Interval Methods for Solving Underdetermined Nonlinear Equations Systems*

Bartłomiej Jacek Kubica
Institute of Control and Computation Engineering,
Warsaw University of Technology, Nowowiejska 15/19,
00-665 Warsaw, Poland
bkubica@elka.pw.edu.pl

Abstract

The problem of enclosing all solutions of an underdetermined system of equations is considered. A few variants of the algorithm to solve this problem are compared – some of the features come from the literature and some are original. The paper discusses both implementational and theoretical issues of the problem, including a useful theorem that is proved. Shared-memory parallelization, using OpenMP is also considered and numerical results for proper test problems are presented.

Keywords: underdetermined systems of equations, interval Newton operators, componentwise Newton operator, constraint satisfaction, parallel programming
AMS subject classifications: 65G20, 65Y05, 62J10, 03E72

1 Introduction and previous work

The following problem is considered: enclose all solutions of the equations system $f(x) = 0$, where $x \in [\underline{x}, \overline{x}]$ and $f\colon \mathbb{R}^n \to \mathbb{R}^m$, $m < n$. This is one of the problems that can be quite conveniently solved by interval methods while it is difficult to apply other approaches to it.

The most notable paper on this topic is due to Neumaier [12] in the late eighties. Not many researchers continued the investigations presented there. Notable exceptions include [11], where the vectorization of Neumaier's algorithm was considered and [5], where methods similar to [12] were used to solve a very specific problem, arising in homotopy methods. Kolev (e. g. [8]) mentions underdetermined problems, but uses quite different methods, applying some affine arithmetic notions.

Also some books (e.g. [2], [6], [13]) mention problems with non-square Jacobi matrix, but do not consider this topic in details.

*Submitted: January 10, 2009; Revised: February 10, 2010; Accepted: March 1, 2010.

2 Basics

The main meta-algorithm used to solve systems of nonlinear equations is the interval branch-and-prune method. In particular, this approach can be used for underdetermined systems of equations.
What has to be considered carefully, are the rejection/reduction tests, as they are supposed to seek segments of a continuous manifold now, not isolated points. Several variants of the interval Newton operator can be applied here, but – as the Jacobi matrix is non-square for underdetermined systems – some modifications with respect to well-determined variants might be necessary.
The main algorithm has the following form:

```
IBP (x^(0); f)
// x^(0) is the initial box, f(·) is the interval extension of the function f: R^n → R^m
// L_ver is the list of boxes verified to contain a segment of the solution manifold
// L_pos is the list of boxes that possibly contain a segment of the solution manifold
L = L_ver = L_pos = ∅ ;
x = x^(0) ;
loop
    process the box x, using the rejection/reduction tests ;
    if (x does not contain solutions) then discard x ;
    else if (x is verified to contain a segment of solution manifold) then push (L_ver, x) ;
    else if (tests subdivided x into x^(1) and x^(2)) then
        x = x^(1) ;
        push (L, x^(2)) ;
        cycle loop ;
    else if (x is small enough) then push (L_pos, x) ;
    if (x was discarded or stored) then
        x = pop (L) ;
        if (L was empty) then exit loop ;
    else
        bisect (x), obtaining x^(1) and x^(2);
        x = x^(1) ;
        push (L, x^(2)) ;
    end if ;
end loop
```

How can it be verified that a segment of the solution manifold is contained in a box? This will be discussed in the later part of the paper.
Now, let us focus on the rejection/reduction tests, i. e. interval Newton operators.

Interval Newton operators

In this paper three main forms of the Newton operator are applied: Hansen's variant, Neumaier's variant and the componentwise Newton operator.

Hansen's variant. This approach was inspired by Hansen's technique of finding feasible points for equality-constrained optimization problems (see e. g. [6]). As the Jacobi matrix is not square and has no diagonal, we cannot apply the Gauss-Seidel operator directly. So we create a square submatrix, selecting variables, by the Gauss elimination with full pivoting, applied to midpoint of the Jacobi matrix.

