4^{th} GRACM Congress on Computational Mechanics
GRACM 2002
Patra, 27–29 June, 2002

IMPROVING THE EFFICIENCY OF A POLYNOMIAL SYSTEM SOLVER VIA A REORDERING TECHNIQUE

D.G. Sotiropoulos, J.A. Nikas, and T.N. Grapsa

University of Patras, Department of Mathematics,
Division of Computational Mathematics & Informatics,
GR-265 00, Patras, Greece.
e-mail: {dgs,nikas,grapsa}@math.upatras.gr.

Keywords: nonlinear polynomial systems, reordering technique, interval methods, interval Gauss-Seidel step, Hansen's algorithm.

Abstract. *Methods of interval arithmetic can be used to reliably find with certainty all solutions to nonlinear systems of equations. In such methods, the system is transformed into a linear interval system and a preconditioned interval Gauss-Seidel method may then be used to compute such solution bounds. In this work, a new heuristic for solving polynomial systems is presented, called reordering technique. The proposed technique constitutes a preprocessing step to interval Gauss-Seidel method to improve the overall efficiency of an interval Newton method. The key idea is to exploit some properties of the original polynomial system, expressed by two suitable permutation matrices, by reordering the resulted linearized system. Numerical experiments have been shown that the permuted system can be solved efficiently when it is combined with an interval Newton method, like Hansen's algorithm. We present the motivation for the reordering scheme and support these arguments by a sample of several test results.*

1 INTRODUCTION

We consider the problem of finding all solutions of the following nonlinear polynomial system

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0, \end{aligned} \tag{1}$$

contained in a bounded rectangular region (box) $\mathbf{B} \subseteq \mathbb{R}^n$. In vector notations the system (1) will be written as $F(X) = 0$, where $F = (f_1, f_2, \ldots, f_n)^\top$, and $X = (x_1, x_2, \ldots, x_n)^\top$.

For what it follows we need some notations and notions. Let $\mathbb{IR} = \{[a, b] \mid a \leq b,\ a, b \in \mathbb{R}\}$ be the set of real intervals, while the set of n-dimensional interval vectors, i.e. n-dimensional boxes, will be denoted by $\mathbb{IR}^n$. Intervals, interval vectors and interval matrices will be denoted by capital boldface letters. Thus, an interval vector $\mathbf{X} = (\mathbf{X}_i) = (\mathbf{X}_1, \ldots, \mathbf{X}_n)^\top \in \mathbb{IR}^n$ has n real interval components $\mathbf{X}_i \in \mathbb{IR}$. Similarly, an interval matrix $\mathbf{A} = (\mathbf{A}_{ij}) \in \mathbb{IR}^{n\times n}$ has interval elements $\mathbf{A}_{i,j} \in \mathbb{IR}$, $i, j = 1, \ldots, n$. If $\mathbf{X} = [\underline{X}, \overline{X}]$ is a given interval, where $\underline{X}$ and $\overline{X}$ are the lower and upper endpoints, respectively, we define as $mid(\mathbf{X}) = (\underline{X} + \overline{X})/2$ the midpoint and $wid(\mathbf{X}) = \overline{X} - \underline{X}$ the width of X. The midpoint of an interval vector $\mathbf{X} = (\mathbf{X}_i)$ or matrix $\mathbf{A} = (\mathbf{A}_{ij})$ are defined componentwise. We call a function $\mathbf{F} : \mathbb{IR}^n \to \mathbb{IR}$ an inclusion of $f : \mathbb{R}^n \to \mathbb{R}$ if $f_{rg} \subseteq \mathbf{F}(\mathbf{X})$ for any $\mathbf{X} \in \mathbb{IR}^n$ where $f_{rg} = \{f(x) : x \in \mathbf{X}\}$ be the range of f over $\mathbf{X}$. More information on interval arithmetic can be found in many places (e.g., [10], [11],[1],[13],[6])

