then, Y cannot contain a global minimizer in its interior. Thus, a global minimizer can only lie on a part of the edge of Y, if this part is also a part of X. Therefore, if f satisfies

$$\frac{\partial f}{\partial x_i}(y) < 0, \quad \text{for all } y \in Y, \quad \text{or} \quad \frac{\partial f}{\partial x_i}(y) > 0, \quad \text{for all } y \in Y,$$

for any i, then the subbox Y can be deleted or reduced to one of its edges with respect to the i-th component.

With $Y = [\underline{y}, \overline{y}]$, $X = [\underline{x}, \overline{x}]$, and $\nabla_i F = \dfrac{\partial F}{\partial x_i}$, the monotonicity test is:

for $i := 1$ **to** n **do**
  if $0 < \inf \nabla_i F(Y)$ **then**
    if $\underline{x_i} = \underline{y_i}$ **then** $Y_i := [\underline{y_i}, \underline{y_i}]$
    else Delete Y and $Exit_{i\text{-loop}}$
  else if $\sup \nabla_i F(Y) < 0$ **then**
    if $\overline{x_i} = \overline{y_i}$ **then** $Y_i := [\overline{y_i}, \overline{y_i}]$
    else Delete Y and $Exit_{i\text{-loop}}$

2.4 Concavity Test

The concavity test (non-convexity test) in $\mathbb{R}^n$ is used to figure out whether the function f is *not convex* in a subbox $Y \subseteq X$. Then, Y cannot contain a global minimizer in its interior. A global minimizer can only lie on a part of the edge of Y if this part is also a part of X.

A function f is convex in y, if the Hessian $\nabla^2 f(y)$ is positive definite. To be positive definite, it is necessary that all diagonal elements of $\nabla^2 f(y)$ are positive. Therefore, if f satisfies

$$\frac{\partial^2 f}{\partial x_i^2}(y) < 0, \quad \text{for all } y \in Y,$$

for any i, then there is no positive definite matrix within the interval matrix $\nabla^2 f(Y)$, and the subbox Y can be deleted or reduced to its edges with respect to the i-th component.

With $Y = [\underline{y}, \overline{y}]$, $X = [\underline{x}, \overline{x}]$, and $\nabla_{ii}^2 F = \dfrac{\partial^2 F}{\partial x_i^2}$, the concavity test is:

for $i := 1$ **to** n **do**
  if $\sup \nabla_{ii}^2 F(Y) < 0$ **then**
    if $(\underline{x_i} = \underline{y_i})$ **and** $(\overline{x_i} = \overline{y_i})$ **then** $Y_i := [\underline{y_i}, \underline{y_i}] \cup [\overline{y_i}, \overline{y_i}]$ { pair of intervals }
    else if $\underline{x_i} = \underline{y_i}$ **then** $Y_i := [\underline{y_i}, \underline{y_i}]$
    else if $\overline{x_i} = \overline{y_i}$ **then** $Y_i := [\overline{y_i}, \overline{y_i}]$
    else Delete Y and $Exit_{i\text{-loop}}$
if Y was not deleted **then**
  Resolve pair notation of the edge parts and store the remaining vectors.

2.5 Interval Newton Step

For our global optimization method, we apply one step of the interval Newton method (Alefeld/Herzberger [1]) to the problem

$$\nabla f(y) = 0, \quad y \in Y.$$

That is, we compute

$$Y := \mathrm{mid}\,(Y) - (\nabla^2 f(Y))^{-1} \cdot \nabla f(\mathrm{mid}\,(Y))$$

by solving

$$\nabla^2 f(Y) \cdot (\operatorname{mid}(Y) - Y) = \nabla f.$$

For practical computations, a preconditioning and an intersection with the old value is used. For preconditioning, we first calculate $J = \nabla^2 f(Y)$, $M \approx (\operatorname{mid} J)^{-1}$, and $m = \operatorname{mid} Y$. Then we calculate $A = M \cdot J$ and $b = M \cdot \nabla f(m)$ and we solve

$$A(m - Y_N) = b.$$

using the Gauss-Seidel method [9]. The new Y is given by the intersection of the old Y and Y_N. This intersection is performed componentwise in the single steps of the Gauss-Seidel method.

The following theorem (Moore, Nickel, Krawczyk, Kahan, Hansen, Sengupta, Neumaier and others [9]) provides a computational existence and uniqueness test for a zero $y \in Y$.

Theorem: *Suppose $\nabla f : Y \subseteq I\!R^n \to I\!R^n$ is Lipschitz continuous on Y and Y_N is the result of applying one interval Newton step for ∇f to Y, then:*

(a) Every zero $y^* \in Y$ of ∇f satisfies $y^* \in Y_N$.

