An Intelligent Hybrid Algorithm for Solving Non-linear Polynomial Systems*

Jiwei Xue[1,2], Yaohui Li[1], Yong Feng[1], Lu Yang[1], and Zhong Liu[1]

[1] Chengdu Institute of Computer Applications, Chinese Academy of Sciences, Chengdu 610041, P. R. China
xuejiwei@163.com, {phillip138, mathyfeng}@hotmail.com
[2] Computer Science and Engineering College, Daqing Petroleum Institute, Daqing 163318, P. R. China

Abstract. We give a hybrid algorithm for solving non-linear polynomial systems. It is based on a branch-and-prune algorithm, combined with classical numerical methods, symbolic methods and interval methods. For some kinds of problems, Gather-and-Sift method, a symbolic method proposed by L. Yang, was used to reduce the dependency of variables or occurrences of the same variable, then interval methods were used to isolate the real roots. Besides these, there are some intelligent judgments which can improve the system's efficiency significantly. The algorithm presented here works rather efficiently for some kinds of tests.

1 Introduction

In this paper, we address the problem of finding all solutions to polynomial systems, a fundamental and important problem in the research of real algebra from the viewpoint of algorithm research. This is an old problem and there have been some works concerning this issue. Several interesting methods have been proposed in the past for this task, including two fundamentally different methods: numerical methods[4,5,6,9,10,11,15] and symbolic methods[1,2,20,21, 22,23,24].

Classical numerical methods start from some approximate trial points and iterate. Thus, there is no way to guarantee correctness (i.e. finding all solutions) and to ensure termination. Interval methods can overcome these two shortcomings but tend to be slow.

Symbolic computation plays important role in applied mathematics, physics, engineering and other areas. But currently it is only possible to solve small examples, because of the inherent complexity of the problems in symbolic computation. Symbolic methods include Ritt-Wu method, Gröbner basis methods or resultant methods [3], but all these methods are time consuming, especially when the number of variables > 10.

In order to improve the efficiency of the system, we propose an *intelligent hybrid* algorithm. *Hybrid* means we combine numerical methods, interval methods

* This research was partially supported by NSFC (10172028).

and symbolic methods. *Intelligence* means before using interval methods we will use our knowledge to tight the starting box, or use classical numerical methods to approximate the root directly once we can ensure that only one root exists.

The rest of this paper is structured as follows: *Gather-and-Sift* algorithm and some improvements are presented in section 2. Section 3 devotes to univariate and multivariate interval Newton methods. Section 4 presents some of improvements made in our method in order to improve the efficiency. Some examples and their results are given in Section 5. Section 6 concludes the paper.

In this paper, boldface (e.g. $\boldsymbol{x}$, $\boldsymbol{y}$) will denote intervals, lower case (e.g. x, y) will denote scalar quantities, and upper case (e.g. A, B) will denote vectors or matrices, bold upper case (e.g. $\boldsymbol{A}$, $\boldsymbol{B}$) will denote interval vectors (or *boxes*). Brackets "[ ]" will delimit intervals. Underscores $\underline{x}$ will denote lower bounds of intervals and overscores $\bar{x}$ will denote upper bounds of intervals. The set of real intervals will be denoted by IR.

2 Gather-and-Sift Algorithm

Gather-and-Sift algorithm[23,24], which was proposed by L. Yang et al. in 1995, is a very efficient method in solving nonlinear algebraic equation system both of parametric coefficients and of numeric coefficients. *Gather* means to construct some ascending chains whose zeros contain all the zeros of the original systems. *Sift* means to remove extra zeros from ascending chains such that only the required ones remain. GAS, a MAPLE program based on DIXON resultant, will be called before interval methods are used if the number of variables ≤ 4 in our system. The effect of this modification can be seen from *Example 1*. We will give a sketch of *Gather-and-Sift* method and modifications have been made, for details and further references about *Gather-and-Sift* see [13,23,24].

