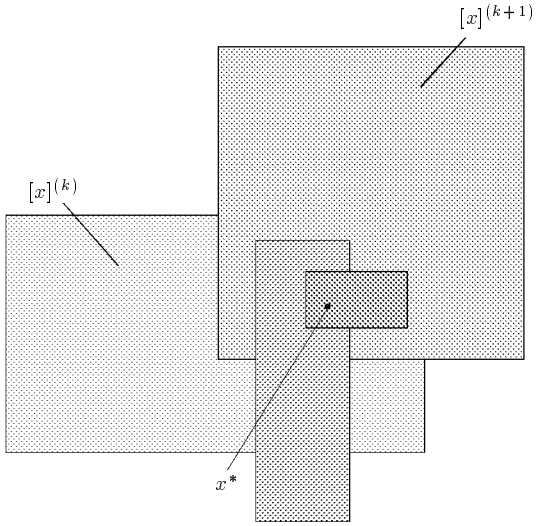
Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method

Stefan Herbort, Dietmar Ratz

Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation



Bericht 2/1997

Impressum

Herausgeber:	Institut für Angewandte Mathematik Lehrstuhl Prof. Dr. Ulrich Kulisch Universität Karlsruhe (TH) D-76128 Karlsruhe
Redaktion:	Dr. Dietmar Ratz

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über

`ftp://iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis: `/pub/documents/reports`

oder über die World Wide Web Seiten des Instituts

`http://www.uni-karlsruhe.de/~iam`

Autoren-Kontaktadresse

Rückfragen zum Inhalt dieses Berichts bitte an

Dietmar Ratz, Stefan Herbort
Institut für Angewandte Mathematik
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: Dietmar.Ratz@math.uni-karlsruhe.de

Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method

Stefan Herbort, Dietmar Ratz

Contents

Zusammenfassung

Ein komponentenweises Newton-Verfahren: Zur Einschließung aller Lösungen eines nichtlinearen Gleichungssystems wird ein leistungsfähiges Branch-and-Prune Verfahren vorgestellt. Es basiert auf einem komponentenweise arbeitenden Intervall-Newton-Operator, der die Funktionen eines Gleichungssystems vorübergehend wie eindimensionale reelle Funktionen mit Intervall-Koeffizienten behandelt. Durch geschickte Anwendung der Intervallrechnung und mit Hilfe diverser Verbesserungen an der komponentenweisen Methode ist ein Verfahren entstanden, das vor allem bei ,,real-world"-Problemen sehr effizient arbeitet.

Abstract

A Componentwise Interval Newton Method: We give an efficient branch-and-prune algorithm for finding enclosures of all solutions of a system of nonlinear equations. It is based on a componentwise interval Newton operator that temporarily considers a function of the system of equations as a one-dimensional real-valued function having interval coefficients. Using interval arithmetic and enhancing the componentwise method by several techniques, we present an algorithm that works rather efficiently, especially on many "real-world" problems.

1 Introduction

We address the problem of reliably finding all solutions of the nonlinear system

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1, \ldots, n, \tag{1}$$

where the variables x_j are bounded by real intervals:

$$x_j \in [x]_j, \quad j = 1, \ldots, n.$$

As usual, the set of real intervals is denoted by $I\!I\!R$, accordingly $I\!I\!R^n$ is the set real interval vectors. Thus, we denote the search area by $[x] = ([x]_1, [x]_2, \ldots, [x]_n)^\top \in I\!I\!R^n$. When we write a function with interval arguments, e.g. $f([x])$, we always think of its natural interval extension. Usually, we do not have the exact range of the function, anyway.

With $f = (f_1, f_2, \ldots, f_n)^\top$ we are looking for enclosures of all $x^* \in [x] \subseteq I\!I\!R^n$ with

$$f(x^*) = 0. \tag{2}$$

For this purpose we introduce a componentwise interval Newton method (Sections 2–4) that works rather efficiently. We give an algorithmic discription (Section 5) and some numerical results (Section 7). The algorithm is close to that from [2, Chapter 13], and we also compare the results of our componentwise method to that method. For detailed information on the componentwise method see [4].

2 Idea of a Componentwise Interval Newton Method

The componentwise Newton method for finding enclosures of the solution vectors of a system of nonlinear equations is based on a quite simple idea using the properties of interval arithmetic. Similar conceptions can be found in [5] and [14].

The n-dimensional system of equations

$$f(x) = (f_1(x), \ldots, f_n(x))^\top = 0 \quad \text{where} \quad f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n \tag{3}$$

consists of the n real-valued functions $f_i : D \to \mathbb{R}$.

The vector $x^* \in D$ is a solution of (2) iff

$$f_i(x^*) = 0 \quad \text{for all} \quad i \in \{1, 2, \ldots, n\}.$$

Hence, by leaving $n-1$ of the n variables fixed each function f_i can be interpreted as a one-dimensional real-valued function. Now, the ordinary interval Newton method for one-dimensional functions can be applied. Of course, it is not quite probable to find any solution of $f_i = 0$ if the fixed variables are set to some arbitrary real values. But, by replacing the fixed variables by the corresponding components $[x]_j$ of the interval vector $[x]$ we can get enclosures of a solution, and interval arithmetic guarantees that no solution is lost.

Example 2.1 Let $f = (f_1, f_2)^\top : D \subseteq \mathbb{R}^2 \to \mathbb{R}^2$ a continuously differentiable function and let $[x] \in I\mathbb{R}^2$ be an interval vector (box) with $[x] = ([x]_1, [x]_2)^\top \subseteq D$.

We consider the function $f_1 : D \to \mathbb{R}$. As a necessary condition for $x^* \in [x]$ solving the system (2) we get:

$$f_1(x^*) \stackrel{!}{=} 0.$$

Let the function $g : \mathbb{R} \to \mathbb{R}$ be defined by

$$g(y) := f_1(x_1^*, y).$$

Applying the one-dimensional interval Newton method all roots of g can be found, and because of the above mentioned necessary condition any root y^* of g is a "candidate" for the 2nd component of a solution of problem (2). Since the point x_1^* is usually unknown, it is replaced by the interval $[x]_1$, i.e. by the 1st component of the search box. Thus we get the expression $f_1([x]_1, y)$, and, since the interval extension is inclusion isotonic, we have

$$g(y) \in f_1([x]_1, y).$$

The derivative $g'([x]_2)$ that is required for the interval Newton operator is treated in the same manner. This leads to:

$$N([x]_2) := c - \frac{g(c)}{g'([x]_2)} \subseteq c - \frac{f_1([x]_1, c)}{\frac{\partial f_1}{\partial x_2}([x]_1, [x]_2)}, \tag{4}$$

where $c := m([x]_2)$; assuming $0 \notin g'([x]_2) = \frac{\partial f_1}{\partial x_2}([x]_1, [x]_2)$. Applying (4) the component $[x]_2$ of the search interval vector may be reduced. In the same way, i.e. leaving x_2 fixed and replacing it by the (possibly) reduced 2nd component of $[x]$, we can treat the 1st component $[x]_1$ of the search box.

Considering this first example some advantages of the componentwise method can already be seen. Solving systems of nonlinear equations with the multi-dimensional Newton method or the preconditioned interval Newton Gauss-Seidel method (see [2, Chapter 13]) requires the inversion of a matrix. Since the componentwise method treats a system in a quasi one-dimensional way, no inversion is necessary. Hence, the existence of a singular matrix in the Jacobian $J_f([x])$ does not mean any problem; using extended interval arithmetic the componentwise method is applicable in any case. Moreover, for each of the n components $[x]_j$ of the search box n different functions f_i can be chosen trying to prune the interval $[x]_j$. We will see that a suitable choice can effect a high performance of the componentwise Newton method.

The following section deals with the detailed derivation of a componentwise interval Newton operator and its most important properties.

3 The Componentwise Newton Operator N_{cmp}

Similar to the ordinary one-dimensional interval Newton method we derive the componentwise method from the mean-value theorem. Let $f = (f_1, \ldots, f_n)^\top : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be the considered function. By choosing one component f_i of the multi-dimensional function and leaving all variables but x_j fixed we get the one-dimensional, real-valued function

$$\tilde{f}_{(ij)}(x_j) := f_i(x_1^*, \ldots, x_{j-1}^*, x_j, x_{j+1}^*, \ldots, x_n^*)$$

with $i, j \in \{1, 2, \ldots, n\}$ and $x_k^* \in \mathbb{R}$ $(k \neq j)$.

The real variable x_j may vary within the jth component $[x]_j$ of the search box $[x] \in I\mathbb{R}^n$. So we get from the already mentioned mean-value theorem:

$$\exists \xi \in [x]_j : \ \tilde{f}_{(ij)}(c_j) - \tilde{f}_{(ij)}(x_j) = \tilde{f}'_{(ij)}(\xi) \cdot (c_j - x_j), \tag{5}$$

where $x_j \in [x]_j$ and $c_j := m([x]_j)$. According to the definition of $\tilde{f}_{(ij)}$ the derivative $\tilde{f}'_{(ij)}$ is the partial derivative $\frac{\partial f_i}{\partial x_j}$.

