

# A PARALLELIZED VERSION OF THE COVERING ALGORITHM FOR SOLVING PARAMETER - DEPENDENT SYSTEMS OF NONLINEAR EQUATIONS

P. S. V. NATARAJ AND A. K. PRAKASH

*Systems and Control Engineering Group  
Department of Electrical Engineering  
IIT Bombay 400 076, India.  
Email: nataraj@ee.iitb.ernet.in*

ABSTRACT. The so-called covering algorithm for enclosing the solution set of parameter - dependent systems of nonlinear equations has been recently proposed by Neumaier [12]. However, in the covering algorithm, only one box is processed in each iteration. This paper presents a parallelized version of the covering algorithm, in which *all* boxes present are processed *simultaneously* in each iteration. It is shown through several examples that this strategy results in speed-up of the algorithm by several orders of magnitude, particularly so in demanding problems. The proposed parallelized version can be run even on ordinary computers, i.e., it does not require a parallel computer.

## 1. INTRODUCTION

This paper addresses the problem of finding in a given box (i.e., a rectangular parallelepiped) all solutions of a nonlinear system with more variables than equations. This problem is clearly of a broad scope and has numerous applications in engineering and sciences.

The problem can sometimes be solved by one of the following methods [3] : (i) random search, (ii) an exhaustive grid search on the given box, (iii) more specialized or ad hoc methods, such as the Jenkins-Traub method for finding all roots of a single polynomial, and (iv) homotopy continuation methods [9]. The interested reader is referred to [3], [4] for a discussion and comparison of these methods.

In the frame work of interval analysis [7], Neumaier proposed the so-called covering algorithm [12] to solve the problem. Consider a finite-dimensional system of nonlinear equations of the form

$$(1) \quad F(\tilde{x}) = 0$$

where  $F$  is a function defined on a subset  $D \subseteq R^n$  with values in  $R^m$ ,  $m \leq n$ . If  $F$  is continuously differentiable in  $D$  and  $F'(\tilde{x})$  has rank  $m$  for all  $\tilde{x}$  in a neighborhood of the solution set

$$M = \{\tilde{x} \in D \mid F(\tilde{x}) = 0\}$$

then the solution set  $M$  of (1) is a  $p$ -dimensional manifold in  $R^n$ ,  $p = n - m$ . The vector  $\tilde{x}$  of variables often contains  $p$  distinguished variables, called as *parameters*.

We are interested in that part of  $M$  for which all variables (and parameters) lie within certain bounds

$$l_i \leq \tilde{x}_i \leq u_i \quad (i = 1, \dots, n)$$

so that only the solutions of (1) contained in a box  $x^0 = [l, u] \in IR^n$  are sought.

For any  $x \in ID$  (the set of interval boxes contained in  $D$ ), define

$$(2) \quad \sum (F, x) := \{\tilde{x} \in x \mid F(\tilde{x}) = 0\}.$$

The covering algorithm of Neumaier [12] consists of covering the set  $\sum (F, x^0)$  by a collection of smaller and smaller boxes which give increasingly accurate information about the location of the solution set. The algorithm uses the zero exclusion test and the generalized Gauss-Seidel method to discard irrelevant parts of  $x^0$ . However, in each iteration of the covering algorithm, only *one* box is processed.

The aim of this paper is to show that the covering algorithm can be speeded up by *several* orders of magnitude, if in each iteration of the algorithm, we simultaneously process *all* boxes that are present.

## 2. NEUMAIER'S COVERING ALGORITHM

We first outline Neumaier's covering algorithm to solve (1).

**Algorithm: Neumaier's covering algorithm** [12]

Inputs: the initial box  $x^0$ , a continuous interval extension (also denoted as  $F$ ) of the given function  $F$ , and a parameter  $\varepsilon$  to check if the width of a box is small.

Begin Algorithm

1. Enter the initial box into the stack.
2. If the stack is empty, go to step 9.
3. Choose the first box from the stack.
4. Discard irrelevant parts of the box using the zero exclusion test and the GGS method (see Remarks 2.1 and 2.2 below).
5. If the box is empty, go to step 2.
6. If the width of the box is less than or equal to  $\varepsilon$ , print the box and go to step 2.
7. Bisect the box along the maximum width coordinate direction and enter the halved boxes into the stack at the end.
8. Go to step 2.
9. Stop.