A successful approach to problem (1) is generalized bisection in conjunction with interval Newton methods. For a more thorough discussion underlying these methods see ([1], [11], [13]). The basic idea in interval Newton methods is to solve the linear interval equation system

$$\mathbf{F}'(\mathbf{X}^{(k)})(\mathbf{N}^{(k)} - X^{(k)}) = -\mathbf{F}(X^{(k)}) \tag{2}$$

D. T. Tsahalis (ed.), Proceedings of 4th GRACM Congress on Computational Mechanics (CRACM2002)
Vol. III, pp. 970-976, 2002.

for $\mathbf{N}^{(k)}$, where k is an iteration counter, $\mathbf{F}'(\mathbf{X}^{(k)})$ is a suitable interval extension of the Jacobian matrix over the current box $\mathbf{X}^{(k)}$ (with $\mathbf{X}^{(0)} = \mathbf{B}$), and $X^{(k)} \in \mathbf{X}^{(k)}$ represent an initial guess point, usually taken to be the midpoint. We then define the next iterate $\mathbf{X}^{(k+1)}$ by

$$\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} \cap \mathbf{N}^{(k)}. \tag{3}$$

This scheme is termed an interval Newton method. For several techniques for finding $\mathbf{N}^{(k)}$ from Eq. (2), it can be proven ([13]) that if $\mathbf{N}^{(k)} \subset \mathbf{X}^{(k)}$, then the system of equations in (1) has a unique solution in $\mathbf{X}^{(k)}$, and furthermore that Newton's method with real arithmetic will converge to that solution from any initial point in $\mathbf{X}^{(k)}$. Conversely, if $\mathbf{X}^{(k)} \cap \mathbf{N}^{(k)} = \emptyset$ then there exist no solutions in $\mathbf{X}^{(k)}$; if the components of $\mathbf{X}^{(k+1)}$ are not smaller than those of $X^{(k)}$, then one could bisect $X^{(k+1)}$ to form two new boxes and continue the iteration (3) with one of the resulting boxes. This is the basic idea of interval Newton/Generalized bisection methods.

The various interval Newton methods differ in how they determine $\mathbf{N}^{(k)}$ from Eq. (2). $\mathbf{N}^{(k)}$ may be computed using interval Gaussian elimination; frequently, however, it is computed componentwise using a preconditioned Gauss–Seidel method, developed by Hansen and Sengupta [4], as described below. Preconditioning Eq. (2) with a scalar nonsingular matrix $Y^{(k)}$ we obtain

$$Y^{(k)}\,\mathbf{F}'(\mathbf{X}^{(k)})\,(\mathbf{N}^{(k)} - X^{(k)}) = -Y^{(k)}\,\mathbf{F}(X^{(k)}), \tag{4}$$

where the matrix $Y^{(k)}$ is usually chosen to be the inverse of the real matrix formed from the midpoints of the components of $\mathbf{F}'(\mathbf{X}^{(k)})$, i.e.,

$$Y^{(k)} = \left[mid\left(\mathbf{F}'(\mathbf{X}^{(k)})\right)\right]^{-1}. \tag{5}$$

Defining $\mathbf{M} = Y^{(k)}\,\mathbf{F}'(\mathbf{X}^{(k)})$ and $\mathbf{b} = Y^{(k)}\,\mathbf{F}(X^{(k)})$, the preconditioned interval Gauss–Seidel algorithm is:

Algorithm 1.1 *(Preconditioned Gauss-Seidel) Do the following for $i = 1$ to n.*

1. *Compute*

$$\boldsymbol{N}_i^{(k)} = x_i^{(k)} - \left[\boldsymbol{b}_i + \sum_{j=1}^{i-1} \boldsymbol{M}_{ij}\,(\boldsymbol{X}_j^{(k+1)} - x_j^{(k+1)}) + \sum_{j=i+1}^{n} \boldsymbol{M}_{ij}\,(\boldsymbol{X}_j^{(k)} - x_j^{(k)})\right] \Big/ \boldsymbol{M}_{ii} \tag{6}$$