Assuming $x^* \in \mathbb{R}^n$ to be a root of f and $\tilde{f}'_{(ij)} \neq 0$, (5) can be transformed to

$$x_j^* = c_j - \frac{\tilde{f}_{(ij)}(c_j)}{\tilde{f}'_{(ij)}(\xi)}$$

because x^* is a zero of f_i as well. Now, replacing the unknown $\xi \in [x]_j$ by the whole interval $[x]_j$ leads to

$$x_j^* \in c_j - \frac{\tilde{f}_{(ij)}(c_j)}{\tilde{f}'_{(ij)}([x]_j)} =: \mathcal{N}.$$

However, the step from the real function $\tilde{f}'_{(ij)}$ to its interval extension may lead to $0 \in \tilde{f}'_{(ij)}([x]_j)$. We handle this case by applying the extended interval division. Then the set N may have a gap.

The only information on the fixed x_k^* (with $k \neq j$) appearing in function $\tilde{f}_{(ij)}$ is the interval they have to lie in. If the search box is $[x] = ([x]_1, \ldots, [x]_n)^\top \in I\mathbb{R}^n$ then x_k^*

may vary within $[x]_k$. Substituting each x_k^* $(k \neq j)$ by the corresponding interval $[x]_k$ we get a superset of $\mathcal{N}$ because of inclusion isotonicity. Obviously, inclusion isotonicity must also hold for the extended interval arithmetic (see [4, Chapter 3], [13]).

In terms of the original system of equations we have

$$\mathcal{N} \subseteq c_j - \frac{f_i([x]_1, \ldots, [x]_{j-1}, c_j, [x]_{j+1}, \ldots, [x]_n)}{\frac{\partial f_i}{\partial x_j}([x]_1, \ldots, [x]_n)}.$$

Now, we can define the following operator that uses the ith component of the system of equations to treat the jth component of the search box.

Definition 3.1 *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$, $f = (f_1, \ldots, f_n)^\top$ be a continuously differentiable function, and let $[x] = ([x]_1, \ldots, [x]_n)^\top \in I\mathbb{R}^n$ be an interval vector with $[x] \subseteq D$ and $i, j \in \{1, \ldots, n\}$. Then the* componentwise interval Newton operator N_{cmp} *is defined by*

$$N_{\mathrm{cmp}}([x], i, j) := m([x]_j) - \frac{f_i([x]_1, \ldots, [x]_{j-1}, m([x]_j), [x]_{j+1}, \ldots, [x]_n)}{\frac{\partial f_i}{\partial x_j}([x]_1, \ldots, [x]_n)} \tag{6}$$

Similar to other interval Newton operators the operator N_{cmp} has some important properties concerning the existence of a zero.

Theorem 3.2 *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function, and let $[x] = ([x]_1, \ldots, [x]_n)^\top \in I\mathbb{R}^n$ be an interval vector with $[x] \subseteq D$. Then, the componentwise interval Newton operator N_{cmp} has the following properties:*

1. *Let $x^* \in [x]$ be a zero of f, then we have for arbitrary $i, j \in \{1, \ldots, n\}$:*
   $x^* \in ([x]_1, \ldots, [x]_{j-1}, N_{\mathrm{cmp}}([x], i, j), [x]_{j+1}, \ldots, [x]_n)$
2. *If $N_{\mathrm{cmp}}([x], i, j) \cap [x]_j = \emptyset$ for any $i, j \in \{1, \ldots, n\}$, then there exists no zero of f in $[x]$.*

Proof: The proof of 1 is very close to the derivation of the N_{cmp}-operator. Let $x^* \in [x]$ be a zero of f, and let $i, j \in \{1, \ldots, n\}$. Considering f_i as a real-valued one-dimensional function in x_j, we get from the mean-value theorem

$$x_j^* = c_j - \frac{f_i(x_1^*, \ldots, x_{j-1}^*, c_j, x_{j+1}^*, \ldots, x_n^*)}{\frac{\partial f_i}{\partial x_j}(x_1^*, \ldots, x_{j-1}^*, \xi, x_{j+1}^*, \ldots, x_n^*)} \quad \text{with } \xi \in [x]_j,$$

where $c_j := m([x]_j)$ and assuming $\frac{\partial f_i}{\partial x_j}(\cdots) \neq 0$. Replacing the unknown ξ by the interval $[x]_j$, we get an inclusion of x_j^*. Moreover, we substitute each x_k^* (with $k \neq j$) by the corresponding $[x]_k$, that leads to a superset of the mentioned inclusion. Hence, we have:

$$\begin{aligned} x_j^* &\in c_j - \frac{f_i(x_1^*, \ldots, x_{j-1}^*, c_j, x_{j+1}^*, \ldots, x_n^*)}{\frac{\partial f_i}{\partial x_j}(x_1^*, \ldots, x_{j-1}^*, [x]_j, x_{j+1}^*, \ldots, x_n^*)} \\ &\subseteq c_j - \frac{f_i([x]_1, \ldots, [x]_{j-1}, c_j, [x]_{j+1}, \ldots, [x]_n)}{\frac{\partial f_i}{\partial x_j}([x]_1, \ldots, [x]_{j-1}, [x]_j, [x]_{j+1}, \ldots, [x]_n)} \\ &= N_{\mathrm{cmp}}([x], i, j), \end{aligned}$$

due to inclusion isotonicity.

Since the operator $N_{\text{cmp}}([x], i, j)$ only treats the jth component of $[x]$, it is shown:

$$x^* \in ([x]_1, \ldots, [x]_{j-1}, N_{\text{cmp}}([x], i, j), [x]_{j+1}, \ldots, [x]_n).$$

Item 2 can be proved by contradiction:
Assuming $x^* \in [x]$ to be a zero of f, and applying 1 for some arbitrary $i, j \in \{1, \ldots, n\}$, we have

$$x_j^* \in N_{\text{cmp}}([x], i, j) \subseteq N_{\text{cmp}}([x], i, j) \cap [x]_j = \emptyset,$$

and the assumed existence of a zero is contradicted. □

4 Some Improvements on the Componentwise Method

The described operator has been tested within a quite naive implementation in which the $N_{\text{cmp}}([x], i, j)$-operator is simply applied for $i = j$ and $i = 1, \ldots, n$. The ith component of the search box $[x]$ is replaced by the results of $N_{\text{cmp}}([x], i, i)$ intersected with the old $[x]_i$ preventing the iterative method from diverging. Termination is guaranteed by bisecting the box if the operator does not prune it sufficiently, and extended interval arithmetic is used if $0 \in \frac{\partial f_i}{\partial x_i}([x])$ for some $i \in \{1, \ldots, n\}$. The results of that implementation has been rather varying in comparison to the `nlss`-module from [2]. Many modifications have been tested to improve the componentwise operator on the difficult problems. We discuss the most successful ideas.

For detailed information on the used inclusion isotonic extended interval arithmetic see [4, Chapter 3] or [13].

4.1 Using Index-Lists

The definition of the N_{cmp}-operator is made for arbitrary indices i, j with $i, j \in \{1, \ldots, n\}$. Hence, we have several possibilities to choose such pairs (i, j). We want to reduce the interval vector $[x]$, but the N_{cmp}-operator only treats the jth component. Therefore, we have to apply N_{cmp} for all $j \in \{1, \ldots, n\}$, successively. Even if an order of j is given, for each j the index i can be chosen from $\{1, \ldots, n\}$. Experimental results show that the choice of the pairs (i, j) has big influence on the efficiency of the componentwise method. This is especially true if the operator is applied on sparse systems.

Obviously, it does not make sense to compute the N_{cmp}-operator, if the partial derivative, i.e. the denominator in (6) (page 7), equals zero. In that case, the extended interval division would result in $(-\infty, +\infty)$, provided that the numerator contains zero. Thus, any evaluation of the function or the derivative would not make sense, since the N_{cmp}-operator could not reduce the search box.

Now, automatic differentiation in conjunction with interval arithmetic seems to be rather convenient. We need a single evaluation of the Jacobian matrix for the start box $[y]$ to obtain some important information about the system of equations. For example,

let $[A] \supseteq J_f([y])$ be an interval matrix that includes the Jacobian over $[y]$. If $[a]_{ij} = 0$, the partial derivative $\frac{\partial f_i}{\partial x_j}$ equals zero. Because of inclusion isotonicity we must have $[a]_{ij} = 0$ for any $[\tilde{y}] \subseteq [y]$. That means, we should not apply the N_{cmp}-operator on any pair (i,j) with $[a]_{ij} = 0$. If $0 \in f([y])$ holds for the current search box $[y]$ (otherwise, there is nothing to do on $[y]$), then the numerator in our operator will mostly contain zero.