2.1 A Sketch of *Gather-and-Sift*

Given a system PS consisting of k polynomials in k indeterminates, *Gather-and-Sift* algorithm can be summarized as follows:

Step 1. Regarding x_1 as a parameter, construct a polynomial system DPS, which is the *Dixon derived polynomial set* of PS with respect to $\{x_2, \cdots, x_k\}$;

Step 2. Transform DPS into the following *standard form* (x_1 was regarded as a parameter):

$$DPS : \begin{cases} q_1 = c_{1n}e_n + \cdots + c_{12}e_2 + c_{11}e_1 = 0, \\ q_2 = c_{2n}e_n + \cdots + c_{22}e_2 + c_{21}e_1 = 0, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ q_m = c_{mn}e_n + \cdots + c_{m2}e_2 + c_{m1}e_1 = 0 \end{cases} \tag{1}$$

where $e_n, \cdots, e_1$ represent all the power products of $x_k, x_{k-1}, \cdots, x_2$ appeared in DPS sorted into a decreasing order according to a lexicographical order or a degree order;

Step 3. Do a fraction-free Gaussian elimination for the above system DPS, which is a linear equation system in $e_n, \cdots, e_1$, then we have:

$$GPS : \begin{cases} h_m = b_{mn}e_n + \cdots + b_{m2}e_2 + b_{m1}e_1 = 0, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ h_2 = b_{22}e_2 + b_{21}e_1 = 0, \\ h_1 = b_{11}e_1 = 0 \end{cases} \tag{2}$$

where $b_{ij} \in K[x_1](i = 1, 2, \cdots, m; j = 1, 2, \cdots, n)$. Now regard x_1 as a indeterminate, GPS will be written as follows:

$$GPS : \begin{cases} h_1 = h_1(x_1, x_2, \cdots, x_k) = 0, \\ h_2 = h_2(x_1, x_2, \cdots, x_k) = 0, \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ h_m = h_m(x_1, x_2, \cdots, x_k) = 0 \end{cases} \tag{3}$$

The above three steps is called GPS algorithm, and a generic program for this algorithm written in Maple was called GPS program.

Step 4. Select k polynomials from GPS to form a *triangular form* TS in $\{x_1, x_2, \cdots, x_k\}$;

Step 5. Establish normal ascending chain ASC (or normal ascending chains $ASC_1, \cdots, ASC_l$) from the system TS resulting from last step;

Step 6. For every normal ascending chain ASC, do *relatively simplicial decomposition* w.r.t. PS by using WR method. This step can sift out the extra zeros.

2.2 Some Improvements on the *Gather-and-Sift* Method

From the above algorithm we can see that if a *triangular form* in k indeterminates cannot be found, the efficiency of the *Gather-and-Sift* method will be reduced greatly. The following two improvements have been made to increase the possibility of finding the *triangular form.*

Unknown-Order-Change. By calling GPS program, the output polynomial set GPS has great difference in form if the given sequence of the indeterminates is different. Sometimes, you even cannot find a *triangular form* in all variables from GPS directly. The possibility of finding the *triangular form* can be increased by Z. Liu's method, i.e., unknown-order-change. It can be described as follows:

```
BEGIN
    FOR i FROM 1 TO n DO
        regard the ith arrange of the given indeterminates as the current sequence;
        regard the first element of current sequence as parameter and call GPS;
        select a triangular form $TS$ in all indeterminates from GPS;
        IF such $TS$ can be found THEN
            return $TS$;
        END IF;
    ENDDO
END
```

Extension of The Polynomial Set. By experiments we also found such a fact: sometimes a *triangular form TS* cannot be found just because of shortage of polynomial in some indeterminates in GPS, but such a polynomial can easily be found in the original polynomial set.