2. *(The new box is empty.) If $\boldsymbol{N}_i^{(k)} \cap \boldsymbol{X}_i^{(k)} = \emptyset$, then signal that there is no solution of $F(X)$ in $\boldsymbol{X}^{(k)}$*
3. *(Prepare for the next coordinate.)* $\boldsymbol{X}_i^{(k+1)} = \boldsymbol{N}_i^{(k)} \cap \boldsymbol{X}_i^{(k)}$

If $0 \in \mathbf{M}_{ii}$ for some i in Eq. (6), then extended interval arithmetic [3] must be used. In this case a gap can be produced in the corresponding components of $\mathbf{X}_i^{(k+1)}$, i.e. $\mathbf{X}_i^{(k+1)}$ is given by one or two intervals obtained from the extended interval division and the succeeding intersection with the old $\mathbf{X}_i^{(k)}$ (see, [3], [6]).

Herbort and Ratz in [5] have been suggested a componentwise interval Newton method for finding enclosures of all solutions of a system of nonlinear equations. The proposed operator is applied on a temporarily univariate real-valued function of the system, which has interval coefficients. For each of the n components $\mathbf{X}_j$ of the search box n different functions F_i can be chosen trying to prune the interval $\mathbf{X}_j$. Hence, we have several possibilities to choose such pairs (f_i, x_j).

In this work, motivated from the above ascertainment, we develop heuristics that aim in determining pairs (f_i, x_j), exploiting properties of polynomial equations, for solving polynomial systems using an interval Newton method. According to our knowledge these properties do not seem to have been exploited yet in the area of interval methods. These heuristics are summarized in two algorithms and used to permute the vector of variables and equations. The resulting reordering can be expressed by two permutation matrices, which are applied to the linearized interval system (2). According to our numerical experiments, when a polynomial system solver is enhanced with our preprocessing step is, in many cases, much faster than the application of the solver to the original system.

The remainder of the paper is organized as follows: In Section 2 we briefly present the preprocessing step in algorithmic form, while in Section 3 we report numerical results from a sample of several test experiments. We evaluate the proposed reordering technique by using Hansen's algorithm from [8]. Finally, Section 4 contains conclusions and suggestions for further research.

2 PREPROCESSING STEP: A REORDERING TECHNIQUE

The polynomial equations $f_1, f_2, \ldots, f_n$ are written as sums of terms. Thus,

$$f_j(x_1, x_2, \ldots, x_n) = \mathrm{MON}_1 + \mathrm{MON}_2 + \cdots + \mathrm{MON}_k,$$

where k is the number of terms in f_j. Each term is a product of a constant and variables raised to powers. The general term, called *monomial*, has the following form

$$\mathrm{MON} = a x_1^{m_1} x_2^{m_2} \cdots x_n^{m_n},$$

where a is a constant and $m_1, m_2, \ldots, m_n$ are nonnegative integers. The degree of the monomial is defined as the sum of the degrees of each variable, i.e. $\deg(\mathrm{MON}) = m_1 + \cdots + m_n$,

Definition 2.1 ([12]) *Let $f_j = \sum_{l=1}^{k} \mathrm{MON}_l$ be a polynomial equation, and $\deg(\mathrm{MON}_l)$ be the degree of the l-th monomial. Then, we define as degree of the equation f_i the biggest of the degree of its monomials*

$$d_i = deg(f_i) = \max_l deg(\mathrm{MON}_l) \tag{7}$$

Let $\mathcal{S}_x = \{x_1, \ldots, x_n\}$ be the set of variables x_j, $j = 1, \ldots, n$, and $\mathcal{S}_f = \{f_1, \ldots, f_n\}$ be the set of equations f_i, $i = 1, \ldots, n$. We correspond each variable $x_j \in \mathcal{S}_x$ to a subset $\mathcal{S}_f^j = \{f_i \mid \frac{\partial f_i}{\partial x_j} \neq 0\} \subseteq \mathcal{S}_f$, and each function $f_i \in \mathcal{S}_f$ to a subset $\mathcal{S}_x^i = \{x_j \mid \frac{\partial f_i}{\partial x_j} \neq 0\} \subseteq \mathcal{S}_x$.