(b) If $Y_N \cap Y = \emptyset$, then ∇f contains no zero in Y.

(c) If $Y_N \overset{\circ}{\subseteq} Y$ (contained in the interior), then ∇f contains a unique zero in Y.

The following algorithm specifies one step of the interval Newton (Gauss-Seidel) method in $I\!R^n$ applied to the gradient of f.

1. Calculate $J = \nabla^2 f(Y)$ and $M \approx (\operatorname{mid} J)^{-1}$.
2. Calculate $m = \operatorname{mid} Y$, $A = M \cdot J$, and $b = M \cdot \nabla f(m)$.
3. For $i = 1, 2, \ldots, n$:
   (a) Calculate $Y_i := \left(m_i - A_{ii}^{-1} \left(b_i + \sum_{j=1, j\neq i}^{n} A_{ij}(Y_j - m_j) \right) \right) \cap Y_i.$
   (b) If $Y_i = \emptyset$, goto 6.
   (c) If Y_i has a gap $(0 \in A_{ii})$, then save i and the gap.
4. If there is no gap, then set $V_1 := Y$ and goto 7.
5. Use the largest gap to split Y in V_1 and V_2 after closing all other gaps. Goto 7.
6. $Y := \emptyset$.
7. End.

It is a simplified form of the special relaxation method (Gauss-Seidel method) used by Hansen-Greenberg [3].
In our implementation, we do evaluate the whole sum

$$b_i + \sum_{j=1, j\neq i}^{n} A_{ij}(Y_j - m_j)$$

with highest accuracy, by using an *accurate expression* provided by PASCAL–XSC [6].

2.6 Boundary Treatment

We have to take special care of the edge of the starting box X because it is possible, that a global minimizer lies on the edge. Thus, after each step of the algorithm which possibly deletes parts of the edge of X (Newton step), a special treatment of the boundary points is necessary. This can be achieved by entering the corresponding parts of the edge into the list L as degenerated boxes (dimension $< n$).

2.7 The Implemented Algorithm

In our implementation we enter the pairs (Y, ify) in the list L by satisfying the two conditions:

- The second components of the list elements may not decrease.
- The new box is entered behind all older boxes having equal second components.

We use the following modification of the algorithm of Hansen:

1. $Y := X$.
2. Calculate $F(Y)$ and $\tilde{f} = \sup F(c)$, where $c = \operatorname{mid} Y$.
3. $\mathit{ify} := \inf F(Y)$.
4. Initialize $L := (Y, \mathit{ify})$.
5. Choose $k \in \{i : w(Y_i) = w(Y)\}$.
6. Bisection: $(V_1, V_2) := \operatorname{bisect}(Y, k)$, that is: $Y = V_1 \cup V_2$.
7. For $i = 1, 2$:
   (a) Monotonicity test for V_i. If deleted: $\mathit{Exit}_{i\text{-loop}}$.
   (b) $\mathit{ifv}_i := \inf F(V_i)$.
   (c) Midpoint test for V_i. If $\tilde{f} < \mathit{ifv}_i$: $\mathit{Exit}_{i\text{-loop}}$.
   (d) Concavity test for V_i. If deleted: $\mathit{Exit}_{i\text{-loop}}$.
   (e) Newton step applied to V_i, getting at most two boxes W_{i1}, W_{i2}. $p :=$ "number of boxes" ($= 0$, 1, or 2).
   (f) Boundary treating.
   (g) For $j = 1$ to p:
      i. Monotonicity test for W_{ij}. If deleted: $\mathit{Exit}_{j\text{-loop}}$.
      ii. $\mathit{ifw}_{ij} := \inf F(W_{ij})$.
      iii. Midpoint test for W_{ij}. If $\tilde{f} < \mathit{ifw}_{ij}$: $\mathit{Exit}_{j\text{-loop}}$.
      iv. $L := L + (W_{ij}, \mathit{ifw}_{ij})$ (with respect to ifw_{ij} and age of boxes).
8. $L := L - (Y, \mathit{ify})$ (eliminating).
9. $(Y, \mathit{ify}) := \text{head\_of}(L)$ (satisfying $\mathit{ify} \le \mathit{ifz}$ for all pairs (Z, ifz)).
10. $c := \operatorname{mid} Y$ and $\tilde{f} = \min(\tilde{f}, \sup F(c))$.
11. Midpoint test: Remove all pairs (Z, ifz) satisfying $\tilde{f} < \mathit{ifz}$ from the list.
12. If termination criteria not satisfied, then goto 5.