Neumaier's variant. The linear equations system $Av = b$ (being the linearization of the nonlinear system; $v = x - \operatorname{mid} \boldsymbol{x}$) is transformed to the homogeneous form $\widetilde{A}d = 0$, where $d = (v_1, \ldots, v_n, 1)^T$ and $\widetilde{A} = (A \mid -b)$. On such a system we perform a process similar to Hansen's variant, but we do not create a submatrix – only the preconditioner and list of pairs (equation number, variable number). Then an extended Gauss-Seidel step is performed on the preconditioned system with the rectangular matrix.

Componentwise Newton operator. This technique [4] can be used to any system of equations (under-, over- or well-determined) without any changes to the operator itself:

$$N_{cmp}(\boldsymbol{x}, \mathsf{f}, i, j) = \operatorname{mid} \boldsymbol{x}_j - \frac{\mathsf{f}_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, \operatorname{mid} \boldsymbol{x}_j, \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n)}{\frac{\partial \mathsf{f}_i}{\partial x_j}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)} \ .$$

The only issue is to choose pairs (i, j) properly. A few techniques are possible there – e. g. the Herbort and Ratz heuristic [4], that selects all pairs, for which the corresponding element of the Jacobi matrix is not zero or the Goualard heuristic [3] that tends to find a transversal, computing the maximum perfect matching of a biparite graph, described by the matrix $W = (W_{ij})$, where:

$$W_{ij} = \begin{cases} |\boldsymbol{J}_{ij}| & \text{if } 0 \in \boldsymbol{J}_{ij} \\ \langle \boldsymbol{J}_{ij} \rangle + \max |\boldsymbol{J}_{ij}| & \text{otherwise} \end{cases} , \tag{1}$$

where – following conventions from [7] – $\langle \cdot \rangle$ denotes the mignitude and $|\cdot|$ – magnitude of an interval, i. e. the minimal and maximal value of the absolute value of its elements.

3 Main theorem

Theorem 3.1 *Consider a box $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$. Consider a set J of m variables. Suppose a Newton operator was computed for each x_j, $j \in J$ and resulted in $N_{cmp}(\boldsymbol{x}, \mathsf{f}, i, j) \subset \boldsymbol{x}_j$.*
Let us denote $J = \{j_1, \ldots, j_m\}$ and the set of variables not in J – $\{k_1, \ldots, k_{n-m}\}$. Then:

$$\forall x_{k_1} \in \boldsymbol{x}_{k_1} \ldots \forall x_{k_{n-m}} \in \boldsymbol{x}_{k_{n-m}} \ \exists! x_{j_1} \in \boldsymbol{x}_{j_1} \ldots \exists! x_{j_m} \in \boldsymbol{x}_{j_m} \quad f(x_1, \ldots, x_n) = 0 \ .$$

The proof can be constructed in a way similar to other Newton operators.
Proof.
Consider the function $\varphi(\cdot) = f_i(\xi_1, \ldots, \xi_{j-1}, \cdot, \xi_{j+1}, \ldots, \xi_n)$, defined on the interval $\boldsymbol{x}_j$. Derivative of this function is $\varphi'(x_j) = \frac{\partial f_i(\xi_1, \ldots, \xi_{j-1}, x_j, \xi_{j+1}, \ldots, \xi_n)}{\partial x_j}$.
Function $\varphi(x_j)$ can be linearized, using the first-order Taylor form: $\varphi(c) + a \cdot (x_j - c)$, where $c = \operatorname{mid} x_j$ and $a \in \boldsymbol{a} = \frac{\partial f_i(\xi_1, \ldots, \xi_{j-1}, \boldsymbol{x}_j, \xi_{j+1}, \ldots, \xi_n)}{\partial x_j}$. This way we can bound $\varphi(\cdot)$ by a centered form.
Now, assume that the interval $c - \frac{\varphi(c)}{\boldsymbol{a}}$ is contained in $\boldsymbol{x}_j$. This implies that both extremal functions of the linearization cross the OX axis in the interval $\boldsymbol{x}_j$ and consequently $\varphi(\cdot)$ must have a zero in $\boldsymbol{x}_j$.
Moreover, there is only one such zero, as if there were two of them, there would have been a $\zeta \in \boldsymbol{x}_j$ such that $\varphi'(\zeta) = 0$. Which is impossible as both extremal function cross OX and $\boldsymbol{a}$ cannot contain 0.