As it has been mentioned in Section 1, our aim is the determination of pairs (f_i, x_j) by developing strategies using some polynomial properties. Algorithm 2.1 summarizes these mechanisms.

Algorithm 2.1 Pair_Determination (Input: Sets $\mathcal{S}_x$ and $\mathcal{S}_f$; Output: Set of pairs $\mathcal{T}$)

1. *(Initialization.) Create subsets $\mathcal{S}_x^i$ and $\mathcal{S}_f^j$, $\forall\, i, j \in \{1, \ldots, n\}$, and initialize $\mathcal{T} = \emptyset$.*
2. *While $\mathcal{S}_x^i \neq \emptyset$ and $\mathcal{S}_f^j \neq \emptyset$ do the following steps*
3. *Determine a subset $\mathcal{S}$ such that $\mathcal{S} \subseteq \mathcal{S}_x$ or $\mathcal{S} \subseteq \mathcal{S}_f$ according to the following rules*
   (a) *Select the subset $\mathcal{S}$, which cardinality is $card(\mathcal{S}) = \min_{i,j}\{\, card(\mathcal{S}_f^j),\, card(\mathcal{S}_x^i)\,\}$*
   (b) *If subset $\mathcal{S}$ is not uniquely defined from (a), then*
      i. *If there exists at least one subset $\mathcal{S} \subseteq \mathcal{S}_f$ then we select this one which the corresponding variable x_j has the biggest power in the whole system.*
      ii. *If every subset $\mathcal{S} \subseteq \mathcal{S}_x$ then we select this one which the corresponding equation f_i*
         A. *has the biggest degree d_i.*
         B. *has the smallest width of range over the initial searching box.*
4. *If $\mathcal{S} \subseteq \mathcal{S}_f$ then select the equation f_i*
   (a) *with the biggest degree d_i.*
   (b) *with the smallest width of range over the initial searching box.*
5. *If $\mathcal{S} \subseteq \mathcal{S}_x$ then select the variable x_j with the biggest power in the whole system.*
6. *Insert the pair (f_i, x_j) in $\mathcal{T}$.*
7. *Delete the subsets $\mathcal{S}_f^j$, $\mathcal{S}_x^i$.*
8. *Delete the equation f_i from the sets $\mathcal{S}_f^k$, where $k \in \{1, \ldots, i-1, i+1, \ldots, n\}$ and the variable x_j from the sets $\mathcal{S}_x^l$, where $l \in \{1, \ldots, j-1, j+1, \ldots, n\}$*

Algorithm 2.1 first creates $2n$ subsets and initializes the result set $\mathcal{T}$ which consists of the preferable pairs. The main iteration (from Step 3 to Step 8) starts by selecting either an equation f_i or a variable x_j. Specifically, in Step 3 are defined priority rules by means that we proceed in the next rule if the current rule does not determine uniquely an equation or a variable. Step 4 or 5 specifies the remaining component of the pair, i.e. the function f_i or the variable x_j, respectively, while in Step 6 the pair is stored in $\mathcal{T}$. Step 7 removes the corresponding subsets, while Step 8 reduces the cardinality of the remaining subsets by removing equation f_i and variable x_j from all the subsets in which are contained.

In many cases, an equation f_i cannot be determined uniquely, so, we have enforced the algorithm with another one heuristic criterion which uses the estimated range of an equation. Using this criterion the determination of f_i is more effective since we have to compare real numbers (width of range) instead of integers (degree of an equation). The price to pay for this criterion is at most n evaluations of single components f_i.

Subsequently, Algorithm 2.2 reorders the generated pairs by the Algorithm 2.1 using two criteria for choosing the preferable order of pairs. Reorderings can be conveniently described by two permutation matrices P_1 and P_2, which are the output of the Algorithm 2.2.