Z. Liu proposed an extension of the polynomial set method to further increase the possibility of finding the *triangular form*. The method can be summarized briefly as: after running the GPS algorithm, add the original polynomial set PS into GPS and the result still be denoted by GPS, then try to select a *triangular form* in all indeterminates from GPS. If succeed, output the TS, otherwise try to use the next arrange of the indeterminates to redo the above steps.

3 Interval Newton Methods

Modern development of interval arithmetic began with R. E. Moore's dissertation in 1962. Since then thousands of research articles and numerous books have appeared on the subject. For details and further references about interval arithmetic, see[4,5,6,7,9,10,11,14,15,16,18,19].

The classical Newton method does not mathematically guarantee to find *all* roots within a given domain. Computational result obtained by finite precision arithmetics may not be reliable both mathematically and computationally. To overcome these problems, extensive studies on interval Newton methods e.g. [8, 10,12,16,17,19] have been done.

Interval Newton methods combine the *classical Newton method*, the *mean value theorem* and *interval analysis*. These methods may be used both to discard root free subintervals, and to replace subintervals by smaller ones via a rapidly converging iteration scheme.

3.1 Univariate Interval Newton Methods

Suppose $f : \boldsymbol{x} = [\underline{x}, \bar{x}] \rightarrow R$ has a continuous first derivative on $\boldsymbol{x}$, suppose that there exists $x^* \in \boldsymbol{x}$ such that $f(x^*) = 0$, and suppose that $\check{x} \in \boldsymbol{x}$. Then, since the *mean value theorem* implies $0 = f(x^*) = f(\check{x}) + f'(\xi)(x^* - \check{x})$ we have $x^* = \check{x} - f(\check{x})/f'(\xi)$ for some $\xi \in \boldsymbol{x}$. If $\boldsymbol{f}'(\boldsymbol{x})$ is any interval extension of the derivative of f over $\boldsymbol{x}$, then

$$x^* \in \check{x} - f(\check{x})/\boldsymbol{f}'(\boldsymbol{x}) \text{ for any } x^* \in \boldsymbol{x}. \tag{4}$$

From equation (4) we can get the *univariate interval Newton operator*:

$$\boldsymbol{N}(f, \boldsymbol{x}, \check{x}) = \check{x} - f(\check{x})/\boldsymbol{f}'(\boldsymbol{x}) \tag{5}$$

It is well known that $\boldsymbol{N}(f, \boldsymbol{x}, \check{x})$ has the following properties:

1. If $x^* \in \boldsymbol{x}$ and $f(x^*) = 0$, then $x^* \in \boldsymbol{N}(f, \boldsymbol{x}, \check{x})$;
2. If $\boldsymbol{x} \cap \boldsymbol{N}(f, \boldsymbol{x}, \check{x}) = \emptyset$, then $\forall x \in \boldsymbol{x},\ f(x) \neq 0$;
3. If $\boldsymbol{N}(f, \boldsymbol{x}, \check{x}) \subset \boldsymbol{x}$, then $\exists x^* \in \boldsymbol{x},\ f(x^*) = 0$.

3.2 Multivariate Interval Newton Methods

Multivariate interval Newton methods are analogous to univariate ones, the iteration step is as follows:

$$\begin{cases} \boldsymbol{F}'(\boldsymbol{X}_k)(\tilde{\boldsymbol{X}}_k - X_k) = -F(X_k) \\ \boldsymbol{X}_{k+1} = \boldsymbol{X}_k \cap \tilde{\boldsymbol{X}}_k \end{cases} \tag{6}$$

where k=0,1,2,$\cdots$, $\boldsymbol{F}'(\boldsymbol{X}_k)$ is a suitable interval extension of the Jacobian matrix over the box $\boldsymbol{X}_k$ (with $\boldsymbol{X}_0 = \boldsymbol{X}$), and where $X_k \in \boldsymbol{X}_k$ represents a predictor or initial guess point.