Now, please note that:

$$c - \frac{\varphi(c)}{\boldsymbol{a}} = \operatorname{mid} x_j - \frac{f_i(\xi_1, \ldots, \xi_{j-1}, \operatorname{mid} \boldsymbol{x}_j, \xi_{j+1}, \ldots, \xi_n)}{\frac{\partial \mathsf{f}_i(\xi_1, \ldots, \xi_{j-1}, \boldsymbol{x}_j, \xi_{j+1}, \ldots, \xi_n)}{\partial x_j}} \subseteq$$

$$\operatorname{mid} x_j - \frac{\mathsf{f}_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}, \operatorname{mid} \boldsymbol{x}_j, \boldsymbol{x}_{j+1}, \ldots, \boldsymbol{x}_n)}{\frac{\partial \mathsf{f}_i(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)}{\partial x_j}} = N_{cmp}(\boldsymbol{x}, i, j, \mathsf{f})$$

Moreover, the above relation holds for all possible values of ξ's, so when

$$N_{\mathrm{cmp}}(\boldsymbol{x}, i, j, \mathsf{f}) \subset \operatorname{int} \boldsymbol{x}_j,$$

this implies that:

$$\forall x_1 \in \boldsymbol{x}_1 \ldots \forall x_{j-1} \in \boldsymbol{x}_{j-1} \forall x_{j+1} \in \boldsymbol{x}_{j+1} \ldots \forall x_n \in \boldsymbol{x}_n \exists! x_j \in \boldsymbol{x}_j \, f_i(x_1, \ldots, x_n) = 0 \ .$$

Now, suppose the above was verified for a few variables – let their indices be j_1, j_2, …, j_l. Then, obviously, as we can substitute j_t (for $t = 1, \ldots, l$) into the above formulae to see that:

$$\forall x_{k_1} \in \boldsymbol{x}_{k_1} \ldots \forall x_{k_{n-l}} \in \boldsymbol{x}_{k_{n-l}} \ \exists! x_{j_1} \in \boldsymbol{x}_{j_1} \ldots \exists! x_{j_l} \in \boldsymbol{x}_{j_l} \quad f(x_1, \ldots, x_n) = 0 \ .$$

In particular, it holds for $l = m$. □

Comment. For $l > m$ the above formula cannot hold as we have only m equations to use for verification. And if we manage to verify it for m variables, we can be sure that a segment of the solution manifold is contained in the investigated box – and for all values of variables not belonging to the set J there is a corresponding solution point.
Theorem 3.1 has an important use in presented algorithms for solving underdetermined problems. After verifying that a box is guaranteed to contain a segment of the solution manifold, we store this box without considering it further, though the box might be relatively large. This idea – original to the author's best knowledge – allows to reduce the computational amount significantly.

4 Numerical experiments

Numerical experiments were performed on a computer with 16 cores, i. e. 8 Dual-Core AMD Opterons 8218 with 2.6GHz. The machine ran under control of a Fedora 10 Linux operating system.
The solver was implemented in C++, C-XSC 2.2.3 library [15] was used for interval computations and perfect weighted matchings were computed using a free implementation of the Hungarian algorithm [17]. The GCC 4.3.2 compiler was used.
The following test problems were considered.
The first one is a curve on a plane; it is the sum of two concentric circles:

$$\begin{aligned} &(x_1^2 + x_2^2 - 4) \cdot (x_1^2 + x_2^2 - 1) = 0 \ , \\ &x_1, x_2 \in [-3, 5] \ . \end{aligned} \tag{2}$$

The second example is borrowed from [11] and [12], where it is called the hippopede problem:

$$\begin{aligned} &x_1^2 + x_2^2 - x_3 = 0 \ , \quad x_2^2 + x_3^2 - 1.1 x_3 = 0 \ . \\ &x_1 \in [-1.5, 1.5], \ x_2 \in [-1, 1], \ x_3 \in [0, 4] \ . \end{aligned} \tag{3}$$