Algorithm 2.2 Reordering (Input: Set of pairs $\mathcal{T}$; Output: Permutation matrices P_1 and P_2)

1. *$P_1 = I$, $P_2 = I$*
2. *While $T \neq \emptyset$ do the following steps*
3. *Select a pair (f_i, x_j) according to the following priority rules*
   (a) *The pair with biggest power of variable x_j in the equation f_i*
   (b) *The pair with the smallest width of range of the function f_i*
4. *Permute properly the rows and columns of P_1 and P_2, respectively.*
5. *Delete the pair (f_i, x_j) from $\mathcal{T}$*

Remark 2.1 *An equivalent polynomial system of equations can be immediately obtained from the application of permutation matrices P_1 and P_2 over the vectors of equations and variables, respectively.*

Algorithms 2.1 and 2.2 constitute our proposed preprocessing step and can be applied in any polynomial system solver. The only requirement for non-interval methods is the availability of an interval arithmetic library ([7], [8], [9]). However, in this work, we concentrate in the application of the preprocessing step to interval Newton methods and especially in interval Hansen's multi-dimensional algorithm (see [3] and [8] for details).

The previous described reordering technique can be applied either to original polynomial system, as pointed out in Remark 2.1, or to linearized interval system (2). More precisely, the obtained equivalent linear system takes the form

$$\left(P_1 \cdot \mathbf{F}'(\mathbf{X}^{(k)}) \cdot P_2\right) \cdot \left(P_2^{\top} \cdot (\mathbf{N}^{(k)} - X^{(k)})\right) = -P_1 \cdot \mathbf{F}(X^{(k)}) \tag{8}$$

where P_1 and P_2 are the permutation matrices which describe the reorderings, as resulted from Algorithm 2.2. Similarly to Eq. (5), the new preconditioned matrix Y' will become

$$Y'^{(k)} = \left[mid\left(P_1 \cdot \mathbf{F}'(\mathbf{X}^{(k)}) \cdot P_2\right)\right]^{-1}. \tag{9}$$

Obviously, the application of the preprocessing step as formed in Eq. (8) is significant only from theoretical point of view. Thus, in practice we have applied the reordering technique directly to the original polynomial system.

3 EXPERIMENTAL RESULTS

In this section, we report a sample of several test experiments on the reordering technique mentioned in section 2. The computations were carried out on a processor Pentium-III computer (866 MHz, 512Mbyte). The implementation of the preprocessing step (Algorithms 2.1 and 2.2) has been carried out in C++ using the C–XSC library [8].

Our preprocessing step has been applied in conjunction with the module `nlss` from [8] which implements the Hansen's multi-dimensional algorithm. For each test-problem we report the original and the permuted system. The results are summarized in tables where we use the following abbreviations:

Method	Used solving method. `nlss` marks the Hansen's algorithm as implemented in [8], while `nlss (+)` incorporates the preprocessing step.
Time	Total CPU time
Encls	Number of root enclosures found.
FcEv	Number of evaluations of a single component of the function.
JcEv	Number of evaluations of a single component of the Jacobi matrix.
GS	Number of calls of the interval Gauss–Seidel step.
Bisect	Number of bisections executed by the algorithm.

Example 3.1 "Brown's almost linear function" is a system that can be considered for arbitrary dimensions. For $n = 3$, we have

$$\begin{aligned} 2x_1 + x_2 + x_3 - 4 &= 0 \\ x_1 + 2x_2 + x_3 - 4 &= 0 \\ x_1x_2x_3 - 1 &= 0 \end{aligned} \quad \overset{P_1,P_2}{\Longrightarrow} \quad \begin{aligned} 2x_3 + x_1 + x_2 - 4 &= 0 \\ x_3 + 2x_1 + x_2 - 4 &= 0 \\ x_3x_1x_2 - 1 &= 0 \end{aligned}$$

Starting Box: $[-10, 10]^3$, tolerance: $\varepsilon = 10^{-6}$

Method	`nlss`	`nlss (+)`	Improvement %
Time	0.61	0.44	27.88
Encls	7	3	-
FcEv	1899	1368	27.96
JcEv	3861	2736	29.14
GS	197	149	24.37
Bisect	209	150	28.23

Module `nlss (+)` seems to accelerate the Hansen's algorithm, in regard of the total CPU time. Additionally, a significant reduction of function and Jacobian evaluations is observed as well as a reduction in the expensive Gauss-Seidel steps.