4 Some Improvements Made in Our Method Besides *Gather-and-Sift*

4.1 Intelligence+Numerical Method+Interval Arithmetic

The classical numerical method's disadvantages include: incorrectness (i.e. finding all solutions) and unreliability. For some applications each variable's degree is *one*, which means that there is only one solution to the equation. So before use *Interval Newton methods*, first judge if the equation only has one solution by collecting the maximal degree of the variables appeared in the polynomial systems. If the maximal degree is equal to one, then we can use the following method to isolate the root. Because there is only one solution so the correctness can be guaranteed, while numerical reliability is obtained by using interval arithmetic. From *Example 2*, it can be seen that this intelligent method can greatly improve the system's performance.

4.2 Numerical Method+Interval Arithmetic

As we have realized that *classical Newton method* can be made more efficient if the initial value was chosen close to the root. This is also true to *interval Newton methods*. Until now, most people choose the midpoint of the interval as the initial value, for lots of tests, this will need many times of *Interval Newton* iteration. In our method, after knowing that there is a root in a certain interval, we will use an ordinary numerical method (e.g. *classical Newton method*, *classical Quasi-Newton method*) to compute the approximation to the root, then use *Interval Newton method* to bound the *error*.

5 Examples and Results

In this section we report the performance of our method on some examples. All results were obtained by running our system on a PC (Pentium 566MHz CPU, 256Mb of main memory) with Maple 9.

Example 1. The system $\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ x_1^2 - x_2 = 0 \end{cases}$ can be found in many papers. Given initial interval vector $[-10^8, 10^8]^2$, tolerance $\varepsilon = 10^{-8}$, the comparison results without and with calling GAS are given as follows:

1. Without calling GAS, the following two intervals are achieved after 0.641s:

[[-0.7861513793, -0.7861513766], [0.6180339869, 0.6180339912]]
[[0.7861513766, 0.7861513793], [0.6180339869, 0.6180339912]]

2. While if GAS is called firstly, after 0.010s it gives the following result:

$$\begin{cases} -x_1^2 + 1 - x_1^4 = 0 \\ x_1^2 - x_2 = 0 \end{cases}$$

Then do as without calling GAS, we will get the following result after another 0.100s, i. e., it will cost 0.110s totally to do the same task.

[[-0.7861513779, -0.7861513776], [0.6180339885, 0.6180339890]]
[[0.7861513776, 0.7861513779], [0.6180339885, 0.6180339890]]

Example 2. The following system is an examples given by Moore and Jones [6].

$$\begin{cases} x_1 - 0.25428722 - 0.18324757 x_4 x_3 x_9 = 0 \\ x_2 - 0.37842197 - 0.16275449 x_1 x_{10} x_6 = 0 \\ x_3 - 0.27162577 - 0.16955071 x_1 x_2 x_{10} = 0 \\ x_4 - 0.19807914 - 0.15585316 x_7 x_1 x_6 = 0 \\ x_5 - 0.44166728 - 0.19950920 x_7 x_6 x_3 = 0 \\ x_6 - 0.14654113 - 0.18922793 x_8 x_5 x_{10} = 0 \\ x_7 - 0.42937161 - 0.21180486 x_2 x_5 x_8 = 0 \\ x_8 - 0.07056438 - 0.17081208 x_1 x_6 x_7 = 0 \\ x_9 - 0.34504906 - 0.19612740 x_6 x_8 x_{10} = 0 \\ x_{10} - 0.42651102 - 0.21466544 x_1 x_4 x_8 = 0 \end{cases}$$

Given starting box $[-1, 1]^{10}$ and tolerance $\varepsilon = 10^{-8}$, it costs 0.180s gives the following interval as the result: $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}] =$
$[[0.2578333932, 0.2578333943], [0.3810971543, 0.3810971550],$
$[0.2787450173, 0.2787450174], [0.2006689638, 0.2006689647],$
$[0.4452514243, 0.4452514254], [0.1491839199, 0.1491839201],$
$[0.4320096989, 0.4320096991], [0.07340277776, 0.07340277779],$
$[0.3459668268, 0.3459668269], [0.4273262759, 0.4273262760]]$

Without changing the tolerance, if the starting box was taken as $[-10, 10]^{10}$, the traditional method (without intelligence) does not terminate after running 7200s. But by running the intelligent analyzing module in our method, we know that if this equation has root in the given interval it will only has one root. So we can use numerical method combined with interval method to get the result interval, it only costs 0.190s. Furthermore, if GAS is called firstly, it does not terminate after running 3600s.

