Substitution-based CHR Solver for Bivariate Binomial Equation Sets

Alia el Bolock, Amira Zaki, and Thom Frühwirth

Ulm University, Institute of Software Engineering and Compiler Construction, Germany
{aliaelbolock}@gmail.com,{amira.zaki,thom.fruehwirth}@uni-ulm.de

Abstract. Various methods for solving non-linear algebraic systems exist, as this question is amongst the most popular in both the realm of mathematics and computation. As most of these methods use approximations, this work focuses on finding and directly solving a tractable subset. Bivariate binomial systems of non-linear polynomial equations were chosen and solved by simulating the by hand method, using the declarative logic programming language Constraint Handling Rules. Substitution methods and different equation notations are used to extend the solvability of the subset.

Key words: Binomial, Bivariate, Constraint Handling Rules, Non-linear, Polynomial, Substitution

1 Introduction

The world we live in is to the most part non-linear. Thus, it is natural that non-linear systems reoccur everywhere around us. Diverse fields of science and life, such as mechanics, robotics, chemistry and economics require solving non-linear systems for their basic applications. The special case of polynomial systems occurs even more frequently in the real world and has the advantage of being simpler than random non-linear systems and easier to visualize.

Solving non-linear algebraic systems of equations, polynomial and non-polynomial, is a very important subfield of mathematics, as non-linear systems of equations can not be solved quantitatively but to the most part only through approximations. Over the years many different methods for solving non-linear polynomial and non-polynomial systems of equations have been developed. The most common approaches for dealing with non-linear equations are either numerical or symbolic [18, 11, 12], continuation [16], reduction [19, 20] or iterative and interval methods [17, 14, 5, 9, 13], and sometimes even a combination of them, for example in most computer algebra tools [26] and [4]. But only one of these algorithms is based on the logic programming paradigm using the rule based programming language Constraint Handling Rules, namely INCLP($\mathbb{R}$) [6], which is also based on approximating results of a non-linear system.

The aim of this work and the conducted research is finding a tractable subset of non-linear systems of equations, for which exact roots can be efficiently

and non-aproximatively calculated and providing a Constraint Handling Rules (CHR) solver for said subset. The subset chosen is that of binomial bivariate equation sets and polynomial equations of degrees up to four. The method derived for solving this subset, is one that simulates one of the possible approaches of humans when met with such problem sets. The implemented solver artificially reanimates what humans would do by hand, always baring in mind the most efficient approach given the problem set and the advantages provided by CHR. It basically uses isolation and substitution methods for solving the bivariate system of non-linear equations.

The majority of the various pre-existing methods for solving non-linear polynomial equation sets, especially those in the context of constraint programming, are based on interval and approximation methods. This work focuses on trying to find the largest subset that can be solved *exactly* and thus having the highest precision possible.

The problem field is narrowed down to cover non-linear polynomial equations. Starting from bivariate systems, alongside univariate equations with degrees less than five, accuracy and solvability could be ensured. As proof of concept, the solver algorithm was tested for the binomial case.

2 Concepts

2.1 Algebra

Properties of Equations Univariate equations are ones with one variable, while bivariate equations have two variables. We distinguish between univariate and multivariate polynomials, meaning polynomials with only one variable and multiple variables respectively.

Non-linear System of Equations and its Roots A non-linear system of equations is a set of n equations, containing at least one non-linear equation; meaning an equation with degree not equal to one. Finding the roots of the system of equations, means finding a vector $x = (x_1, ..., x_n)$ that simultaneously solves all equations within the systems [25]. Non-linear systems of equations can either have a finite number of solutions, infinite solutions (consistent) or no solutions at all (inconsistent). Under-defined systems of equations are ones with more variables than equations, while an over-defined system has more equations than variables.