Example 3.2 Powell's singular function:

$$\begin{aligned} x_1 + 10x_2 &= 0 \\ \sqrt{5}(x_3 - x_4) &= 0 \\ (x_2 - 2x_3)^2 &= 0 \\ \sqrt{10}(x_1 - x_4)^2 &= 0 \end{aligned} \quad \overset{P_1,P_2}{\Longrightarrow} \quad \begin{aligned} (x_4 - 2x_1)^2 &= 0 \\ \sqrt{10}(x_2 - x_3)^2 &= 0 \\ \sqrt{5}(x_1 - x_3) &= 0 \\ x_1 + 10x_4 &= 0 \end{aligned}$$

The above system has singular Jacobian matrix and has a unique zero at $x^* = (0, 0, 0, 0)^\top$.

Starting Box: $[-1, 1]^4$, tolerance: $\varepsilon = 10^{-6}$

Method	`nlss`	`nlss (+)`	Improvement %
Time	4.34	2.97	31.57
Encls	13	7	-
FcEv	15104	10052	33.45
JcEv	48064	32256	32.89
GS	759	490	35.44
Bisect	1495	1004	32.84

Reliant on the above results we can deduce the good performance of the enhanced module `nlss (+)` on sparse systems. It is noticeable that the number of root enclosures which have been found by the module `nlss` is almost twice compared to `nlss (+)`. We can also observe 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. However, contrary to module `nlss`, module `nlss (+)` seems to stay impressively unaffected from the choice of the starting box, as it is shown in the following table.

Starting Box: $[-0.5, 1]^4$, tolerance: $\varepsilon = 10^{-6}$

Method	`nlss`	`nlss (+)`	Improvement %
Time	962.68	1.31	99.86
Encls	1	1	-
FcEv	3349468	4668	99.86
JcEv	11133952	15008	99.87
GS	141494	228	99.84
Bisect	347935	468	99.87

Example 3.3 The following system is a representative of a set of examples given by Moore and Jones [5].

$$\begin{array}{rcl}
x_1 - 0.25428722 - 0.18324757x_4x_3x_9 = 0 & & x_1 - 0.19807914 - 0.15585316x_8x_4x_7 = 0 \\
x_2 - 0.37842197 - 0.16275449x_1x_{10}x_6 = 0 & & x_2 - 0.37842197 - 0.16275449x_4x_{10}x_7 = 0 \\
x_3 - 0.27162577 - 0.16955071x_1x_2x_{10} = 0 & & x_3 - 0.27162577 - 0.16955071x_4x_2x_{10} = 0 \\
x_4 - 0.19807914 - 0.15585316x_7x_1x_6 = 0 & & x_9 - 0.07056438 - 0.17081208x_4x_8x_7 = 0 \\
x_5 - 0.44166728 - 0.19950920x_7x_6x_3 = 0 & \overset{P_1,P_2}{\Longrightarrow} & x_4 - 0.25428722 - 0.18324757x_1x_3x_5 = 0 \\
x_6 - 0.14654113 - 0.18922793x_8x_5x_{10} = 0 & & x_7 - 0.14654113 - 0.18922793x_9x_6x_{10} = 0 \\
x_7 - 0.42937161 - 0.21180486x_2x_5x_8 = 0 & & x_5 - 0.34504906 - 0.19612740x_{10}x_7x_9 = 0 \\
x_8 - 0.07056438 - 0.17081208x_1x_7x_6 = 0 & & x_6 - 0.44166728 - 0.19950920x_8x_7x_3 = 0 \\
x_9 - 0.34504906 - 0.19612740x_{10}x_6x_8 = 0 & & x_8 - 0.42937161 - 0.21180486x_2x_6x_9 = 0 \\
x_{10} - 0.42651102 - 0.21466544x_4x_8x_1 = 0 & & x_{10} - 0.42651102 - 0.21466544x_1x_9x_4 = 0
\end{array}$$