The following problem, called Puma (see e. g. [11], [12]) arose in the inverse kinematics of a 3R robot and is one of typical benchmarks for nonlinear system solvers:

$$\begin{aligned}
&x_1^2 + x_2^2 - 1 = 0 \ , \quad x_3^2 + x_4^2 - 1 = 0 \ , \\
&x_5^2 + x_6^2 - 1 = 0 \ , \quad x_7^2 + x_8^2 - 1 = 0 \ , \\
&0.004731 x_1 x_3 - 0.3578 x_2 x_3 - 0.1238 x_1 - 0.001637 x_2 - 0.9338 x_4 + x_7 = 0 \ , \\
&0.2238 x_1 x_3 + 0.7623 x_2 x_3 + 0.2638 x_1 - 0.07745 x_2 - 0.6734 x_4 - 0.6022 = 0 \ , \\
&x_6 x_8 + 0.3578 x_1 + 0.004731 x_2 = 0 \ , \\
&-0.7623 x_1 + 0.2238 x_2 + 0.3461 = 0 \ , \\
&x_1, \ldots, x_8 \in [-1, 1] \ .
\end{aligned} \tag{4}$$

In the above form it is a well-determined (8 equations and 8 variables) problem with 16 solutions that are easily found by several solvers. To make it underdetermined the last one or two equations was dropped. Both variants (with 7 and 6 equations) were considered in numerical experiments.
The last problem arose in aircraft equilibrium problems (see [12], [14]):

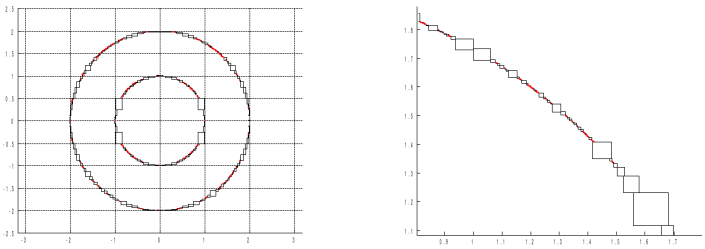
$$\begin{aligned}
&-3.933 x_1 + 0.107 x_2 + 0.126 x_3 - 9.99 x_5 - 45.83 x_7 - 7.64 x_8 + \\
&-0.727 x_2 x_3 + 8.39 x_3 x_4 - 684.4 x_4 x_5 + 63.5 x_4 x_7 = 0 \ , \\
&-0.987 x_2 - 22.95 x_4 - 28.37 x_6 + 0.949 x_1 x_3 + 0.173 x_1 x_5 = 0 \ , \\
&0.002 x_1 - 0.235 x_3 + 5.67 x_5 + 0.921 x_7 - 6.51 x_8 - 0.716 x_1 x_2 + \\
&-1.578 x_1 x_4 + 1.132 x_4 x_7 = 0 \ , \\
&x_1 - x_4 - 0.168 x_6 - x_1 x_2 = 0 \ , \\
&-x_3 - 0.196 x_5 - 0.0071 x_7 + x_1 x_4 = 0 \ .
\end{aligned} \tag{5}$$

This problem has 5 equations in 8 variables, but [12] considers a variant with 6 variables only: we set $x_6 = 0.1$ and $x_8 = 0$. Both variants - with 6 and 8 variables were considered in our experiments. As neither in [12], nor in [14] any bound were given, we set $x_i \in [-2, 2]$.
The following methods are compared in the tables:

- Hansen – Hansen's variant,
- Neum – Neumaier's variant,
- c+HR – the componentwise Newton operator with pairs chosen according to the Herbort and Ratz [4] heuristic (static lists are created on the beginning of the program).
- c+Gou – componentwise Newton operator with the Goualard technique [3] for pairs choosing; lists are recomputed for each box.
- c+GE – the componentwise Newton operator; pairs chosen by Gaussian elimination on midpoint of the Jacobi matrix; lists recomputed for each box.
- c+GouGE – as c+GE, but the Goualard matrix (1) is used instead of the Jacobi matrix.

In early experiments also the author considered also the variants of N_{cmp} method with static lists created by Gou, GE and GouGE heuristics. The results were very poor and these experiments are not presented.
Compared quantities are:

Figure 1: Boxes, covering the solution manifold, computed for problem (2) by Hansen's variant of the algorithm (and zoom of a part the set)



- fun.evals, grad.evals – numbers of functions evaluations and its gradients evaluations,
- pos.boxes, verif.boxes – number of elements in the computed lists of boxes that possibly contain a segment of the solution manifold and that are verified to contain it,
- Leb.pos., Leb.verif. – Lebesgue measures (total hypervolumes) of boxes from both lists,
- time – computation time in seconds.