End Algorithm.

**Remark 2.1.** *The function is evaluated using the interval extension  $F$  and a box  $x$  from the stack. We may use a natural interval extension [7], e.g., if  $f(\tilde{x}) = 1 - 5\tilde{x} + 1/3\tilde{x}^2$ , then  $F(x) = 1 - 5x + 1/3x^2$  is the natural interval extension of  $f(\tilde{x})$ . As another example, if  $f(\tilde{x}) = \tilde{x}_1 * \sin \tilde{x}_2 - \tilde{x}_3$ , then  $F(x) = x_1 * \text{ISIN}(x_2) - x_3$  is the natural interval extension of  $f(\tilde{x})$ , where  $\text{ISIN}$  is the pre-declared interval  $\sin$  function in some programming language.*

**Remark 2.2.** *If  $0 \notin F(x)$  then  $x$  contains no solution point and is discarded (the zero exclusion test). Note that since  $F(x)$  generally overestimates the range  $\{F(\tilde{x}) \mid \tilde{x} \in x\}$ , there may or may not be a solution point in  $x$  if  $0 \in F(x)$ . In this case a more refined test is used as in the following remark.*

**Remark 2.3.** *The algorithm is speeded up by finding a smaller box containing all solutions in  $x$ , using the generalized Gauss-Seidel method (GGS) [12]. The GGS method is applied to the homogeneous linear interval equation*

$$(3) \quad \tilde{A}\tilde{d} = 0, \tilde{A} \in A, \tilde{d} \in d$$

where

$$(4) \quad \tilde{d} \in d := \begin{pmatrix} x - \tilde{z} \\ 1 \end{pmatrix}, \quad \tilde{A} \in A := (F'(x), F(\tilde{z}));$$

and  $\tilde{z} \in x$ . If this linear interval system is found incompatible then  $x$  can be discarded. Else, the algorithm proceeds by replacing  $x$  with  $x' = \tilde{z} + d'$ , where  $d'$  is the solution constructed using the GGS method. To further speedup the algorithm, preconditioning of the linear system is done, and interval slopes [5] instead of gradients are used.

### 3. PROPOSED ALGORITHM

Note that in each iteration of Neumaier's covering algorithm, only one box is processed. Such processing is inherently slow, due to its *sequential* nature. On the other hand, in each iteration of the covering algorithm, it is clearly possible to simultaneously process *all* boxes that are present (this can be done without altering in any way the essence of the algorithm). As will be seen below, the strategy results in greatly speeding up the algorithm, because all boxes present in every iteration are processed simultaneously, i.e., in a *parallel* manner. We call this version of the covering algorithm as the *parallelized* covering algorithm.

#### Algorithm: Parallelized covering algorithm

arises from Neumaier's covering algorithm by making the following changes.

- Step-3: Choose all the boxes present in the stack.
- Step-4: Discard irrelevant parts of all boxes:
  - (parallelized zero-exclusion test): using vectorized interval arithmetic operations, evaluate the interval extension  $F$  over all the boxes. Discard all those boxes for which  $0 \notin F(x)$ . If there are no more boxes remaining, go to step 9.
  - (parallelized GGS method) using vectorized gradient or slope evaluations, set up the  $A, d$  matrices in (3, 4) for all boxes. Using vectorized interval arithmetic operations, apply the GGS method simultaneously to all the resulting homogeneous linear interval equations, and obtain smaller boxes containing the solution points in the respective boxes.
- Step-5: Find and discard all empty boxes. If there are no more boxes remaining, go to step 9.
- Step-6: Using vectorized interval arithmetic operations, find the widths of all boxes. Then, find and print all those boxes satisfying the box-width condition (i.e., width of box  $\leq \varepsilon$ ). Discard the just printed boxes.
- Step-7: If there are no boxes remaining, go to step 9. Else, using vectorized interval arithmetic operations, find the maximum width coordinate directions for all boxes, and bisect simultaneously all the boxes along these (respective) directions. Enter all the resulting halved boxes into the stack.