Polynomials A polynomial is a finite sum of terms with non-negative degrees. A polynomial consisting of one term is called monomial and that of two terms is called binomial. The standard form of polynomial equations is $P(x) = c_n x^n +$

$c_{n-1}x^{n-1} + ... + c_0$, where $c_i \in \mathbb{Q}$, $n \in \mathbb{Z}$, $c_n \neq 0$ and $n \neq 0$. According to the fundamental theorem of algebra, any polynomial equation can be expressed as $P(x) = c_n(x-r_1)(x-r_2)...(x-r_n)$, where $r_i \in \mathbb{C}$ are the roots of the polynomial. Such roots can be directly determined for univariate polynomial equations with degrees up to four, for which solution formulae exist [21, 15]. It was however proven that no such formulae could exist for polynomial equation with higher degrees [22–24].

2.2 Constraint Handling Rules

Constraint Handling Rules (CHR) is a high-level, constraint-based, declarative logic programming language, invented by Prof. Thom Frühwirth in 1991. CHR adapts the basic concepts of mathematical logic representation and is thus highly and easily applicable to various problems. CHR is a committed-choice, single-assignment language, with multi-headed rules and conditional rule application through guards. Having simplification, propagation and simpagation (a mixture of the afore mentioned rules) as the only operators that can deal with constraints, CHR is well suited for representing mathematical problems and solving them straightforwardly. The properties of CHR enable the user to design anytime, online, confluent and concurrent algorithms, depending on the semantics used. More detailed explanations of CHR, its properties and advanced examples, can be found in [7].

3 Solution Algorithm

The chosen methodology for solving an input system of equations, is based on the usual thought procedure most humans would follow. The implemented solver gives a numeric solution for a non-linear input system. Given a set of two equations, the solver basically first tries to turn one of them into a univariate equation, by isolating one of the system variables. Then, this univariate equation is solved and its solutions renders the second equation univariate, in which case it can in turn be solved. This can be achieved for equations in one of the two solvable cases: an already univariate equation, or an equation with a singly occurring variable that either stands alone or is part of a term. The subset of univariate equations with solution formulae can be directly solved, the remainder is simplified. Should this not be directly applicable, then some substitutions are to be done to transform the equation set into one that can be solved by the above mentioned method. Figure 1 gives the flow diagram of this solving algorithm.

The equations' terms are ordered before the check for equations in the directly solvable cases is made, to be sure that the leading term is always the simplest term, when needing to isolate it or to take it as a reference point for subsitutions. The order of a term depends on the powers of its variables and the second term variable is taken as the reference point to give the priority in isolation to the variable x, thus colexicographic ordering is used. The colexicographic ordering