From *Example 1* and *Example 2*, we can get the following conclusion: Hybrid method without any consideration sometimes will not improve the systems's efficiency, contrarily, it may make the situation even worse.

Example 3. This is another standard benchmark given by Moore and Jones [5].

$$\begin{cases} x_1 - 0.25428722 - 0.18324757x_4^3x_3^3x_9^3 + x_3^4x_9^7 = 0 \\ x_2 - 0.37842197 - 0.16275449x_1^3x_{10}^3x_6^3 + x_{10}^4x_6^7 = 0 \\ x_3 - 0.27162577 - 0.16955071x_1^3x_2^3x_{10}^3 + x_2^4x_{10}^7 = 0 \\ x_4 - 0.19807914 - 0.15585316x_7^3x_1^3x_6^3 + x_1^4x_6^7 = 0 \\ x_5 - 0.44166728 - 0.19950920x_7^3x_6^3x_3^3 + x_6^4x_3^7 = 0 \\ x_6 - 0.14654113 - 0.18922793x_8^3x_5^3x_{10}^3 + x_5^4x_{10}^7 = 0 \\ x_7 - 0.42937161 - 0.21180486x_2^3x_5^3x_8^3 + x_5^4x_8^7 = 0 \\ x_8 - 0.07056438 - 0.17081208x_1^3x_6^3x_7^3 + x_7^4x_6^7 = 0 \\ x_9 - 0.34504906 - 0.19612740x_6^3x_8^3x_{10}^3 + x_6^4x_8^7 = 0 \\ x_{10} - 0.42651102 - 0.21466544x_1^3x_4^3x_8^3 + x_8^4x_1^7 = 0 \end{cases}$$

Given starting box $[-1, 1]^{10}$ and tolerance $\varepsilon = 10^{-8}$, it costs 0.381s gives the following interval as the result: $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}] =$

$$\begin{aligned} &[[0.2542852239, 0.2542852240], [0.3784225742, 0.3784225743], \\ &[0.2715848389, 0.2715848390], [0.1980797710, 0.1980797711], \\ &[0.4416682234, 0.4416682235], [0.1464438740, 0.1464438741], \\ &[0.4293719571, 0.4293719572], [0.07056502913, 0.07056502914], \\ &[0.3450490767, 0.3450490768], [0.4265110279, 0.4265110280]] \end{aligned}$$

6 Conclusion and Future Work

In this paper, we have studied a hybrid method for isolating real solutions of polynomial systems. On the one hand, we use interval Newton methods in conjunction with bisection methods to overcome classical numerical methods' shortcomings; on the other hand, we use classical numerical methods to remedy interval methods' deficiency (i. e., slow). But there are also some problems deserve further study.

1. We use classical Quasi-Newton method, a superlinear convergence method, to approximate the root. Next, we can use some high-order convergence methods to further increase the efficiency of the algorithm.

2. It is computationally very expensive for polynomials with multiple occurrences of the same variables. Next, we will use more symbolic methods (e.g., Gröbner basis, Wu-method) to reduce the dependency of variables or occurrences of the same variables. But it is well known that all symbolic methods are time consuming (e.g., *Example 2*), so we must further study how to cooperate different methods and the extent of cooperation.