**Remark 3.1.** The functions can be programmed in the following way to obtain parallelized evaluation. Consider, for example, the 1- dimensional manifold in [12]

$$F(x_1, x_2) = x_1^3 - x_1x_2^2 + x_1^2 - x_1x_2 - x_2^2$$

and suppose we want to evaluate  $F$  over a set of boxes taken from the stack. Then, using the notation of INTLAB (which is based on MATLAB), we can do this using the single program statement

$$F = \text{power}(x(:, 1), 3) - x(:, 1) .* \text{sqr}(x(:, 2)) + \text{sqr}(x(:, 1)) - x(:, 1) .* x(:, 2) - \text{sqr}(x(:, 2))$$

where,  $x(:, 1)$  denotes  $x_1$  for all boxes and  $x(:, 2)$  denotes  $x_2$  for all boxes. The function evaluation over all the boxes is done in a parallelized manner with this statement, because the operations  $+$ ,  $-$ ,  $.*$ ,  $\text{sqr}$ , and power are performed element-wise between vectors (cf. [1]) and INTLAB overloads these ordinary arithmetic operations with the corresponding interval arithmetic operations [13].

**Remark 3.2.** It is emphasized that the formulation of the algorithm is independent of MATLAB / INTLAB, and it can be run on any computer that has an interval arithmetic compiler supporting vectorized interval arithmetic operations, such as Forte Fortran 95 [2].

**Remark 3.3.** The parallelized covering algorithm does not require a parallel computer. That is, the parallelization is efficient even on serial architectures where vectorization brings advantages.

**Remark 3.4.** It follows from a result in [10, Theorem 4.15] that for  $\varepsilon > 0$ , the above algorithm terminates after at most  $\chi^n - 1$  iterations, where  $\chi = w(x^0)/\varepsilon$ .

#### 4. NUMERICAL RESULTS

We consider some examples for comparing the performance of the proposed parallelized covering algorithm with that of the covering algorithm. The examples are listed in Appendix.

All computations are carried out on a PC/Pentium-III 550 MHz machine with 384 MB RAM using INTLAB [13]. Tables 1 and 2 give the computational results for the various examples. The Tables list the number of boxes in the covering of the solution set, the execution time (seconds), and the number of floating operations (*flops*) taken. We used two values of  $\varepsilon$ , that is,  $\varepsilon = 0.01$  and  $0.001$ .

The following observations are made regarding the results given in the Tables :

1. The same number of covering boxes are obtained using either algorithm.
2. The proposed algorithm is faster than the covering algorithm in all examples. In all examples except example 3, the reduction in computational time is around 96 – 99%. In example 3, the same is around 30 – 40%.
3. The speedup is particularly attractive in those examples demanding large computational times using the covering algorithm. In such examples, the proposed algorithm is faster than the original one by up to 2 orders of magnitude.
4. The speedup factor gets better with accuracy.
5. The number of *flops* is less with the proposed algorithm in every example.

We conclude the paper with some remarks:

**Remark 4.1.** The observed reduction in computational overhead (in terms of *flops*) can be attributed to the fact that in INTLAB, interval arithmetic and slopes are done by operator overloading, which incurs substantial overhead. This overhead is needed once for each box in the (sequential) covering algorithm, but only once per major iteration in the parallelized version [11].

**Remark 4.2.** The actual speedup factor perhaps varies considerably with the computing environment used, and may be less conspicuous in programming languages where control structures are more efficiently implemented than in MATLAB.

## 5. CONCLUSIONS

A parallelized version of Neumaier's covering algorithm was proposed for solving finite-dimensional systems of parameter - dependent nonlinear equations. It was demonstrated through several test examples that the parallelized algorithm is significantly faster (by up to 2 orders of magnitude in demanding problems) than the original algorithm. Moreover, it is noteworthy that the parallelized algorithm does not require a parallel computer, but can be run on any computer (such as a PC) that has an interval arithmetic compiler supporting vectorized interval arithmetic operations.