On Figure 2 we have the solution computed by one of the algorithms (the others performed similarly) for Problem (2).

Table 1: Results for Problem (2), $\varepsilon = 10^{-5}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	140979	6844	801803	142912	142913	142912
grad.evals	141922	8066	401033	143496	143496	143496
pos.boxes	65605	160	191401	66014	66014	66014
verif.boxes	3390	1200	7080	3614	3614	3614
Leb.pos.	2e-6	3e-9	7e-6	2e-6	2e-6	2e-6
Leb.verif.	0.58	0.60	0.51	0.55	0.55	0.55
time (sec.)	2.20	0.13	5.24	1.86	1.72	1.81

5 Computational results

Numerical experiments have shown that in most cases the best method was either the Neumaier's algorithm or the Herbort and Ratz technique.
Methods, recomputing the lists of pairs for the componentwise Newton step performed worse, usually. It is worth noting that all three variants (Gou, GE, GouGE) performed

Table 2: Results for Problem (3), $\varepsilon = 10^{-5}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	11820182	186174	30065516	31081749	27462889	26863397
grad.evals	17198176	211968	13051004	39050540	33532252	60759260
pos.boxes	1378660	20864	1549332	4198728	3647752	5626484
verif.boxes	3400	4120	588	788	564	788
Leb.pos.	4e-10	4e-12	3e-10	7e-10	7e-10	1e-9
Leb.verif.	0.001991	0.002829	0.004804	0.004803	0.001244	0.004803
time (sec.)	170	3	127	328	253	433

Table 3: Results for Problem (4) without the last equation, $\varepsilon = 10^{-4}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	39040127	3776850	154679461	110980461	123275719	117086109
grad.evals	47802356	4302186	60371164	145022192	177525642	183190672
pos.boxes	1245564	124968	2197904	3763096	4315908	4351128
verif.boxes	12964	20056	440	484	360	376
Leb.pos.	2e-29	2e-32	4e-29	4e-29	2e-29	5e-29
Leb.verif.	6e-12	3e-11	5e-15	3e-20	3e-20	8e-18
time (sec.)	1563	159	1116	2777	2779	3044

Table 4: Results for Problem (4) without two last equations, $\varepsilon = 10^{-1}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	1682802	1263426	3099287	2246536	2568064	2432272
grad.evals	1882224	1451424	1173888	2502720	3057324	3037824
pos.boxes	90776	61568	70968	117144	136736	134496
verif.boxes	432	1600	0	16	72	88
Leb.pos.	6e-7	2e-7	6e-7	9e-7	7e-7	8e-7
Leb.verif.	2e-9	5e-8	0	2e-11	5e-10	6e-10
time (sec.)	57	49	23	50	49	54

Table 5: Results for Problem (5) with 6 variables, $\varepsilon = 10^{-3}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	4183290	819720	1157504	6645338	3469047	3216334
grad.evals	5795160	1024850	222015	10923640	5487600	11548340
verif.boxes	0	0	149	0	0	0
Leb.pos.	2e-14	3e-15	2e-17	5e-14	2e-14	4e-14
Leb.verif.	0	0	3e-13	0	0	0
time (sec.)	223	46	9	327	149	310

similarly. Moreover, GE was significantly better than the two other ones in Tables 5 and 6. So, probably, computing the maximum perfect matching might be replaced by simple Gaussian elimination – known much better for researchers dealing with numerical computations, simpler and probably less expensive.