Using these ideas, it is possible to create a list of pairs (i,j) giving the parameters for the N_{cmp}-operator. Such an "index-list" can be created once before the main algorithm is started. Each element (i,j) signals that the component $[x]_j$ of the current search box should be treated by the component f_i of the function. The actual construction of an index-list can be performed in various manners. Probably, it is not possible to create something like an optimal list. What we are interested in is to find an optimal function $f_{\tilde{\imath}}$ to a given component $\hat{\jmath}$, so that $d(N_{\text{cmp}}([x],i,\hat{\jmath}) \cap [x]_{\hat{\jmath}})$ becomes the minimum among all $i \in \{1,\ldots,n\}$ for $i = \tilde{\imath}$. That is, finding an index $\tilde{\imath}$ with

$$d(N_{\text{cmp}}([x],\tilde{\imath},\hat{\jmath}) \cap [x]_{\hat{\jmath}}) = \min_{i \in \{1,\ldots,n\}} d(N_{\text{cmp}}([x],i,\hat{\jmath}) \cap [x]_{\hat{\jmath}}).$$

Using $i = \tilde{\imath}$, the componentwise Newton operator would work most efficiently on the given component $\hat{\jmath}$.

It is rather expensive to compute the optimal parameters for N_{cmp}, and it is not practical do this in each step of a componentwise method. But there is another idea, we can achieve high efficiency with. For finding the optimal parameter i, we must at least compute n values of the function f and n values of partial derivatives. This effort can be used as following: we do not really calculate the best i before applying the N_{cmp}-operator, but we compute

$$[x] := ([x]_1,\ldots,[x]_{\hat{\jmath}-1},\ N_{\text{cmp}}([x],i,\hat{\jmath}) \cap [x]_{\hat{\jmath}},\ [x]_{\hat{\jmath}+1},\ldots,[x]_n)$$

for all $i = 1,\ldots,n$, successively. After these n steps the diameter of $[x]_{\hat{\jmath}}$ is at most as large as it would be using the optimal function $f_{\tilde{\imath}}$ and computing N_{cmp} once. The component $[x]_{\hat{\jmath}}$ can even be smaller because the possibly improved component is used immediately in the next step. Thus, we can achieve good reduction of the $\hat{\jmath}$th component of the search box. Moreover, we do not need to compute $N_{\text{cmp}}([x],i,\hat{\jmath})$ with a fixed $\hat{\jmath}$ for all functions f_i $(i = 1,\ldots,n)$, but we can create an index-list to control the indices i to be used.

In an improved implementation of the componentwise Newton method two different index-lists are created to determine the application of N_{cmp}. The pairs (i,j) of the first list L_1 are used if the element of the Jacobian does not contain zero, i.e. usual interval division is applicable. The indices in the second list L_2 are only used if the needed element of the Jacobian contains zero and therefore extended interval arithmetic has to be applied. The lengths of the index-lists depend on the evaluation of the Jacobian over the starting search box.

The list L_1 is built as follows, starting with the diagonal element, going downwards and jumping from the last to the first row, all components in a column of the Jacobian are analysed. The pair (i,j) is added to the list if the corresponding element of the Jacobian does not equal the thin interval $[0,0]$, because in that case usual interval

division would never be applicable. With an additional parameter *max_f/* that may vary from 1 to n we can control the maximum number of index-pairs that is added to the list L_1 for a single column of the Jacobian.

The application of extended interval arithmetic may produce splittings and thereby require a large number of recursive calls. Hence, the list L_2 has the maximum length of n. From each column of the Jacobian at most one pair (i, j) is added to L_2. In the current implementation we choose in each column the element with maximum diameter that contains zero. So, the list may also be empty if none of the components of the Jacobian contains zero.

Example 4.1 We look at the following sparse system to understand the manner in which the lists are created.

$$f = \begin{pmatrix} 10 \cdot (x_2 - x_1^2) \\ 1 - x_1 \\ 10 \cdot (x_4 - x_3^2) \\ 1 - x_3 \end{pmatrix}$$

Evaluating the Jacobian over the starting box $[x] = [-10, 10]^4$ using automatic differentiation we get the interval matrix:

$$J_f([x]) = \begin{pmatrix} [-200, 200] & [10, 10] & [0, 0] & [0, 0] \\ [-1, -1] & [0, 0] & [0, 0] & [0, 0] \\ [0, 0] & [0, 0] & [-200, 200] & [10, 10] \\ [0, 0] & [0, 0] & [-1, -1] & [0, 0] \end{pmatrix}$$

If maximum length of L_1 is allowed, then each position with an interval component unequal to $[0, 0]$ is added to the list. As described above, the diagonal elements have the priority in each column. Hence, we get

$$L_1 = \{(1, 1), (2, 1), (1, 2), (3, 3), (4, 3), (3, 4)\}$$

The list L_2 contains the index-pairs for extended interval arithmetic. So, the only relevant elements are those that contain zero but are different from $[0, 0]$. Among these "candidates", in each column the interval with the maximum diameter is chosen to determine the entry for L_2. That leads to the second list:

$$L_2 = \{(1, 1), (3, 3)\}$$

Certainly, there are many other ways to construct such index-lists. Moreover, we can imagine to change the lists during the algorithm, for different search boxes would probably lead to different index-lists. We looked for a method which does not need to many evaluations of the Jacobian because that costs a lot of time. The described procedure that builds both lists once before the main algorithm starts, has been developed in a quite heuristical way, but it works well on most of the test-problems.

4.2 Combination with an Interval Newton Gauss-Seidel Step

Many tests with a simple implementation of the N_{cmp}-operator have shown that the componentwise method sometimes does not work very well. Especially, if the system is almost linear the operator is often not able to reduce the search box, and many bisections are necessary. However, the module `nlss` from [2] using an interval Newton Gauss-Seidel operator is quite efficiently on those problems. On the other hand, the componentwise method using the index-lists is the better one on sparse systems and many other problems.

For detailed information on the interval Newton Gauss-Seidel method see [1], [3] or [12]. Because of the importance of the interval Newton Gauss-Seidel operator for the module `Ncmp_mod` (described below) we cite the following theorem giving the most important properties of that operator (see [2]).

Theorem 4.2 *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function, and let $[x] \in I\mathbb{R}^n$ be an interval vector with $[x] \subseteq D$. The interval Newton Gauss-Seidel operator $N_{\text{GS}}([x])$ from [2] has the following properties:*

1. *Every zero $x^* \in [x]$ of f satisfies $x^* \in N_{\text{GS}}([x])$.*
2. *If $N_{\text{GS}}([x]) = \emptyset$, then there exists no zero of f in $[x]$.*
3. *If $N_{\text{GS}}([x]) \overset{\circ}{\subset} [x]$, then there exists a unique zero of f in $[x]$ and hence in $N_{\text{GS}}([x])$.*

Proof: See [11]. □

The interval Newton Gauss-Seidel step from [2] is very expensive because it needs the evaluation of the whole Jacobian matrix and the inverse of its midpoint matrix. Hence, a combination of the componentwise Newton operator with the Gauss-Seidel operator should be realized in a way that a Gauss-Seidel step is applied as rarely as possible but as often as necessary. On the other hand, it is quite difficult to decide which operator should be used next until both of them have really been calculated. The following method has been found to be rather successful.

A simple version of the Gauss-Seidel step is applied only if the N_{cmp}-operator returns a single box, i.e. no splittings have been produced. That simplified Gauss-Seidel step does not produce any splittings, because it is not very effective to produce too many recursive calls. Hence, the search box may only be splitted by bisection or by the N_{cmp}-operator using extended interval division. We emphasize that the simplified Gauss-Seidel method still has all the properties of $N_{\text{GS}}([x])$ mentioned in Theorem 4.2.

4.3 Verifying the Uniqueness of a Solution

To prove the uniqueness of a solution we apply Theorem 4.2. Accordingly, a box $[x]$ contains a locally unique solution if the condition $N_{\text{GS}}([x]) \overset{\circ}{\subset} [x]$ is fulfilled. This condition is checked after each Gauss-Seidel step that results in a single interval vector. Otherwise, i.e. if splittings have been occurred, uniqueness cannot be proved. The method can also be applied in a separate verification step that uses the interval Newton Gauss-Seidel operator and the so-called ε-inflation to check on inner inclusion.

So, we have two possibilities to prove the uniqueness of a solution:

1. after each Gauss-Seidel step that is only applied if the N_{cmp}-operator results in one box,

2. by an additional verification step that is applied on each box which uniqueness has not yet been proved for by method 1.

The second step is similar to an algorithm from [2], but we made some changes with respect to the componentwise Newton method.

5 Algorithmic Description

The explained improvements on the componentwise method and the componentwise Newton operator have been combined to a PASCAL-XSC-module called `Ncmp_mod`. This module can reliably enclose all solutions of a system of nonlinear equations into narrow interval vectors. We describe the most important aspects of the algorithms, a more comprehensive description can be found in [4].