An analogous reduction is occurred in the above 10x10 polynomial system, using as starting searching box the interval $[0, 2]^{10}$.

Starting Box: $[0, 2]^{10}$, tolerance: $\varepsilon = 10^{-6}$

Method	`nlss`	`nlss (+)`	Improvement %
Time	0.99	0.77	22.22
Encls	1	1	-
FcEv	1450	1240	14.48
JcEv	12500	10900	12.8
GS	19	14	26.32
Bisect	61	53	13.11

Example 3.4 A non-singular problem [2].

$$\begin{array}{rcl}
x_3^3 - x_3x_1x_2 = 0 & & x_1^3 - x_1x_2x_3 = 0 \\
x_1^2 - x_3x_2 = 0 & \overset{P_1,P_2}{\Longrightarrow} & x_2^2 - x_1x_3 = 0 \\
10x_3x_2 + x_1 - x_3 - 0.1 = 0 & & 10x_1x_3 + x_2 - x_1 - 0.1 = 0
\end{array}$$

The most impressive example which demonstrate the effectiveness of module `nlss (+)` is the above non-singular problem. For each reported parameter we have an improvement more than 70%.

Starting Box: $[-10, 10]^3$, tolerance: $\varepsilon = 10^{-8}$

Method	`nlss`	`nlss (+)`	Improvement %
Time	155.39	41.47	73.31
Encls	2	2	-
FcEv	437058	113799	73.96
JcEv	913356	225765	75.28
GS	44200	12846	70.92
Bisect	50739	12540	75.29

4 CONCLUSIONS–FURTHER WORK

The proposed heuristics do not constitute a new method but seems to optimize the performance of the using interval method. At the moment, we have considered the proposed technique as a preprocessing step, however, our major goal is to incorporate this reordering technique in the componentwise interval Newton method proposed in [5] by Herbort and Ratz.

References

[1] Götz Alefeld and Jürgen Herzberger. *Introduction to Interval Computations.* Academic Press, London, 1983.

[2] T.N. Grapsa and M.N. Vrahatis. A dimension-reducing method for solving systems of nonlinear equations in $\mathbb{R}^n$. *Intern. J. Computer Math.*, 32:205–216, 1990.

[3] Eldon R. Hansen. *Global Optimization using Interval Analysis.* Marcel Dekker, inc., New York, 1992.

[4] E.R. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT*, 21:203–211, 1981.

[5] S. Herbort and D. Ratz. Improving the efficiency of a nonlinear–system–solver using a componentwise newton method. Technical Report Bericht 2/1997, Institut für Angewandte Mathematik, Universität Karlsruhe (TH), 1997.

[6] R.Baker Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Netherlands, 1996.

[7] R.B. Kearfott, M. Dawande, K.-S. Du, and C.-Y. Hu. Algorithm 737: INTLIB: a portable FORTRAN 77 interval standard function library. *ACM Trans. Math. Software*, 20:447–459, 1994.

[8] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, and A. Wiethoff. *C-XSC – A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, Heidelberg, 1993.

[9] O. Knüppel. PROFIL/BIAS—a fast interval library. *Computing*, 53:277–287, 1994.

[10] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1966.

[11] Ramon E. Moore. *Methods and Applications of Interval Analysis.* Siam, Philadelphia, 1979.

[12] Alexander Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[13] Arnold Neumaier. *Interval Methods for systems of equations.* Cambridge University Press, 1990.