Table 6: Results for Problem (5), $\varepsilon = 10^{-1}$.

method	Hansen	Neum	c+HR	c+Gou	c+GE	c+GouGE
fun.evals	83926845	32204420	139333524	82585776	59874378	117011038
grad.evals	101122600	36138220	25886790	92489260	64409105	250960280
pos.boxes	5348354	1955575	1948669	4513924	3917502	9839533
verif.boxes	26228	56168	21768	0	21553	0
Leb.pos.	0.000276	0.018141	0.000159	0.000394	0.000201	0.001231
Leb.verif.	6e-5	0.005801	0.001677	0	0.001572	0
time (sec.)	4185	1705	1087	2954	1906	7109

Table 7: Parallelization of algorithms for Problem (4) with 7 equations.

method \ threads num.		1	2	4	6	8	10	12
Hansen	time (sec.)	1563	810	405	286	209	176	140
	speedup	1	1.93	3.86	5.47	7.48	8.88	11.16
Neum	time (sec.)	159	82	41	28	21	17	14
	speedup	1	1.94	3.88	5.68	7.57	9.35	11.36
c+HR	time (sec.)	1116	589	304	201	152	124	105
	speedup	1	1.89	3.67	5.55	7.34	9.0	10.63
c+Gou	time (sec.)	2777	1453	725	533	373	307	254
	speedup	1	1.91	3.83	5.21	7.45	9.05	10.93

Table 8: Parallelization of algorithms for Problem (5) with 8 variables.

method \ threads num.		1	2	4	6	8	10	12
Hansen	time (sec.)	4185	2201	1113	758	579	471	400
	speedup	1	1.90	3.76	5.52	7.23	8.89	10.46
Neum	time (sec.)	1705	907	456	309	244	194	162
	speedup	1	1.88	3.74	5.52	6.99	8.79	10.52
c+HR	time (sec.)	1087	567	295	195	150	122	104
	speedup	1	1.92	3.68	5.57	7.25	8.91	10.45
c+Gou	time (sec.)	2954	1559	780	526	400	326	276
	speedup	1	1.89	3.79	5.62	7.38	9.06	10.70

Parallelization of algorithms

OpenMP [16] was used for the shared-memory parallelization. The general idea was similar to the one in [1]: main loop of the IBP algorithm was executed concurrently in a few threads, push/pop operations were guarded by locks (each of the lists had a different lock, obviously) and there was a variable, representing the number of working threads, used to finish the computations (as OpenMP gives us no condition variables, an active wait had to be used – contrary to [9], where we used POSIX threads).

As already investigated in [9] the C-XSC library – at least in version 2.2.3 – is not well suited for multi-threaded computations. In particular, computing the midpoint of an interval had to be replaced by an arithmetic operation; also matrix-matrix and matrix-vector multiplications had to be implemented manually as original functions are not reentrant.

Parallelization increased the efficiency of the algorithms in all investigated cases.

The observed speedup was worse than the one observed in similar experiments of [1]

(they investigated well-determined systems of equations, but that should make no difference). It seems a few reasons may cause worse performance in our experiments – like different compiler or different computer architecture. It is worth noting that early experiments were performed on earlier version on GCC (4.1.2) and the parallel version performed yet far worse.
Nevertheless, speedups reported in [1] seem rather optimistic. Please note that different threads have to synchronize several times:

- taking a box from the list,
- putting a box to one of the lists (only threads manipulating on the same list at a time),
- allocating memory while bisecting a box – the memory is a resource of a process, not a thread; the synchronization is done under the covers and is probably highly dependent on OS, version of the kernel and glibc (GNU C library), etc.

Moreover, the implementation considered in [1] was putting both boxes to the queue after the bisection and taking one from the queue – our algorithm puts one of the boxes to the queue and processes the other one, which should increase the concurrency.
All of the algorithms parallelized relatively well, but the speedup was not linear. No significant difference in the speedup between different variants of the algorithm was observed.

6 Conclusions

We investigated several variants of an IBP method for solving underdetermined systems of nonlinear equations. Considered variants of the Newton operator were either taken from the literature directly or modified only slightly.
However the use of them was different than traditional (e. g. [10], [11]). Theorem 3.1, presented and proved in the paper, allowed us to enclose some parts of the solution manifold by boxes verified to contain its segment. Consequently, the resulting list of such "verified" boxes will contain some boxes of diameter larger than the prescribed accuracy ε; the author decided to store such boxes as solutions, but it would depend on the application, if such approach is acceptable and useful.
As we can infer from Figure 2, the solution manifold is covered quite precisely.
In most cases the Neumaier's method was the best, but the Herbort and Ratz technique outperformed it sometimes.
These results show some interesting phenomenons, analysis of which is beyond the scope of this paper. Great performance of the Neumaier's method is particularly interesting. As investigated in a related paper [10], it outperforms the Hansen's method not because of the uniform structure of the linear system, but rather thanks to a different linearization – using the rectangular Jacobi matrix, not its square submatrix. It is worth noting that linearization used in the Hansen's technique is often more precise, which makes the superiority of Neumaier's method even more surprising. Some possible reasons of this unexpected behavior are analyzed in [10], but the topic requires further studies.
Other minor result was the suggestion to change the Goualard method, by substituting computation of the maximal perfect matching with the well-known Gauss elimination.
The parallelization of all algorithms increased the efficiency of computations, but the speedup was slightly smaller than the expected one. Further research is going to