The user of the module can call the global procedure AllNcmp. The input data are the function f, the search-box $[x] \in I\!I\!R^n$, the tolerance ε and two further parameters *max_f* and *UseGSS* that control the behaviour of the module. The flag *UseGSS* tells whether the GaussSeidelStep is applied besides the CmpNewtonStep or only within the VerificationStep. The parameter *max_f* controls the length of the index-list L_1. More precisely, the componentwise Newton operator processes a variable x_j with at most *max_f* functions f_i, i.e. L_1 contains $n \cdot max\_f$ (or less) index-pairs.

The output data of AllNcmp are the interval matrix [*Sol*], containing the enclosures of the solutions row by row, the boolean vector *Info*, containing information on the local uniqueness of each solution, the number N of the computed enclosures and an error-flag *Err*).

Algorithm 5.1: AllNcmp($f, [x], \varepsilon, max\_f, UseGSS, [Sol], Info, N, Err$)

1. Err := CheckParameters;
2. **if** $Err \neq$ "No Error" **then return** Err;
3. $N := 0$;
4. **if** (ε is too small) **then** ε := "1 ulp accuracy" {check ε}
5. **if** ($max\_f \notin \{1, \ldots, n\}$) **then** $max\_f := 1$; {check max_f}
6. ComputeIndex($f, [x], max\_f, L_1, L_2$); {pointers at L_1 and L_2 are global!}
7. XINcmp($f, [x], \varepsilon, UseGSS, true, false, [Sol], Info, N$);
8. **return** $[Sol], Info, N, Err$;

The procedure ComputeIndex creates the two index-lists L_1 and L_2 as described in Section 4.1. It needs the function f, the search-box $[x] \in I\!I\!R^n$, and the parameter max_f.

Algorithm 5.2: ComputeIndex$(f, [x], max_f, L_1, L_2)$

1. $L_1 := \{\}$; $L_2 := \{\}$; {initialization by empty lists}
2. $[J] := J_{\diamond f}([x])$
3. **for** $j := 1$ **to** n **do** {L_1 used by CmpNewtonStep without ext. interval arithmetic}
   (a) $k := 0$;
   (b) **for** $i := j, \ldots, n, 1, \ldots, j-1$ **do**
   if $([J]_{i,j} \neq 0)$ **and** $(k \leq max\_f)$ **then**
   $L_1 := L_1 \uplus (i, j)$; $k := k + 1$;
4. **for** $j := 1$ **to** n **do** {L_2 used by CmpNewtonStep with ext. interval arithmetic}
   (a) $\hat{\imath} := 0$;
   (b) determine $\hat{\imath}$ with $d([J]_{\hat{\imath}j}) = \max\limits_{i=1,\ldots,n} \{d([J]_{ij}) \mid 0 \in [J]_{ij}\}$
   (c) **if** $(\hat{\imath} \neq 0)$ **and** $(d([J]_{\hat{\imath}j}) > 0)$ **then**
   $L_2 := L_2 \uplus (\hat{\imath}, j)$;

The procedure XINcmp is the central part of the module. It is the realization of a branch-and-prune-method. The pruning can be done by both the CmpNewtonStep (Step 2) and the GaussSeidelStep (Step 3(b)ii). The latter may be enabled or disabled by the flag *UseGSS*.

If a box satisfies the desired accuracy (relative diameter) specified by the input parameter ε and has not yet been found to contain a unique solution the VerificationStep is called (Algorithm 5.5). It may happen that an empty intersection occurs in the VerificationStep (i.e. in the called GaussSeidelStep), in that case the current box is thrown away. Otherwise, the box is stored as a solution together with the information on its uniqueness.

Algorithm 5.3: XINcmp$(f, [y], \varepsilon, UseGSS, UseL2, uniq, [Sol], Info, N)$

1. **if** $(0 \notin f([y]))$ **then return** ;
2. CmpNewtonStep$(f, [y], \varepsilon, UseL2, [Y_p], p)$;
3. **if** $(p = 1)$ **then**
   (a) $noBisect := ([Y_p]_1 \overset{\circ}{\subset} [y])$;
   (b) **if** $UseGSS$ **and** $(d_{rel,\infty}([Y_p]_1) > \varepsilon)$ **then**
   i. $[y_{\text{old}}] := [Y_p]_1$;
   ii. GaussSeidelStep$(f, [Y_p]_1, q)$;
   iii. **if** $(q = 0)$ **then return** ;

iv. **if** $(q = 1)$ **then**
$\quad$ *InnerIncl* := $([Y_p]_1 \overset{\circ}{\subset} [y_{\text{old}}])$;
$\quad$ *uniq* := *uniq* **or** *InnerIncl*;
$\quad$ *noBisect* := *noBisect* **or** *InnerIncl*;
$\quad$ **else**
$\quad\quad$ *uniq* := *false*;

else $\{p \neq 1\}$
$\quad$ *noBisect* := *false*; *uniq* := *false*;

4. **for** $i := 1$ **to** p **do**

(a) **if** $(d_{rel,\infty}([Y_p]_i) \leq \varepsilon)$ **then**
$\quad$ **if** $(0 \in f([Y_p]_i))$ **then**
$\quad\quad$ **if not** *uniq* **then** VerificationStep$(f, [Y_p]_i, uniq)$;
$\quad\quad$ **if** $([Y_p]_i \neq [/])$ **then**
$\quad\quad\quad$ $N := N + 1$; $[Sol]_N := [Y_p]_i$; $[Info]_N := uniq$;

(b) **else**
$\quad$ **if not** *noBisect* **then**
$\quad\quad$ Split$([Y_p]_i, [y_l], [y_r])$; $\{[Y_p]_i = [y_l] \cup [y_r]\}$
$\quad\quad$ XINcmp$(f, [y_l], \varepsilon, UseGSS, UseL2, uniq, [Sol], Info, N)$;
$\quad\quad$ XINcmp$(f, [y_r], \varepsilon, UseGSS, UseL2, uniq, [Sol], Info, N)$;
$\quad$ **else** {recursive call of XINcmp for $[Y_p]_i$}
$\quad\quad$ XINcmp$(f, [Y_p]_i, \varepsilon, UseGSS, UseL2, uniq, [Sol], Info, N)$;

5. **return** $[Sol]$, N;

The simplified Gauss-Seidel step (see Section 4.2) is described as Algorithm 5.4. Once more, we stress the importance of the principle "as rarely as possible but as often as necessary" for applying GaussSeidelStep. We tried to realize this in XINcmp, but there will always be a certain amount of heuristical methods.

If the Gauss-Seidel step leads to a union of two non-empty intervals, the procedure is stopped (see Step 5(d)) and the current box is not changed. Moreover, the procedure GaussSeidelStep can deliver useful information if an empty intersection occurs. Then it is proved that there is no solution in $[y]$ and the recursive call of XINcmp is stopped by setting $q = 0$ (see Steps 4(c) and 5(c)).

Algorithm 5.4: GaussSeidelStep$(f, [y], q)$

1. MatInv$(m(J_f([y])), R, InvErr)$; {invert midpoint matrix of the Jacobian}
2. **if** $(InvErr \neq$ "No Error"$)$ **then** $R := I$;
3. $c := m([y])$; $[A] := R \cdot J_f([y])$; $[b] := R \cdot f_\diamond(c)$; $[y_c] := [y] - c$; {initializations}
4. **for** $i := 1$ **to** n **do** {interval Gauss-Seidel step for $0 \notin [A]_{ii}$}
   (a) **if** $(0 \in [A]_{ii})$ **then next**$_i$;

(b) $[y]_i := \left(c_i - \left([b]_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} [A]_{ij} \cdot [y_c]_j\right) \Big/ [A]_{ii}\right) \cap [y]_i$;

(c) **if** $[y]_i = [/]$ **then return** $q = 0$;

(d) $[y_c]_i := [y]_i - c_i$;

5. **for** $i := 1$ **to** n **do** {interval Gauss-Seidel step for $0 \in [A]_{ii}$}

(a) **if** $(0 \notin [A]_{ii})$ **then next**$_i$;

(b) $[z] := \left(c_i - \left([b]_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} [A]_{ij} \cdot [y_c]_j\right) \Big/ [A]_{ii}\right) \cap [y]_i$; {$[z] = [z]_1 \cup [z]_2$}

(c) **if** $([z] = [/])$ **then return** $q = 0$;

(d) **if** $([z]_2 \neq [/])$ **then return** $[y], q = 2$;

(e) $[y]_i := [z]_1$; $[y_c]_i := [y]_i - c_i$;

6. **return** $[y], q = 1$;

Algorithm 5.5: VerificationStep$(f, [y], Unique)$

1. $k_{\max} := 5$; $k := 0$; $[y_{\text{in}}] := [y]$; $\varepsilon := 0.25$; $Unique := false$; {initializations}