3. We will further study human knowledge which can be used in our method to increase the system's performance.

References

1. Collins, G.E., Loos, R.: Real Zeros of Polynomials. Computer Algebra:Symbolic and Algebraic Computation (1983)

2. Collins, G.E., Johnson, J.R., Krandick, W.: Interval Arithmetic in Cylindrical Algebraic Decomposition. Journal of Symbolic Computation. 34 (2002) 145-157
3. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Springer-Verlag, New York, USA (1992)
4. Hentenryck, P.V., Michel, L., Benhamou, F.: Newton: Constraint Programming over Nonlinear Constraints. Science of Computer Programing. 30(1-2)(1998) 83-118
5. Hentenryck, P.V., McAllester, D., Kapur, D.: Solving Polynomial Systems Using a Branch and Prune Approach. SIAM Journal on Numerical Analysis. 34(2)(1997) 797-827
6. Herbort, S., Ratz, D.: Improving the Efficiency of a Nonlinear Systems Solver Using a Componentwise Newton Method. http://citeseer.nj.nec.com/herbort97improving.html (1997)
7. Hickey, T., Ju, Q., van Emden, M.H.: Interval Arithmetic: From Principles to Implementation. Journal of ACM. 48(5)(2001) 1038-1068
8. Hu, C.Y.: Reliable Computing with Interval Arithmetic. Proceeding of the International Workshop on Computational Science and Engineering '97.
9. Kearfott, R.B., Hu, C.Y., Novoa III, M.: A Review of Preconditioners for the Interval Gauss-Seidel Method. Interval Computations. 1(1)(1991) 59-85
10. Kearfott, R.B.: Interval Computations: Introduction, Uses and Resources. Euromath Bulletin. 2(1)(1996) 95-112
11. Kearfott, R.B., Shi, X.F.: Optimal Preconditioners for Interval Gauss-Seidel Methods. Scientific Computing and Validated Numerics, Akademie Verlag (1996) 173-178
12. Kearfott, R.B., Walster, G.W.: Symbolic Preconditioning with Taylor Models: Some Examples. Reliable Computing. 8(6)(2002) 453-468
13. Liu, Z.: Gather-and-Sift Software GAS Based on DIXON Resultant. Chengdu Institute of Computer Applications, Chinese Academy of Sciences (2003)(Dissertation)
14. Moore, R.E., Yang, C.T.: Interval Analysis I. (1959) 1-49 (Technical document)
15. Ratz, D., Karlsruhe.: Box-splitting Strategies for the Interval Gauss-Seidel Step in a Global Optimization Method. Computing. 53 (1994) 337-353
16. Ratz, D.: On Extended Interval Arithmetic and Inclusion Isotonicity. Institut für Angewandte Mathmatik, Universität Karlsruhe (1996)
17. Revol, N.: Reliable an Daccurate Solutions of Linear and Nonlinear Systems. SIAM Conference on Optimization, Toronto, Ontario, Canada, 20-22 May, 2002.
18. Schichl, H., Neumaier, A.: Interval Analysis - Basics. In: http://solon.cma.univie.ac.at/ neum/interval.html (2003)
19. Stahl, V.: Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations (1995) (Dissertation)
20. Wu, W.T.: On Zeros of Algebraic Equations-An Application of Ritt Principle. Kexue Tongbao. 31 (1986) 1-5
21. Xia, B.C., Yang, L.: An Algorithm for Isolating the Real Solutions of Semi-algebraic Systems. Journal of Symbolic Computation. 34 (2002) 461-477
22. Xia, B.C., Zhang, T.: Algorithm for Real Root Isolation Based on Interval Arithmetic (2003) (draft)
23. Yang, L., Hou, X.R.: Gather-And-Shift: a Symbilic Method for Solving Polynomial Systems. Proceedings for First Asian Technology Conference in Mathemtics,18-21 December 1995, Singapore (1995) 771-780
24. Yang, L., Zhang, J.Z., Hou, X.R.: Nonlinear Algebraic Equation System and Automated Theorem Proving. ShangHai Scientific and Technological Education Publishing House, ShangHai (1996) (in Chinese)