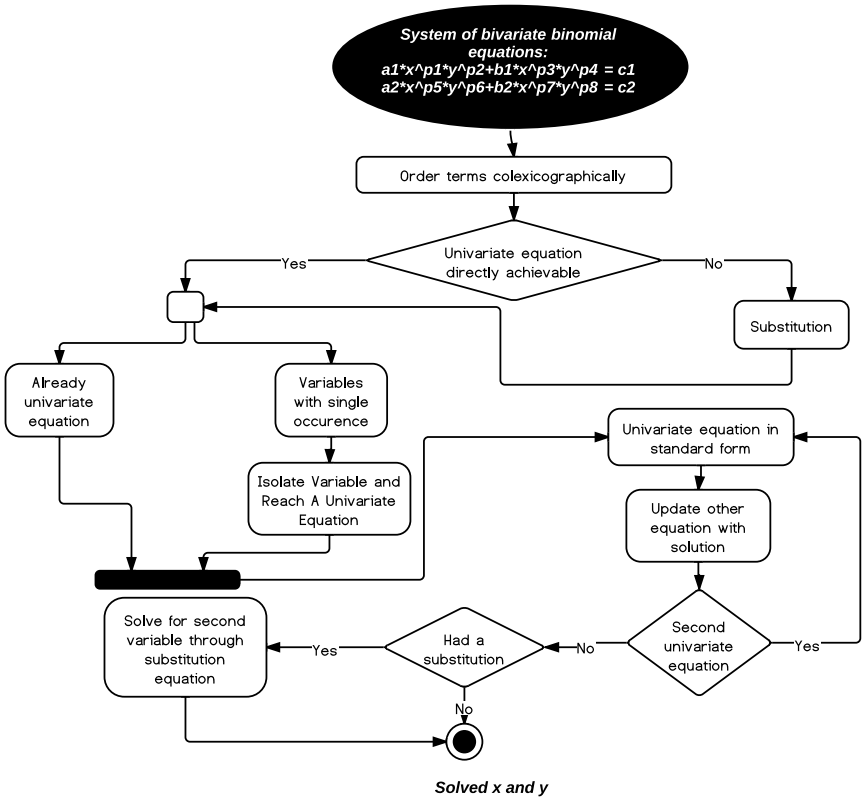

Fig. 1. Flow diagram of the solver algorithm

of two pairs $x^{n1}y^{m1}$ and $x^{n2}y^{m2}$ is defined as follows:

$$colex : x^{n1}y^{m1} \leq x^{n2}y^{m2} \Leftrightarrow m1 < m2 \vee (m1 = m2 \wedge n1 \leq n2). \qquad (1)$$

```
order_eq_exchange @ order_eq([H1,H2] eq C) <=> lex(H2,H1)
                                             | [H2,H1] eq C.
```

The helper predicate `lex(H1,H2)` is true if `H1` is colexicographically less than `H2`. If univariate equations are found they are transformed into the standard form, else the most optimal system variable is isolated in one equation and used to render the other equation univariate. If the set is not in a directly solvable state then the in 3.2 explained substitution is applied before the set is sent back to the initial solving state. All univariate equations in standard form with degrees less than five are solved and their solution produces a second univariate equation to be solved, possibly with the help of the substitution equation. An example for the realization of part of the quartic formula [15] is:

```
A*X^4+B*X^4-E ueq 0 ==> C=0, D=0, F is C-(3*B**2/8), G is (D+(B**3/8))-(B*C/2),
                        H is (E-(3*B**4/256))+((B**2*C/16)-(B*D/4)),
```

```
                        I is F/2, J is (F**2-4*H)/16, K is (-G**2)/64,
                        1*Z^3+I*Z^2+J*Z^1+K ueq 0.

A*X^3+B*X^2+C*X^1+D ueq 0 <=> F is (((3*C)/(A))-(B**2/A**2))/3,
                  G is ((2*B**3/A**3)-(9*B*C/A**2)+((27*D)/(A)))/27,
                  H is (G**2/4)+(F**3/27), H > 0
                | R is (-G/2)+(H**(1/2)), cubic_root(R,S),
                  T is (-G/2)-(H**(1/2)), cubic_root(T,U),
                  X1 is (-((S+U)/2)-(B/3*A)),
                  XI1 is ((S-U)*(3**(1/2))/2),
                  XI2 is (-((S-U)*(3**(1/2))/2)),
                  solved_img(X1,XI1,X1,XI2).

solved_img(Y1,YI1,Y1,YI2),A*X^4+B*X^3-E ueq 0 <=>
                      C=0, D=0, G is (D+(B**3/8))-(B*C/2),
                      img_sqrt(Y1,YI1,PR,PI), S is (B)/(4*A),
                      R is (-G)/(8*(PR**2+PI**2)), X1 is (PR+PR)+(R-S),
                      X2 is (R-S)-(PR+PR), (X=X1;X=X2), solved(X).
```

This shows how the quartic equation is reduced to a cubic one whose solutions are then substituted back into the orginal quartic equation to solve it. In case the helper cubic equation has two imaginary and one real solution, the imaginary solutions are chosen and their real and imaginary parts are seperatly forwarded to the quartic equation using the `solved_img/4` constraint. The auxiliary `img_sqrt/2` predicate, extracts the cubic root of a negative number, as this option is not supported by Prolog.

The algorithm is characterized by its simplicity while covering a wide subset. This is enabled by the different equation representations used and the substitution scheme.