3 The Use of PASCAL–XSC

The programming language PASCAL–XSC [6] is a powerful tool for programming our global optimization algorithm. The important features for our implementation are

- operators and functions of arbitrary result type
- overloading of procedures, functions, operators, and assignments
- controlled rounding
- highly accurate interval arithmetic operators and elementary functions
- exact evaluation of expressions (#-expressions)

Concerning the implementation of our algorithm, many necessary features are already available in PASCAL–XSC. The language concepts allow a comfortable programming of all other features needed in the algorithm.

The predefined *interval arithmetic* for scalar, vector, and matrix types allows access to all usual operations and functions in the usual mathematical notation. All basic operations and function evaluations are of maximum accuracy (1 or 2 ulp). *Extended interval arithmetic* was implemented using the operator concept. So, all operations of the Newton step can be written close to their mathematical notations.

Differentiation arithmetic was implemented using the operator concept and functions with arbitrary result type covering all usual (trigonometric, hyperbolic, etc.) functions. All objective function evaluations can be processed with automatic computation of the gradient and the Hessian.

The *list operations* were implemented using a pointer type, operator concept, and functions with arbitrary result type. So, the notations $L := L + (Y, ify)$, $L := L - (Y, ify)$, or *head_of (...)* used in the algorithm can be used for programming in the same way.

Controlled rounding of all input data is guaranteed by the overloaded input statements. *High accuracy* in the Newton step (Gauss-Seidel step) is achieved by using the predefined *accurate expression*. The sum $b_i + \sum_{j=1, j\neq i}^{n} A_{ij}(Y_j - m_j)$ is calculated with high accuracy by

```
## (b[i] + (for j:=1 to i-1 sum (A[i,j]*Y_minus_m[j]))
        + (for j:=i+1 to n sum (A[i,j]*Y_minus_m[j])))
```

where `Y_minus_m` is defined as `Y - m`.

Interactive input of objective functions was made possible by the implementation of a procedure generating a postfix form and of corresponding functions for the evaluation of the postfix form with automatic differentiation.

4 The Program

The features of our optimization program are

- Organization by menu technique.
- Input from files which can be edited during the program sessions. The structure of the input files is as follows:

```
------ Top of File ------
$n:                                <--- Name (optional)
   Six-hump-camel-back-function
$f:                                <--- Function definition
   4*x^2 - 2.1*x^4 + x^6 / b + x * y - 4*y^2 + 4*y^4;
$v:                                <--- Variables
   x:=[-2.5,2.5];
   y:=[-2.5,2.5];
$p:                                <--- Parameters
   b:=3;
$e:                                <--- Tolerance
   1e-10
------ End of File ------
```

- The objective function f can be specified by a formula consisting of the unknowns (variables), the operations +, -, *, /, and ^ (power), parentheses, all usual mathematical functions, explicit constants, and named constants (parameters).
- The result is given as a list of boxes which can be displayed on screen and/or saved on file.

The implementation of the program is still in progress with respect to some planned improvements, but the actual version of the program is working very well. At the moment, the program runs on personal computers (IBM and compatible), ATARI ST, HP workstations, and SUN SPARCstations. On the IBM 3090, an installation of the PASCAL–XSC compiler is not yet completed, so it was not possible to run the program there. Due to the portability of the compiler, the program is portable, too. Thus, the application of the program is not restricted to special machines or operating systems.

5 Test Results

5.1 Test Functions

The following list gives an overview of some test examples we used for our algorithm (see [13] for a detailed description of parameters).

1. Hansen's function: $24x^4 - 142x^3 + 303x^2 - 276x + 93, \quad 0 \le x \le 3.$
2. Shubert's function: $-\sum_{k=1}^{5} k \sin((k+1)x + k), \quad -10 \le x \le 10.$
3. Function f_1 of [13]: $\sin x + \sin \frac{10x}{3} + \ln x - 0.84x, \quad 2.7 \le x \le 7.5.$
4. Function f_4 of [13]: $(x + \sin x)e^{-x^2}, \quad -10 \le x \le 10.$
5. Six hump camel back function: $4x_1^2 - 2.1x_1^4 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 + 4x_2^4, \quad -5 \le x_i \le 5.$

6. Branin's function: $(\frac{5x_1}{\pi} - \frac{5.1x_1^2}{4\pi^2} + x_2 - 6)^2 + (10 - \frac{10}{8\pi})\cos x_1 + 10, \quad -5 \le x_1 \le 10,$ $0 \le x_2 \le 15.$