2. **while** (**not** $Unique$) **and** $(k < k_{\max})$ **do** {do $k_{\max}$ loops at most}

(a) $[y_{\text{old}}] := [y] \bowtie \varepsilon$; {$\varepsilon$-inflation}

(b) $k := k + 1$; $[y] := [y_{\text{old}}]$;

(c) GaussSeidelStep$(f, [y], p)$;

(d) **if** $p > 1$ **then exit**$_{\text{while-loop}}$; {no verification possible}

(e) **if** $p = 0$ **then return** $[y] = [/]$; {no soulition in $[y]$}

(f) **if** $[y] = [y_{\text{old}}]$ **then**
$\varepsilon := \varepsilon \cdot 8$; {increase ε}
else
$Unique := ([y] \overset{\circ}{\subset} [y_{\text{old}}])$; {inner inclusion $\Longrightarrow$ uniqueness}

3. **if not** $Unique$ **then** $[y] := [y_{\text{in}}]$; {reset $[y]$ to starting value}

4. **return** $[y], Unique$;

The procedure CmpNewtonStep (Algorithm 5.6) is most important for the efficiency of the module. We use the index-lists L_1 and L_2 to control the order of pairs (i, j) the componentwise Newton operator is applied on. First, we pass through the list L_1 and the N_{cmp}-operator is only applied if the required Jacobian element does not equal zero. After that, the list L_2 is used to apply the componentwise Newton operator on those pairs for which we must use the extended interval arithmetic. In either case, an empty intersection during the calculation of N_{cmp} makes the CmpNewtonStep stop, proving that there is no solution within the current box $[y]$.

To reduce the number of evaluations we introduced the variable *UseL2*. This flag indicates whether CmpNewtonStep goes through the list L_2 or not. Often, the elements of the Jacobian matrix listed in L_2 do not contain zero after several iterations. Hence, the part of the procedure CmpNewtonStep that uses the list L_2 and extended interval arithmetic does not apply the Newton operator any more. If the evaluation of the Jacobian over the current interval vector $[y]$ does not contain zero in the L_2-components the flag *UseL2* is set to *false*. Thus, we avoid unnecessary evaluations of partial derivatives in later iterations for the box $[y]$.

Algorithm 5.6: CmpNewtonStep$(f, [y], \varepsilon, \mathit{UseL2}, [V], p)$

1. $p := 0$; {initialization}
2. **for all** $(i,j) \in L_1$ **do**
   (a) **if** $(0 \in [J_f([y])]_{ij})$ **then next**$_{(i,j)}$;
   (b) $c := m([y]_j)$; $[y_h] := [y]$; $[y_h]_j := c$;
   (c) $[y]_j := \left(c - [f_i([y_h])] \;/\; [J_f([y])]_{ij} \right) \cap [y]_j$;
   (d) **if** $([y]_j = [/])$ **then return** $p = 0$;
3. **if not** *UseL2* **or** $(d_{rel,\infty}([y]) \leq \varepsilon)$ **then goto** 6.
4. $\mathit{UseL2} := \mathit{false}$;
5. **for all** $(i,j) \in L_2$ **do**
   (a) **if** $(0 \notin [J_f([y])]_{ii})$ **then next**$_{(i,j)}$;
   (b) $\mathit{UseL2} := \mathit{true}$;
   (c) **if** $(0 \in [f_i([y_h])])$ **then next**$_{(i,j)}$; {ext. interval division $\Longrightarrow (-\infty, +\infty)$!}
   (d) $c := m([y]_j)$; $[y_h] := [y]$; $[y_h]_j := c$;
   (e) $[z] := \left(c - [f_i([y_h])] \;/\; [J_f([y])]_{ij} \right) \cap [y]_j$; {$[z] = [z]_1 \cup [z]_2$}
   (f) **if** $([z] = [/])$ **then return** $[V],\ p,\ \mathit{UseL2}$
   (g) $[y]_j := [z]_1$
   (h) **if** $([z]_2 \neq [/])$ **then**
   $p := p + 1$; $[V]_p := [y]$; $[V]_{pj} := [z]_2$
6. $p := p + 1$; $[V]_p := [y]$; {store current box}
7. **return** $[V],\ p,\ \mathit{UseL2}$;

6 Some Properties of AllNcmp

The global procedure AllNcmp from the module `Ncmp_mod` has some important properties which can be derived from the structure of the algorithms and the properties of the Newton operators.

Theorem 6.1 *The algorithm* AllNcmp *(5.1) terminates after a finite number of steps.*

Proof: The tolerance ε cannot be chosen smaller than the machine accuracy. Thus, the number of bisections and, consequently, the number of recursive calls are limited. All other procedures called by AllNcmp or XINcmp, respectively, only do a finite number of steps. □

Theorem 6.2 *Let $f : D \subseteq I\!R^n \to I\!R^n$ be a continuously differentiable function and let $[x] \in I\,I\!R^n$ be an interval vector with $[x] \subseteq D$. Then the solution set X^* of the system of nonlinear equations $f(x) = 0$ satisfies:*

$$X^* := \{x^* \in [x] \mid f(x^*) = 0\} \subseteq \bigcup_{i=1}^{N} [Sol]_i.$$

Proof: From Theorem 3.2 we know that no solution x^* can be lost when applying the componentwise Newton operator. For arbitrary $i, j \in \{1, \ldots, n\}$ and $[y] \subseteq [x]$ the following inclusion holds:

$$x^* \in [y] \overset{3.2}{\Longrightarrow} x^* \in ([y]_1, \ldots, [y]_{j-1}, N_{\mathrm{cmp}}([y], i, j) \cap [y_j], [y]_{j+1}, \ldots, [y]_n).$$

The same property holds for the Gauss-Seidel step (see Theorem 4.2) that may always (not depending on *UseGSS*) be called by the VerificationStep. We have:

$$x^* \in [y] \overset{4.2}{\Longrightarrow} x^* \in N_{\mathrm{GS}}([y]).$$

Any calculation using interval arithmetic leads to enclosures, hence it is guaranteed that all "candidates" for a solution are stored in $[Sol]$. And so, the union of all $[Sol]_i$ must be a superset of the solution set X^*. □

The described algorithm divides the search-box $[x] \in I\,I\!R^n$ successively into smaller sub-boxes $[y]^{(l)} \subseteq [x]$ that may contain one or more zeros of the problem. Concerning the preceding consideration, no solution $x^* \in X^*$ can be lost during this process. Every box that is thrown away reliably contains no solution. So, all boxes $[y]^{(l)}$ from the (current) ith call of the procedure XINcmp plus all boxes $[y]^{(l)}$ from former recursive calls, which have not yet been treated, may still contain zeros. Let us call these boxes "active candidates", and let

$$\mathcal{I}_i := \left\{ l \in I\!N \mid [y]^{(l)} \text{ is an "active candidat" in the } i\text{th step} \right\}$$

be the corresponding index-set. All active candidates in the ith step are in the set

$$\mathcal{K}_i := \bigcup_{l \in \mathcal{I}_i} [y]^{(l)}.$$

Theorem 6.3 *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function and let $[x] \in I\mathbb{R}^n$ be an interval vector with $[x] \subseteq D$. Moreover, let all calculations be real (i.e. exact), and let the tolerance be $\varepsilon = 0$. Let each point $z \in D$ and every sequence $([z]^{(k)})$ with $[z]^{(k)} \to z$ and $[z]^{(k)} \subseteq D$ satisfy $f([z]^{(k)}) \to f(z)$. Then we have:*

$$X^* = \bigcap_{i=1}^{\infty} \mathcal{K}_i.$$

Proof: As mentioned above, no zero x^* can be lost, i.e. each point $x^* \in X^*$ belongs to every set $\mathcal{K}_i$. Hence,

$$X^* \subseteq \bigcap_{i=1}^{\infty} \mathcal{K}_i$$

holds.
On the other hand, let $\hat{x} \in \bigcap_{i=1}^{\infty} \mathcal{K}_i$, then there exists a sequence $([y]^{(i)})$ with $[y]^{(i)} \subseteq \mathcal{K}_i$ and $\hat{x} \in [y]^{(i)}$ for all i. According to the construction of the algorithm and because of the preliminaries ($\varepsilon = 0$ and exact calculation), we have:

$$d([y]^{(i)}) \to 0 \text{ for } i \to \infty, \quad \text{and hence,} \quad [y]^{(i)} \to \hat{x} \text{ for } i \to \infty.$$

The sequence of the function values satisfies (according to the preliminaries):

$$f([y]^{(i)}) \to f(\hat{x}) \text{ for } i \to \infty.$$

The point $\hat{x}$ belongs to each $\mathcal{K}_i$, therefore, $f(\hat{x}) = 0$ holds and we have $\hat{x} \in X^*$, i.e.

$$\bigcap_{i=1}^{\infty} \mathcal{K}_i \subseteq X^*.$$

□

7 Examples and Results

Now, we give some numerical examples. All results were obtained by running the modules on an Intel-Pentium-based Personal Computer with a 75 MHz CPU. The source code has been compiled using a PASCAL-XSC-compiler, Version 2.03, and a ZORTECH C 3.0 C-compiler on a DOS operating-system.