3.1 Different Equation Representations

The simplicity and efficiency of the algorithm is ensured by using different representations and notations for equations, to distinguish between types of equations and phases of the algorithm. Equations are expressed as constraints to benefit from the features of CHR. As the equations are notated differently depending on the state they are in, the rules are fired voluntarily and no explicit iterations need to be done. This ensures that any possible solutions are calculated and possible simplifications are done at any given point, exploiting the online property of CHR. There are two notations for equations in this solver, and the different equality constraints belong to different notations.

The standard equation notation `A*X^P1*Y^P2+B*X^P3*Y^P4 eq C` is where the equation most resembles the normal mathematical form of equations. The '=' sign is replaced by other constraints e.g. the `eq` constraint, depending on the phase of execution of the algorithm. The `ueq/2` constraint for example indicates a univariate equation, while the `req/2` constraint means an equation with an isolated variable and the `deq/2` indicates a not yet handled equation and that the other equation has been already simplified or solved.

```
X req W \ L eq C <=>
              L = [C1*pot(X,P1)*pot(Y,P2), C2*pot(X,P3)*pot(Y,P4)]
            | C1*W^P1*Y^P2+C2*W^P3*Y^P4-C ueq 0.

solved(X) \ C1*X^P1*Y^P2+C2*X^P3*Y^P4 deq C <=>
                                    L1 is C1*X**P1, L2 is C2*X**P3,
                                    L1*Y^P2+L2*Y^P4-C ueq 0.
```

While the standard notation is easier for users to understand, it does not give full access to the components of the equation, which is why the `pot/4` (potency) constraints were introduced. Each `pot(X,E,T,P)` constraint comprises a variable `X`, its power `P` and the equation and the term it originates from- `E` and `T` respectively. A term consists of a constant and two or three `pot` constraints, and an equation is represented as the following list of constraints `[A*pot(X,E,1,P1)*pot(Y,E,1,P2),B*pot(X,E,2,P3)*pot(Y,E,2,P4)] eq C`, mathematically equivalent to $A{*}X^{P1}{*}Y^{P2}{+}B{*}X^{P3}{*}Y^{P4} = C$. At the beginning of the solver's run, the input equations are transformed into the `pot` notation and the `pot` constraints are added to the constraint store. `pot` constraints give a global insight wether there are univariate equations or singly occuring variables by cross-referencing powers of variables and the term and equation they are in, as each variable is directly accessible through its `pot` constraint. For example having two pot constraints `A*pot(X,1,1,1)` and `pot(Y,1,1,0)` means that Y occurs at most once in equation one, as the first term does not contain it. To decide wether it stands alone or is embedded in a term, the second term needs to be checked. The `pot` constraints are also used for realizing the substitutions in 3.2. For readability, the term and equation identifiers will be removed for the code samples demonstrated here.

```
[C1*pot(X,0)*pot(Y,P1),C2*pot(X,0)*pot(Y,P2)] eq C,
 pot(X,0),pot(X,0) <=> C1*Y^P1+C2*Y^P2-C ueq 0.

[C1*pot(X,0)*pot(Y,P2), C2*pot(X,P3)*pot(Y,P4)] eq C <=>
                              X req ((C-C1*Y^P2)/(C2*Y^P4))^(1/P3).
```

3.2 Substitution Scheme

The solvable subset is extended by introducing a substitution scheme that simplifies the initial problem set to one of the solvable states. If there is no variable x to solve for, without turning the function $f_i : A_i * T_j + B_i * T_j = C_i$, $i \in \{1, 2\}$, $j \in \{1, 2, 3, 4\}$ where $T_j = x^{P1} * y^{P2}$, into a more complicated one, then some substitutions are applied to the equation terms until one of the solution cases is applicable.