## Acknowledgments

The authors are grateful to Prof. A. Neumaier for suggesting the parallelization of the covering algorithm and for his comments on a draft of the paper, and to Prof. S. M. Rump who inspired the latter with the idea through some parallel range computations using INTLAB. The authors also wish to thank Prof. Rump for the software INTLAB, which they have found a delight to use. Finally, the authors are grateful to the anonymous referees who suggested several improvements to the paper.

## REFERENCES

- [1] *MATLAB user guide, version 5.3*. The MathWorks Inc., MA, USA, 2000.
- [2] *Forte Fortran 95 user manual*. Sun Microsystems, Palo Alto, CA, USA, 2001.
- [3] R. B. Kearfott. Abstract generalized bisection and a cost bound. *Mathematics of Computation*, 49(179):187–202, 1987.
- [4] R. B. Kearfott. Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, 1987.
- [5] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM J. Numerical Analysis*, 22:604–616, 1985.
- [6] J. D. Lawrence. *A catalog of special plane curves*. Dover, New York, 1972.
- [7] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.
- [8] A. Morgan and V. Shapiro. Box bisection for solving second degree systems and the problem of clustering. *ACM Trans. Math. Software*, 13:152–167, 1987.
- [9] A. P. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice-Hall, Englewood Cliffs, N. J., 1987.
- [10] P. S. V. Nataraj and G. Sardar. Template generation for continuous transfer functions using interval analysis. *Automatica*, 36:111–119, 2000.
- [11] A. Neumaier. personal communication.
- [12] A. Neumaier. The enclosure of solutions of parameter dependent systems of equations. In R. E. Moore, editor, *Reliability in Computing: The Role of Interval Methods in Scientific Computations*. Academic Press, 1988.
- [13] S. M. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in reliable computing*. Kluwer Academic Publishers, 1999.
- [14] A. Ushida and L. O. Chua. Tracing solution curves of nonlinear equations with sharp turning points. *Circuit Theory and Applications*, 12:1–21, 1984.

Table 1: Comparisons of Algorithms for  $\varepsilon=0.01$ 

S.No	Examples	m	n	Solutions	Covering algorithm	Proposed algorithm	Speed up factor
1	One dimensional manifold [12]	1	2	covering boxes	2406	2406	100.3
				time(s)	180.55	1.6	
				<i>flops</i>	1354502	1060178	
2	Tunneling diode [14]	1	2	covering boxes	1473	1473	97.7
				time(s)	122.13	1.25	
				<i>flops</i>	1051102	783391	
3	Combustion chemistry [8]	2	2	covering boxes	2	2	1.79
				time(s)	1.81	1.09	
				<i>flops</i>	13865	12947	
4	Hippopede [6]	2	3	covering boxes	896	896	68.89
				time(s)	84.05	1.22	
				<i>flops</i>	795304	558268	
5	PUMA robot [8]	8	8	covering boxes	28	28	25.83
				time(s)	105.39	4.08	
				<i>flops</i>	1351569	1193709	

Table 2: Comparisons of Algorithms for  $\varepsilon=0.001$ 

S.No	Examples	m	n	Solutions	Covering algorithm	Proposed algorithm	Speed up factor
1	One dimensional manifold [12]	1	2	covering boxes	26955	26955	313.01
				time(s)	3274.1	10.46	
				<i>flops</i>	14218727	11142507	
2	Tunneling diode [14]	1	2	covering boxes	17060	17060	272.75
				time(s)	1693.8	6.21	
				<i>flops</i>	10011584	7503713	
3	Combustion chemistry [8]	2	2	covering boxes	2	2	1.47
				time(s)	2.15	1.46	
				<i>flops</i>	17575	16567	
4	Hippopede [6]	2	3	covering boxes	6884	6884	189.57
				time(s)	635.07	3.35	
				<i>flops</i>	5231611	3685222	
5	PUMA robot [8]	8	8	covering boxes	40	40	27.06
				time(s)	125.3	4.63	
				<i>flops</i>	1635829	1444654	

## APPENDIX A. LIST OF EXAMPLES

**Example 1 :** The 1- dimensional manifold in [12]

$$x_1^3 - x_1x_2^2 + x_1^2 - x_1x_2 - x_2^2 = 0$$

where,  $x_1 = [-3, 3]$ ,  $x_2 = [-5, 5]$ .