The results of our module `Ncmp_mod` (with four different parameter selections) are compared to the module `nlss` from [2]. The results are summarized in tables where we use the following abbreviations.

Method used solving-method. `TBNLSS` is the module `nlss` from [2] and `Ncmp` marks the componentwise method followed by the value of the parameter *max_f* (here: 1 or n) and the flag *UseGSS*. If the Gauss-Seidel step may be called by the procedure `XINcmp` we write a '`+`'. A '`-`' shows that no extra Gauss-Seidel step was executed.

Time time (in seconds if no different unit is given) the global procedures AllNLSS and AllNcmp, respectively, were active.

Encls number of enclosures found.

Unique number of enclosures proved to contain locally unique solutions.

FcEv number of evaluations of a single component of the function (after an evaluation of the whole function this counter is increased by the dimension n of the systems).

JcEv number of evaluations of a single component of the the Jacobian matrix (an evaluation of the whole Jacobian leads to an increase of 'JcEv' by n^2).

Ncmp number of CmpNewtonStep-calls.

GS number of calls of the interval Newton Gauss-Seidel step (including the calls by the verification step).

Bisect number of bisections executed by XINewton or XINcmp, respectively.

We do not list the found enclosures, they are not very useful for the comparison of the different methods. Sometimes, a certain method gives more enclosures than other ones. Usually, the boxes are pruned and bisected in different ways. The usual overestimation of the range of a function may cause that a box is stored as a "candidate" although it does not contain any zero. Moreover, a poor bisection may result in two boxes containing the same zero. Hence, only the number of enclosures is really interesting in the current background.

We start with some "traditional problems" from several interval arithmetic papers.

Example 7.1 The system

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0 \\ x_1^2 - x_2 &= 0 \end{aligned}$$

can be found in many papers (e.g. [5]). It shows the efficiency of our module searching zeros in a very large starting box.

Starting box: `[ -1.0E+008 ,  1.0E+008 ]`^{2}, tolerance: $\varepsilon =$ `1.0E-008`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	0.50	0.22	0.16	0.22	0.17
Encls	2	2	2	2	2
Unique	2	2	2	2	2
FcEv	686	455	425	225	225
JcEv	1004	266	352	230	276
Ncmp	—	91	75	31	29
GS	90	2	2	29	27
Bisect	123	69	53	21	21

Starting box: `[ -1.0E+016 , 1.0E+016 ]`^{2}, tolerance: $\varepsilon =$ `1.0E-008`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	0.88	0.28	0.27	0.33	0.33
Encls	2	2	2	2	2
Unique	2	2	2	2	2
FcEv	1230	631	599	397	407
JcEv	1812	380	518	408	494
Ncmp	—	121	105	53	53
GS	160	2	2	51	49
Bisect	224	99	81	43	43

Example 7.2 The so-called Feigenbaum-example

$$\begin{aligned} -3.84x_k^2 + 3.84x_k - x_{k+1} &= 0, \quad k = 1, \ldots, n-1 \\ -3.84x_n^2 + 3.84x_n - x_1 &= 0 \end{aligned}$$

demonstrates the behaviour of the methods on increasing dimension n.

Starting box: `[ 0.0E+000 , 1.0E+002 ]`^{3}, tolerance: $\varepsilon =$ `1.0E-010`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	1.87	1.54	2.36	0.88	0.99
Encls	8	10	10	8	8
Unique	8	10	10	8	8
FcEv	1737	3873	5237	886	1142
JcEv	3087	1977	3355	1118	1353
Ncmp	—	670	670	105	105
GS	228	16	16	74	71
Bisect	162	343	338	55	55

Starting box: `[ 0.0E+000 , 1.0E+002 ]`^{5}, tolerance: $\varepsilon =$ `1.0E-010`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	48.55	4.18	6.42	5.06	5.49
Encls	12	12	12	12	12
Unique	12	12	12	12	12
FcEv	33830	10014	13645	4330	5554
JcEv	105550	5714	9457	7089	7966
Ncmp	—	1051	1043	341	339
GS	2532	12	12	182	166
Bisect	2097	467	436	188	179

Example 7.3 "BROWN's almost linear function" is another system that can be considered for arbitrary dimensions.

$$\begin{aligned} x_k + \sum_{j=1}^{n} x_j - (n+1), &= 0, \quad k = 1, \ldots, n-1 \\ \Big(\prod_{j=1}^{n} x_j\Big) - 1 &= 0 \end{aligned}$$

Starting box: `[ -1.0E+001 ,  1.0E+001 ]`^{3}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	1.92	4.06	5.22	1.05	1.04
Encls	7	5	6	4	4
Unique	7	5	6	4	4
FcEv	1899	11025	11990	1012	1315
JcEv	3861	5201	8696	1279	1541
Ncmp	—	1570	1063	108	94
GS	197	37	28	92	73
Bisect	209	1157	607	63	51

Starting box: `[ -1.0E+001 ,  1.0E+001 ]`^{4}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	19.17	69.37	113.03	6.54	7.69
Encls	2	5	9	2	2
Unique	2	5	9	2	2
FcEv	17048	177432	234981	4859	8289
JcEv	50880	82554	181807	7881	10864
Ncmp	—	19312	13088	383	358
GS	1080	372	254	371	307
Bisect	1588	14584	7391	222	196

Here, the N_{cmp}-operator itself (`Ncmp(1-)` and `Ncmp(n-)`) does not work very well, it needs a lot of evaluations and bisection, and using the full index-list is even worse. But, by the combination with the Gauss-Seidel step (`Ncmp(1+)` and `Ncmp(n+)`) our module is much better than the module `nlss` that uses the Gauss-Seidel method exclusively.

Example 7.4 This function is derived from the discretization of a boundary value problem (see [10])

$$2x_k - x_{k+1} - x_{k-1} + \frac{h^2}{2}(x_k + kh + 1)^3, \quad k = 1, \ldots, n$$

where $x_0 = x_{n+1} = 0$ and $h = \frac{1}{n+1}$. Especially for a large n, the nonlinear term becomes very small. Hence, we have another almost linear system. This example shows a limitation of our module, the (preconditioned) Gauss-Seidel method works very well on that kind of problems.

Starting box: `[ -5.0E-001 , 0.0E+000 ]`^{5}, tolerance: $\varepsilon =$ `1.0E-005`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	0.05	0.28	0.60	0.11	0.11
Encls	1	1	1	1	1
Unique	1	1	1	1	1
FcEv	35	390	694	50	74
JcEv	75	240	544	115	139
Ncmp	—	38	38	3	3
GS	3	1	1	3	3
Bisect	0	0	0	0	0

Starting box: `[ -5.0E-001 , 0.0E+000 ]`^{10}, tolerance: $\varepsilon =$ `1.0E-005`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	0.22	34.50	51.63	0.38	0.44
Encls	1	5	5	1	1
Unique	1	5	5	1	1
FcEv	70	51405	64832	100	154
JcEv	300	25142	46592	430	484
Ncmp	—	2456	2040	3	3
GS	3	39	27	3	3
Bisect	0	1736	1030	0	0

Example 7.5 According to KEARFOTT (see [6]) POWELL's singular function is a "severe test of most methods".

$$\begin{aligned} x_1 + 10x_2 &= 0 \\ \sqrt{5}\cdot(x_3 - x_4) &= 0 \\ (x_2 - 2x_3)^2 &= 0 \\ \sqrt{10}\cdot(x_1 - x_4)^2 &= 0 \end{aligned}$$

The system has the (only) solution $x^* = (0,0,0,0)^\mathsf{T}$, and in that point the Jacobian matrix is singular.

Starting box: `[ -1.0E+000 , 1.0E+000 ]`^{4}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	12.14	2.75	0.49	5.11	1.21
Encls	13	16	4	14	4
Unique	0	0	0	0	0
FcEv	15104	6371	1051	6186	1221
JcEv	48064	4504	912	7312	1776
Ncmp	—	533	70	415	67
GS	759	16	4	251	60
Bisect	1495	723	74	569	71

Starting box: `[ -5.0E-001 , 1.0E+000 ]`^{4}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	>999	1.76	0.33	3.79	0.38
Encls	?	1	1	1	1
Unique	?	0	0	0	0
FcEv	?	4086	977	4523	984
JcEv	?	2840	560	5268	592
Ncmp	—	351	44	303	44
GS	?	1	1	177	3
Bisect	?	451	99	449	99

This example demonstrates the very good results on sparse systems. Moreover, the usage of the full index-list improves the componentwise method enormously. We can also notice a strong influence of the choice of the starting box. But, this is not very astonishing because in the first case the starting box is bisected at 0 in each component, hence, the solution is included in both of the sub-boxes. Therefore, the number of found enclosures is rather high. With the second search box the `Ncmp_mod`-module is much better, while the `nlss`-module is even worse.