7. Rastrigin's function: $x_1^2 + x_2^2 - \cos 18x_1 - cos 18x_2, \quad -1 \le x_i \le 1.$

8. Hartman's function: $-\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right), \quad 0 \le x_i \le 1.$

9. Shekel's function: $-\sum_{i=1}^{m} \frac{1}{(x - A_i)(x - A_i)^T + c_i}, \quad 0 \le x_i \le 10.$

10. Griewank's function: $\sum_{i=1}^{n} \frac{x_i^2}{d} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1, \quad -6 \le x_i \le 6.$

11. Function of Goldstein and Price: $(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)), \quad -2 \le x_i \le 2.$

5.2 Performance

We tested the examples with a tolerance 1E−2 for the function evaluation of f and the enclosures of the global minimizers on an IBM PS/2 Model 70 (386-16), on a SUN SPARC SLC workstation, and on a HP 9000/835 Turbo SRX workstation. In the following table we compare the performance results of these machines using a PASCAL–XSC version equipped with a software arithmetic.

No. of Funct.	Dimen-sion	Funct. Eval.	Grad. Eval.	Hess. Eval.	Mini-mizers	Iterat. Steps	CPU time (min:sec)		
							PC	SUN	HP
1	1	53	109	37	1	20	0:07	0:04	0:02
2	1	30	131	27	3	27	2:26	1:25	0:50
3	1	8	26	6	1	6	0:14	0:07	0:04
4	1	7	35	10	1	6	0:11	0:06	0:03
5	2	89	406	120	2	81	0:55	0:29	0:16
6	2	20	56	14	3	13	0:19	0:10	0:06
7	2	17	39	9	1	7	0:11	0:06	0:03
$8_{n=3}$	3	50	200	39	1	46	2:21	1:15	0:39
$8_{n=6}$	6	4607	18682	3269	1	4504	411:21	218:15	115:39
$9_{m=5}$	4	61	298	60	1	60	2:40	1:25	0:45
$9_{m=10}$	4	84	413	83	1	83	5:20	2:50	1:31
$10_{n=2}$	2	11	56	13	1	10	0:23	0:12	0:07
$10_{n=5}$	5	62	404	96	1	61	7:23	4:00	2:06
11	2	9004	22002	8583	1	5178	94:50	50:43	26:07

6 Future Works

Some modifications of the implementation are planned to improve the performance of our optimization program and to spread its application field.

- Use of special preconditioners for the Gauss-Seidel step (suggested by Kearfott [5]).
- Use of local floating-point iterations in the Gauss-Seidel step (suggested by Hansen-Greenberg [3]).
- Use of the prove of uniqueness of the global minimizer within the computed box.
- Improved boundary treatment.
- Use of new tests for updating the approximation $\widetilde{f}$ and new branching strategies.
- Use of an approximation for the minimum computed with floating-point arithmetic.
- Extension of the program to handle constraint optimization problems.
- Development of an interface to make the optimization routine accessible by other programs.

References

[1] Alefeld, G. and Herzberger, J.: *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] Hansen, E.: *Global Optimization Using Interval Analysis – The Multi-Dimensional Case*. Numer. Math. **34**, pp. 247–270, 1980.

[3] Hansen, E. and Greenberg, R.: *An Interval Newton Method*. Applied Math. and Comp., **12**, pp. 89–98, 1983.

[4] Kaucher, E.: *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Räume*. Dr. Theses, University of Karlsruhe, 1973.

[5] Kearfott, R. B.: *Preconditioners for the Interval Gauss-Seidel Method*. SIAM J. Numer. Anal., **27**, 1990.

[6] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., and Ullrich, Ch.: *PASCAL-XSC – Language Reference with Examples*. Springer-Verlag, Heidelberg, to be published 1991/92.

[7] Kulisch, U. and Miranker, W. L.: *Computer Arithmetic in Theory and Practice*. Academic Press, 1981.

[8] Moore, R. E.: *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, Pensylvania, 1979.

[9] Neumaier A.: *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[10] Rall, L. B.: *Automatic Differentiation, Techniques and Applications*. Lecture Notes in Computer Science, No. 120, Springer, Berlin, 1981.

[11] Ratschek, H. and Rokne, J.: *Computer Methods for the Range of Functions*. Ellis Horwood Limited, 1984.

[12] Ratschek, H. and Rokne, J.: *New Computer Methods for Global Optimization*. Ellis Horwood Limited, 1988.

[13] Törn, A. and Žilinskas, A.: *Global Optimization*. Lecture Notes in Computer Science, No. 350, Springer, 1989.