**Example 2 :** This is an equation of a simple tunneling diode [14]

$$0.43x_1^3 - 2.69x_1^2 + 4.56x_1 = 2.5x_2^2 - 10.5x_2 + 11.8x_2 = i$$

where  $x_1 = [0, 5]$ ,  $x_2 = [0, 3]$ , and  $i = 5$  mA.

**Example 3 :** This problem is an example from combustion chemistry [8]. The system consists of two cubic equations:

$$\begin{aligned}\alpha_1 x_1^2 x_2 + \alpha_2 x_1^2 + \alpha_3 x_1 x_2 + \alpha_4 x_1 + \alpha_5 x_2 &= 0 \\ \alpha_6 x_1^2 x_2 + \alpha_7 x_1 x_2^2 + \alpha_8 x_1 x_2 + \alpha_9 x_2^3 + \alpha_{10} x_2^2 + \alpha_{11} x_2 + \alpha_{12} &= 0\end{aligned}$$

where,  $\alpha_1 = -1.697 \times 10^7$ ,  $\alpha_2 = 2.177 \times 10^7$ ,  $\alpha_3 = 0.55$ ,  $\alpha_4 = 0.45$ ,  $\alpha_5 = -1$ ,  $\alpha_6 = 1.585 \times 10^{14}$ ,  $\alpha_7 = 4.126 \times 10^7$ ,  $\alpha_8 = -8.285 \times 10^6$ ,  $\alpha_9 = 2.284 \times 10^7$ ,  $\alpha_{10} = 1.918 \times 10^7$ ,  $\alpha_{11} = 48.4$ ,  $\alpha_{12} = -27.73$ . The box  $x_1 = [0, 1]$ ,  $x_2 = [0, 1]$ .

**Example 4 :** The hippopede problem in [6]

$$z = x_1^2 + x_2^2, \quad az = x_2^2 + z^2$$

where  $x_1 = [-1.5, 1.5]$ ,  $x_2 = [-1, 1]$ ,  $z = [0, 4]$ , and  $a = 1.1$ .

**Example 5 :** This is a set of kinematic equations for a PUMA robot in [8]

$$\begin{aligned}\gamma_1 x_1 x_3 + \gamma_2 x_2 x_3 + \gamma_3 x_1 + \gamma_4 x_2 + \gamma_5 x_4 + \gamma_6 x_7 + \gamma_7 &= 0 \\ \gamma_8 x_1 x_3 + \gamma_9 x_2 x_3 + \gamma_{10} x_1 + \gamma_{11} x_2 + \gamma_{12} x_4 + \gamma_{13} &= 0 \\ \gamma_{14} x_6 x_8 + \gamma_{15} x_1 + \gamma_{16} x_2 &= 0 \\ \gamma_{17} x_1 + \gamma_{18} x_2 + \gamma_{19} &= 0 \\ x_1^2 + x_2^2 - 1 &= 0 \\ x_3^2 + x_4^2 - 1 &= 0 \\ x_5^2 + x_6^2 - 1 &= 0 \\ x_7^2 + x_8^2 - 1 &= 0\end{aligned}$$

where,  $\gamma_1 = 4.731 \times 10^{-3}$ ,  $\gamma_2 = -0.3578$ ,  $\gamma_3 = -0.1238$ ,  $\gamma_4 = -1.637 \times 10^{-3}$ ,  $\gamma_5 = -0.9338$ ,  $\gamma_6 = 1$ ,  $\gamma_7 = -0.3571$ ,  $\gamma_8 = 0.2238$ ,  $\gamma_9 = 0.7623$ ,  $\gamma_{10} = 0.2638$ ,  $\gamma_{11} = -0.07745$ ,  $\gamma_{12} = -0.6734$ ,  $\gamma_{13} = -0.6022$ ,  $\gamma_{14} = 1$ ,  $\gamma_{15} = 0.3578$ ,  $\gamma_{16} = 4.731 \times 10^{-3}$ ,  $\gamma_{17} = -0.7623$ ,  $\gamma_{18} = 0.2238$ ,  $\gamma_{19} = 0.3461$ . The box  $x_i \in [-1, 1]$ ,  $i = 1, \dots, 8$ .