Example 7.6 The following system is a representative of a set of examples given by MOORE and JONES (see [9]).

$$\begin{aligned}
x_1 - 0.25428722 - 0.18324757\, x_4x_3x_9 &= 0\\
x_2 - 0.37842197 - 0.16275449\, x_1x_{10}x_6 &= 0\\
x_3 - 0.27162577 - 0.16955071\, x_1x_2x_{10} &= 0\\
x_4 - 0.19807914 - 0.15585316\, x_7x_1x_6 &= 0\\
x_5 - 0.44166728 - 0.19950920\, x_7x_6x_3 &= 0\\
x_6 - 0.14654113 - 0.18922793\, x_8x_5x_{10} &= 0\\
x_7 - 0.42937161 - 0.21180486\, x_2x_5x_8 &= 0\\
x_8 - 0.07056438 - 0.17081208\, x_1x_7x_6 &= 0\\
x_9 - 0.34504906 - 0.19612740\, x_{10}x_6x_8 &= 0\\
x_{10} - 0.42651102 - 0.21466544\, x_4x_8x_1 &= 0
\end{aligned}$$

Starting box: `[ 0.0E+000 , 2.0E+000 ]`^{10}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	2.41	0.11	0.27	0.22	0.28
Encls	1	1	1	1	1
Unique	1	1	1	1	1
FcEv	1450	120	250	90	160
JcEv	12500	260	410	340	430
Ncmp	—	5	5	3	3
GS	19	1	1	2	2
Bisect	61	0	0	0	0

Starting box: `[ -2.0E+000 , 2.0E+000 ]`^{10}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	54.38	0.16	0.27	0.33	0.39
Encls	1	1	1	1	1
Unique	1	1	1	1	1
FcEv	21840	130	235	110	163
JcEv	155200	270	420	450	540
Ncmp	—	5	5	3	3
GS	631	1	1	3	3
Bisect	775	0	0	0	0

We point at the fact that our algorithm does not make any bisection. With the enlarged starting box the Toolbox-module has severe problems on finding an enclosure.

We continue by presenting some results for systems containing transcendental functions; they are taken from [7].

Example 7.7

$$\begin{aligned} 1 - 2x_2 + 0.05\sin(4\pi x_2) - x_1 &= 0 \\ x_2 - 0.5\sin(2\pi x_1) &= 0 \end{aligned}$$

Starting box: `[ -1.0E+001 , 1.0E+001 ]`^{2}, tolerance: $\varepsilon =$ `1.0E-008`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	1.48	7.25	2.64	1.48	1.27
Encls	6	5	5	5	5
Unique	6	5	5	5	5
FcEv	446	1749	519	261	225
JcEv	616	735	356	252	202
Ncmp	—	341	79	39	27
GS	63	7	5	37	19
Bisect	71	292	23	16	15

Example 7.8

$$\begin{aligned} x_1^2 - x_2 + 1 &= 0 \\ x_1 - \cos(\frac{\pi}{2}x_2) &= 0 \end{aligned}$$

Starting box: `[ -3.0E+000 , 3.0E+000 ]`^{2}, tolerance: $\varepsilon =$ `1.0E-008`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	0.61	2.69	1.43	0.77	0.60
Encls	5	5	4	4	3
Unique	5	5	4	4	3
FcEv	264	1071	509	207	160
JcEv	324	690	344	236	174
Ncmp	—	219	77	31	20
GS	46	5	4	29	18
Bisect	37	192	52	20	9

Example 7.9

$$\begin{aligned} 2\sin(2\pi x_1/5)\cdot\sin(2\pi x_3/5) - x_2 &= 0 \\ 2.5 - x_3 + 0.1x_2\sin(2\pi x_3) - x_1 &= 0 \\ 1 + 0.1x_2\sin(2\pi x_1) - x_3 &= 0 \end{aligned}$$

Starting box: `[ -1.0E+000 , 3.0E+000 ]`^{3}, tolerance: $\varepsilon =$ `1.0E-006`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	3.24	19.72	4.33	2.09	1.65
Encls	4	3	2	1	1
Unique	4	3	2	1	1
FcEv	630	2589	474	181	142
JcEv	1251	1875	513	276	213
Ncmp	—	365	48	19	12
GS	67	6	2	17	9
Bisect	67	321	24	13	6

Similar to the behaviour on almost linear systems, the componentwise method without the Gauss-Seidel operator does not work very well. The `Ncmp(n+)`-method is better than the `TBNLSS` because the latter gives too many enclosures for these examples. The number of enclosures in the rightmost column always equals the correct number of solutions.

We close this section looking at some "real-world" problems. The results show again: by the combination of the componentwise operator with a sufficiently long index-list and supported by the Gauss-Seidel operator we get very high efficiency.

Example 7.10 The robot kinematics problem has 16 solutions within the starting box (see [6])

$$\begin{aligned} a_1x_1x_3 + a_2x_2x_3 + a_3x_1 + a_4x_2 + a_5x_4 + a_6x_7 + a_7 &= 0 \\ a_8x_1x_3 + a_9x_2x_3 + a_{10}x_1 + a_{11}x_2 + a_{12}x_4 + a_{13} &= 0 \\ a_{14}x_6x_8 + a_{15}x_1 + a_{16}x_2 &= 0 \\ a_{17}x_1 + a_{18}x_2 + a_{19} &= 0 \end{aligned}$$

$$\begin{aligned} x_1^2 + x_2^2 - 1 &= 0 \\ x_3^2 + x_4^2 - 1 &= 0 \\ x_5^2 + x_6^2 - 1 &= 0 \\ x_7^2 + x_8^2 - 1 &= 0 \end{aligned}$$

where $a_1 = 4.731 \cdot 10^{-3}$, $a_2 = -0.3578$, $a_3 = -0.1238$, $a_4 = -1.637 \cdot 10^{-3}$, $a_5 = -0.9338$, $a_6 = 1.0$, $a_7 = -0.3571$, $a_8 = 0.2238$, $a_9 = 0.7623$, $a_{10} = 0.2638$, $a_{11} = -0.7745 \cdot 10^{-1}$, $a_{12} = -0.6734$, $a_{13} = -0.6022$, $a_14 = 1.0$, $a_{15} = 0.3578$, $a_{16} = 4.731 \cdot 10^{-3}$, $a_{17} = -0,7623$, $a_{18} = 0.2238$, $a_{19} = 0.3461$

Starting box: `[ -1 , 1 ]`^{8}, tolerance: $\varepsilon =$ `1E-8`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	215.80	???	10.87	26.81	7.80
Encls	16	?	16	16	16
Unique	16	?	16	16	16
FcEv	131272	?	18025	12361	4849
JcEv	794048	?	14643	37872	10675
Ncmp	—	?	573	482	125
GS	3986	?	16	474	110
Bisect	6192	?	284	314	68

Example 7.11 Another kinematics problem is taken from [5].

$$\begin{aligned} -x_3x_{10}x_{11} - x_5x_{10}x_{11} - x_7x_{10}x_{11} + x_4x_{12} + x_6x_{12} + x_8x_{12} - 0.4077 &= 0 \\ x_2x_4x_9 + x_2x_6x_9 + x_2x_8x_9 + x_1x_{10} - 1.9115 &= 0 \\ x_3x_9 + x_5x_9 + x_7x_9 - 1.9791 &= 0 \\ 3x_2x_4 + 2x_2x_6 + x_2x_8 - 4.0616 &= 0 \\ 3x_1x_4 + 2x_1x_6 + x_1x_8 - 1.7172 &= 0 \\ 3x_3 + 2x_5 + x_7 - 3.9701 &= 0 \\ x_1^2 + x_2^2 - 1 &= 0 \\ x_3^2 + x_4^2 - 1 &= 0 \\ x_5^2 + x_6^2 - 1 &= 0 \\ x_7^2 + x_8^2 - 1 &= 0 \\ x_9^2 + x_{10}^2 - 1 &= 0 \\ x_{11}^2 + x_{12}^2 - 1 &= 0 \end{aligned}$$

Starting box: `[ 0 , 1 ]`^{12}, tolerance: $\varepsilon =$ `1E-6`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	2241	???	???	84	44
Encls	2	?	?	2	2
Unique	2	?	?	2	2
FcEv	1032084	?	?	21612	17595
JcEv	10810800	?	?	94408	49419
Ncmp	—	?	?	604	281
GS	10930	?	?	579	253
Bisect	37536	?	?	366	149

Starting box: `[ -1 , 1 ]`^{12}, tolerance: $\varepsilon =$ `1E-6`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	2:47 h	???	???	0:09 h	0:04 h
Encls	16	?	?	16	16
unique	16	?	?	16	16
FcEv	4532376	?	?	141350	101420
JcEv	46867824	?	?	626263	287835
Ncmp	—	?	?	3939	1689
GS	52209	?	?	3807	1453
Bisect	162726	?	?	2607	904