The substitution function $s : F^2 \times F^2 \rightarrow G^3 \times G^3$ assigns each bivariate term to an equivalent trivariate one, by introducing a substitution variable a. Substituting all terms within the initial system of equations $(f_1(x, y) = 0, f_2(x, y) = 0)$ results in a trivariate equation set $(g_1(x, y, a) = 0, g_2(x, y, a) = 0)$, as given by the function s. The function s is represented by the `sub/2` constraint, where the first attribute is the original bivariate term and the second the trivariate

substitution term. The input equation set is transformed into the almost identical output set g_i with $T_j = a^{P1} * x^{P2} * y^{P3}$. The chosen substitution scheme describes system terms in terms of each other while adding only one additional variable to form the base substitution, to be eliminated later. This is efficient, as it excludes the possibility of producing numerous new variables, that cause the system of equations to be strongly under-defined. For the creation of the substitution keys, namely the `sub` constraints, there are four different substitution cases, which should be checked in the order given below:

1. Direct substitution is applicable if there are two identical terms, except for their respective coefficients: $\exists i : T_j = T_i | i \neq j,\ i, j \in \{1, 2, 3, 4\}$.

```
sub(pot(X,P1)*pot(Y,P2),pot(Var,1)) \ pot(X,P1), pot(Y,P2) <=>
                  P1\=0, P2\=0 | pot(Var,1), pot(X,0), pot(Y,0),
                                 sub(pot(X,P1)*pot(Y,P2), pot(Var,1)).
```

2. Multiples substitution is utilized if there exists a term that is the multiple of another: $\exists n \in \mathbb{Z} : T_i^n = T_j, i \neq j$.

```
pot(X,P1),pot(Y,P2),pot(X,P3),pot(Y,P4) <=>
                      P1\=0,P2\=0,P3\=0,P4\=0,divides(P3,P1,Q1),
                      divides(P4,P2,Q2),Q1 == Q2
                    | pot(Var,1),pot(X,0),pot(Y,0),pot(Var,Q1),pot(X,0),
                      pot(Y,0),sub(pot(Y,P2)*pot(X,P1),pot(Var,1)),
                      sub(pot(Y,P4)*pot(X,P3),pot(Var,Q1)).
```

3. Product substitution can be applied, if one term can be expressed as the product of two other terms: $\exists i : \exists j : T_k = T_j * T_i \wedge i \neq j$.

```
sub(pot(X,P3)*pot(Y,P4),pot(Var1,N1)*pot(Y,N2)*pot(X,N3)),
 sub(pot(X,P5)*pot(Y,P6),pot(Var2,N4)*pot(Y,N5)*pot(X,N6)) \
  pot(X,P1),pot(Y,P2) <=> Q1 is P3+P5,P1==Q1,Q2 is P4+P6,
  P2==Q2,F1 is N1+N4,F2 is N2+N5,F3 is N3+N6
| sub(pot(X,P1)*pot(Y,P2),pot(Var1,F1)*pot(Y,F2)*pot(X,F3)),
  pot(Var1,F1), pot(Y,F2),pot(X,F3).
```

4. If none of the above cases apply, the only unsubstituted terms remaining will be those that could be expressed as the multiplication of a term with one of the system variables: $\exists i : T_j = T_i * v, v \in \{x, y\}$.

```
sub(pot(X,P1)*pot(Y,P4), pot(Var,N)) \
 pot(X,P1),pot(Y,P2) <=> P1\=0, P2\=0,P2 > P4, Diff is P2-P4
                      | pot(Var,E1,T1,N),pot(X,E1,T1,0),pot(Y,E1,T1,Diff),
                        sub(pot(Y,P2)*pot(X,P1),pot(Var,N)*pot(Y,Diff)).
```

Even though the equation set is three dimensional, the trivariate equation set will then match one of the solvable cases, as each equation on its own is still bivariate, which is ensured by the chosen substitution scheme. This matching has to happen as one whole term must have been taken as a reference point by s and fully substituted by the substitution variable a. After the `sub` constraints are created, all equations are iterated over and the actual substitution is done. After one of the variables has been solved, the equation $A = X^{P1} * Y^{P2}$ from the base substitution `sub(pot(X,P1)*pot(Y,P2), pot(A,1)*pot(X,0)*pot(Y,0))` is added, to enable the solving of the remaining variables.