identify the bottlenecks and improve the parallelization, possibly using other tolls for multi-threaded computations, e. g. TBB.

Acknowledgments

The research has been supported by the Polish Ministry of Science and Higher Education under grant N N514 416934. Thanks to Adam Woźniak for helping me to prepare the figures. The computer on which experiments were performed is shared with the Institute of Computer Science of our University. Thanks to Jacek Błaszczyk for maintaining it. The author is also very grateful to Sergey P. Shary for interesting remarks during the SCAN 2008 conference and for pointing out paper [12] and to Tibor Csendes and Ali Baharev for other fruitful discussions.

References

[1] Beelitz, T., Lang, B., Bischof, C. H.; *Efficient task scheduling in the parallel result-verifying solution of nonlinear systems*, Reliable Computing 12, pp. 141–151, 2006.

[2] Gavriliu, M.; *Towards more efficient interval analysis: corner forms and a remainder interval method*, PhD thesis, California Institute of Technology, Pasadena, California, 2005.

[3] Goualard, F., Jermann, C.; *On the selection of a transversal to solve nonlinear systems with interval arithmetic*, LNCS 3991, pp. 332–339, 2006.

[4] Herbort, S., Ratz, D.; *Improving the efficiency of a nonlinear-system-solver using the componentwise Newton method*, (1997), available on the web at `http://www.uni-karlsruhe.de/~iam/html/reports/rep9702.ps.gz` .

[5] Kearfott, R. B., Xing, Z.; *An interval step control for continuation methods*, SIAM Journal of Numerical Analysis 31, pp. 892–914, 1994.

[6] Kearfott, R. B.; *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, (1996).

[7] Kearfott, R. B., Nakao, M. T., Neumaier, A., Rump, S. M., Shary, S. P., van Hentenryck, P.; *Standardized notation in interval analysis*, available on the web at `http://www.mat.univie.ac.at/~neum/software/ int/notation.ps.gz` .

[8] Kolev, L. V.; *An improved interval linearization for solving nonlinear problems*, Reliable Computing 37, pp. 213–224, 2004.

[9] Kubica, B. J., Woźniak, A.; *A multi-threaded interval algorithm for the Pareto-front computation in a multi-core environment*, presented at PARA 2008 Conference, Trondheim, Norway (2008), accepted for publication in LNCS 6126–6127.

[10] Kubica, B. J.; *Performance inversion of interval Newton narrowing operators*, presented at XII Conference on Evolutionary Algorithms and Global Optimization (KAEiOG 2009), Zawoja, Poland. Prace Naukowe Politechniki Warszawskiej. Elektronika 169, pp. 111–119, 2009.

[11] Nataraj, P. S. V., Prakash, A. K.; *A parallelized version of the covering algorithm for solving parameter-dependent systems of nonlinear equations*, Reliable Computing 8, pp. 123–130, 2002.

[12] Neumaier, A.; *The enclosure of solutions of parameter-dependent systems of equations*, in *Reliability in Computing* (ed. Moore, R.), Academic Press, 1988.

[13] Neumaier, A.; *Interval methods for systems of equations*, Cambridge University Press, Cambridge, 1990.

[14] Rheinboldt, W. C.; *Computation of critical boundaries on equilibrium manifolds*, SIAM Journal of Numerical Analysis 19, pp. 653–669, 1982.

[15] C-XSC interval library `http://www.xsc.de` .

[16] OpenMP `http://www.openmp.org` .

[17] Hungarian algorithm free implementation in C++ by John Waever `http://johnweaver.zxdevelopment.com/2007/05/22/munkres-code-v2/` .