Example 7.12 A combustion chemistry problem (see [6])

$$\begin{aligned} a_1x_2x_4 + a_2x_2 + a_3x_1x_4 + a_4x_1 + a_5x_4 &= 0 \\ b_1x_2x_4 + b_2x_1x_3 + b_3x_1x_4 + b_4x_3x_4 + b_5x_3 + b_6x_4 + b_7 &= 0 \\ x_1^2 - x_2 &= 0 \\ x_4^2 - x_3 &= 0 \end{aligned}$$

where $a_1 = -1.697 \cdot 10^7$, $a_2 = 2.177 \cdot 10^7$, $a_3 = 0.55$, $a_4 = 0.45$, $a_5 = -1.0$,
$b_1 = 1.585 \cdot 10^{14}$, $b_2 = 4.126 \cdot 10^7$, $b_3 = -8.285000 \cdot 10^6$, $b_4 = 2.284 \cdot 10^7$,
$b_5 = -1.918 \cdot 10^7$, $b_6 = 48.4$, $b_7 = -27.73$

Starting box: `[ 0.0E+000 , 1.0E+001 ]`^{4}, tolerance: $\varepsilon =$ `1.0E-008`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	9.23	88.65	2.36	3.57	1.70
Encls	1	2	1	1	1
Unique	1	2	1	1	1
FcEv	6208	149399	4012	2429	1323
JcEv	14992	107416	3201	4140	2133
Ncmp	—	14327	281	178	77
GS	614	14	3	178	67
Bisect	467	14301	191	129	49

Example 7.13 Modelling the combustion of propane (see [8])

$$\begin{aligned}
x_1x_2 + x_1 - 3x_5 &= 0\\
2x_1x_2 + x_1 + x_2x_3^2 + R_8x_2 - Rx_5 + 2R_{10}x_2^2 + R_7x_2x_3 + R_9x_2x_4 &= 0\\
2x_2x_3^2 + 2R_5x_3^2 - 8x_5 + R_6x_3 + R_7x_2x_3 &= 0\\
R_9x_2x_4 + 2x_4^2 - 4Rx_5 &= 0\\
x_1x_2 + x_1 + R_{10}x_2^2 + x_2x_3^2 + R_8x_2 + R_5x_3^2 + x_4^2 - 1 + R_6x_3 + R_7x_2x_3 + R_9x_2x_4 &= 0
\end{aligned}$$

where $R = 10$, $R_5 = 0.193$, $R_6 = 0.002597/\sqrt{40}$, $R_7 = 0.003448/\sqrt{40}$,
$R_8 = 0.00001799/40$, $R_9 = 0.0002155/\sqrt{40}$, $R_{10} = 0.00003846/40$

Starting box: `[ 0 , 1E+8 ]`^{5}, tolerance: $\varepsilon =$ `1E-6`

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	> 8 h	> 30 min	525 sec	> 30 min	402 sec
Encls	?	?	1	?	1
Unique	?	?	1	?	1
FcEv	?	?	680684	?	275124
JcEv	?	?	534783	?	400313
Ncmp	—	?	34769	?	10927
GS	?	?	380	?	7859
Bisect	?	?	18024	?	5997

Example 7.14 According to Van Hentenryck, McAllester and Kapur a "difficult economic modelling problem" (see [14]).

$$\begin{aligned}
\Big(x_k + \sum_{j=1}^{n-k-1} x_jx_{j+k}\Big) \cdot x_n - c_k &= 0, \quad k = 1, \ldots, n-1,\\
\sum_{j=1}^{n-1} x_j + 1 &= 0
\end{aligned}$$

The constants c_k may be chosen at random.

Starting box: `[ -1.0E+001 , 1.0E+001 ]`^{4}, tolerance: $\varepsilon =$ `1E-4`
$c_1 = 1.5$, $c_2 = -0.4$, $c_3 = 2.0$, $c_4 = 1.83$

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	79.26	11.43	2.30	13.35	2.31
Encls	2	3	4	2	2
Unique	2	3	4	2	2
FcEv	79764	26030	4407	10359	2360
JcEv	254320	14543	3683	16577	3330
Ncmp	—	2418	313	752	133
GS	4044	17	8	685	89
Bisect	7946	2428	182	677	84

Starting box: `[ -5.0E+000 , 8.0E+000 ]`^{4}, tolerance: $\varepsilon =$ `1E-4`
$c_1 = 1.5$, $c_2 = -0.4$, $c_3 = 2.0$, $c_4 = 1.83$

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	32.90	5.93	2.59	8.45	2.31
Encls	2	5	3	2	2
Unique	2	5	3	2	2
FcEv	37620	13556	4873	6792	2230
JcEv	129072	7038	4219	10527	3489
Ncmp	—	1271	360	489	142
GS	1336	14	9	427	92
Bisect	4032	1204	213	461	91

Starting box: `[ -4.0E+000 , 4.0E+000 ]`^{5}, tolerance: $\varepsilon =$ `1E-4`
$c_1 = 1.5$, $c_2 = -0.4$, $c_3 = 2.0$, $c_4 = 1.83$, $c_5 = -2.1$

Method	TBNLSS	Ncmp(1-)	Ncmp(n-)	Ncmp(1+)	Ncmp(n+)
Time	1155.79	31.86	22.13	50.32	13.67
Encls	2	7	10	2	2
Unique	2	7	10	2	2
FcEv	1027215	68847	39752	32886	11840
JcEv	4338250	38174	33131	60914	19202
Ncmp	—	5801	2063	1949	511
GS	31911	58	26	1709	339
Bisect	86764	4780	1164	1654	307

References

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

[2] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D.: *Numerical Toolbox for Verified Computing I: Basic Numerical Problems.* Springer-Verlag, Berlin/Heidelberg/New York (1993)

[3] Hansen, E.R.; Sengupta, S.: *Bounding Solutions of Systems of Equations Using Interval Analysis.* BIT **21**, 203–211 (1981)

[4] Herbort, S.: *Ein komponentenreduzierendes Branch-and-Prune-Verfahren zur verifizierten Lösung nichtlinearer Gleichungssysteme.* Diplomarbeit, Universität Karlsruhe (1996)

[5] Hong, H.; Stahl, V.: *Safe Starting Regions by Fixed Points and Tightening.* Computing **53**, 323–335 (1994)

[6] Kearfott, R.B.: *Some Tests of Generalized Bisection.* ACM Transactions on Mathematical Software **13**, 197–220 (1987)

[7] Knüppel, O.: *Einschließungsmethoden zur Bestimmung der Nullstellen nichtlinearer Gleichungssysteme und ihre Implementierung.* Dissertation, Technische Universität Hamburg-Harburg (1995)

[8] Meintjes, K.; Morgan, A. P.: *Chemical Equilibrium Systems.* ACM Transactions on Mathematical Software **16**, 143–151 (1990)

[9] Moore, R. E.; Jones, S. T.: *Safe Starting Regions for Iterative Methods.* SIAM Journal on Numerical Analysis **14**, 1051–1065 (1977)

[10] Moré, J. J.; Cosnard, M. Y.: *Numerical Solution of Nonlinear Equations.* ACM Transactions on Mathematical Software **5**, 64–85 (1979)

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

[12] Ratz, D.: *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen.* Dissertation, Universität Karlsruhe (1992)

[13] Ratz, D.: *On Extended Interval Arithmetic and Inclusion Isotonicity.* Submitted for publication in SIAM Journal on Numerical Analysis

[14] Van Hentenryck, P.; McAllester, D.; Kapur, D.: *Solving Polynomial Systems Using a Branch and Prune Approach.* Technical Report CS-95-01, Dept. of Comp. Sci., Brown University (1995)

In dieser Reihe sind bisher die folgenden Arbeiten erschienen:

1/1996 Ulrich Kulisch: *Memorandum über Computer, Arithmetik und Numerik.*

2/1996 Andreas Wiethoff: *C–XSC — A C++ Class Library for Extended Scientific Computing.*

3/1996 Walter Krämer: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen.*

4/1996 Dietmar Ratz: *An Optimized Interval Slope Arithmetic and its Application.*

5/1996 Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic.*

1/1997 Astrid Goos, Dietmar Ratz: *Praktische Realisierung und Test eines Verifikationsverfahrens zur Lösung globaler Optimierungsprobleme mit Ungleichungsnebenbedingungen.*

2/1997 Stefan Herbort, Dietmar Ratz: *Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method.*

3/1997 Ulrich Kulisch: *Die fünfte Gleitkommaoperation für top-performance Computer — oder — Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik.*

4/1997 Ulrich Kulisch: *The Fifth Floating-Point Operation for Top-Performance Computers — or — Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic.*

5/1997 Walter Krämer: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen.*