3.3 Evaluation

This solver computes solutions to univariate polynomial equations of degrees up to four and binomial bivariate equation sets without approximations and relying only on substitution and direct methods. While various solvers, especially computer algebra systems, for the chosen subset exist, no tools were found that could solve binomial bivariate equations without adding approximative methods, especially none using CHR.

Most of said computer algebra systems, in particular Mathematica [26, 27] and Maple [28, 29], solve the whole set covered by the implemented CHR solver and a vast amount of other mathematical problems, relying primarily on symbolic evalutions of equations and approximative, iterative methods like the Newton method for calculating numeric results. Although the renowned computer algebra tools cover a much wider solution set, the CHR solver is capable of giving the numerical solutions of some systems of equations directly through substitutions, while Mathematica or Maple for example would have yielded to approximative methods instead. In case no solutions can be calculated, both the CHR solver and computer algebra tools, symbolically simplify the equation set the farthest possible.

The majority of the existing solvers for univariate polynomial equations and computer algebra results, display all complex results, whereas the implemented solver only gives the real results of a system of equations, as Prolog is currently only defined in the domain of real numbers.

The method used in this solver is straightforward and thus does not have a high complexity. Depending on the input set, one or at most two full iterations are done and thus the results are achieved almost directly by firing the correct rules. CHR enables the direct translation of the human-based solution method into a program which facilitates the solving of the whole subset, which is not a commonly used method in other non-linear equation solving tools The addition of the substitution system renders otherwise non-solvable systems of equations solvable without adding much complexity, thus extending the solvable subset.

4 Conclusion and Future Work

4.1 Conclusion

Deriving new ways to solve non-linear algebraic systems of equations or improving existing ones is the concern of many fields in mathematical computing. Most of these solution systems are based on approximation methods. This work aimed at finding an exactly solvable, tractable subset of non-linear equation sets, deriving a method to solve said subset and realizing this method using CHR. After investigating different pre-existing solving mechanisms and the non-linear subsets they solve, it was decided to constrict the solution field to bivariate polynomial systems of equations. A substitution-based method for solving bivariate equation sets was derived. The algorithm reduces one of the system equations

into a solvable univariate one and then uses the solution to reduce the second one. As proof of concept, the method was modeled for binomial equation sets, as any extensions to multinomial sets, would follow analogously. The implemented solver extends the solvability of the chosen subset through substitutions, without resorting to approximative methods. The solver was implemented using K.U.Leuven's CHR implementation with Prolog as the host language. The roots for any consistent system of equations are obtained, given that the resulting univariate equations are standardizable and quantifiable through finite formulae. Otherwise the highest possible simplification is attained.

4.2 Future Work

There is a large scope of extensions for this solver, depending on the needed functionalities, as this solver was intended to prove a concept based on a subset from which multidirectional expansions are possible. The solver could be extended to solve all consistent univariate systems of polynomial equations. Furthermore, the scope of this work could be broadened to cover multinomial bivariate and overdefined non-linear systems. Finally, the same concept of the solver could be a basis for solving more complex types of non-linear equation sets, such as trivariate polynomial equation sets or bivariate non-linear non-polynomial equation sets, e.g. trigonometric functions for which transformative substitutions exist.

References

1. Online function plotter. http://rechneronline.de/function-graphs/,June 2012.
2. Pascal Triangle. http://matheuropa.lfs-koeln.de/pascal/binformel.htm, June 2012.
3. Online diagram drawer. http://www.lucidchart.com, June 2012.
4. J. Xue, Y. Li, Y. Feng, and Z. Liu, An intelligent hybrid algorithm for solving nonlinear polynomial systems., in International Conference on Computational Science (M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, eds.), vol. 3037 of Lecture Notes in Computer Science, pp. 2633, Springer, 2004.
5. P. Van Hentenryck, D. McAllester, and D. Kapur, Solving polynomial systems using a branch and prune approach, SIAM Journal of Numerical Analisys, vol. 34, pp. 797827, April 1997.
6. L. D. Koninck, T. Schrijvers, and B. Demoen, Inclp(r) - interval-based nonlinear constraint logic programming over the reals, in Workshop on Logic Programming, pp. 91100, 2006.
7. T. Frühwirth, Constraint handling rules. Cambridge University Press, 2009.
8. K. I. Joy, Bernstein polynomials, in On-Line Geometric Modeling Notes, 1996.
9. F. Benhamou, D. A. McAllester, and P. V. Hentenryck, Clp(intervals) revisited, in SLP, pp. 124138, 1994.
10. H. Collavizza, F. Delobel, and M. Rueher, A note on partial consistencies over continuous domains, in In Proc. CP98 (Fourth International Conference on Principles and Practice of Constraint Programming, pp. 147161, Springer Verlag, 1998.
11. L. Granvilliers, A symbolic-numerical branch and prune algorithm for solving nonlinear polynomial systems, Journal of Universal Computer Science, vol. 4, pp. 125 146, February 1998.

12. D. G. Sotiropoulos and T. N. Grapsa, Optimal centers in branch-and-prune algorithms for univariate global optimization, Applied Mathematics and Computation, vol. 169, no. 1, pp. 247277, 2005.
13. V. Stahl, Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations. PhD thesis, Forschungsinstitut fur Symbolisches Rech- nen, Technisch-Naturwissenschaftliche Fakultat, Johannes Kepler Universtat Linz, 1995.
14. S. Herbort and D. Ratz, Improving the efficiency of a nonlinear-system-solver using a componentwise newton method, in Berichte aus dem Forschungsschwerpunkt Computerarithmetik, Intervallrechnung und Numerische Algorithmen mit Ergebnisverifikation, 1997.
15. Online quartic calculator. http://www.1728.org/quartic2.htm, June 2012.
16. J. Verschelde, Homotopy Continuation Methods For Solving Polynomial Systems. PhD thesis, K.U.Leuven, Belgium, 1996.
17. X.-W. CHANG, Solving a nonlinear equation. Lecture Slides, COMP 350 Numerical Computing, McGill School of Computer Science, 2011.
18. P. D. D. Nesselmann, Grobner-basen und nicht-lineare gleichungssysteme, in Manuskript zur Vorlesung, pp. 8798, Universitat Rostock, 2009.
19. E. C. Sherbrooke and N. M. Patrikalakis, Computation of the solutions of nonlinear polynomial systems Computer Aided Geometry Design, vol. 10, pp. 379–405, October 1993.
20. B. Mourrain and J.-P. Pavone, Subdivision methods for solving polynomial equations, Journal of Symbolic Computation, vol. 44, no. 3, pp. 292–306, 2009.
21. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes 3rd Edition: The Art of Scientific Computing. New York, NY, USA: Cambridge University Press, 2007.
22. M. I. Rosen, Niels Hendrik Abel and Equations of Fifth Degree, American Mathematical Monthly, Vol. 102, 1995, pp. 495 505.
23. R. J. Drociuk, On the Complete Solution to the Most General Fifth Degree Polynomial, 3-May 2000, http://arxiv.org/abs/ math.GM/0005026/.
24. Abel, U., H., Beweis der Unmoglichkeit algebraische Gleichugen von hoheren Graden als dem vierten allgemein aufzulosen, J, fur die reine und angew. Math., Bd. 1 (1826) 65-84.
25. C. Grosan and A. Abraham, A new approach for solving nonlinear equations systems, IEEE Transactions on Systems, Man and Cybernetics Part A, 2007.
26. Equation Solving Using Mathematica., http://reference.wolfram.com/mathematica/guide/EquationSolving.html, Wolfram Research, Inc., August 2012.
27. Wolfram Mathematica Tutorial Collection: Mathematics and Algorithms. Wolfram Research,2008.
28. Maple-The Essential Math Software Tool, http://www.maplesoft.com/products/Maple/, Maplesoft, August 2012.
29. Holistic Numerical Methods- Transforming Numerical Methods Education for the STEM Undergraduate, http://numericalmethods.eng.usf.edu/mtl/gen/03nle/index.html, Autar Kaw, University of South Florida, August 2012.