Solution of Systems of Polynomial Equations By Elimination

Joel Moses

*IBM Corporation, Cambridge, Massachusetts**

The elimination procedure as described by Williams has been coded in LISP and FORMAC and used in solving systems of polynomial equations. It is found that the method is very effective in the case of small systems, where it yields all solutions without the need for initial estimates. The method, by itself, appears inappropriate, however, in the solution of large systems of equations due to the explosive growth in the intermediate equations and the hazards which arise when the coefficients are truncated. A comparison is made with difficulties found in other problems in non-numerical mathematics such as symbolic integration and simplification.

Let us consider the following problem:

Given a system of polynomial equations with real coefficients,

$$P_i(x_1, x_2, \cdots, x_n) = 0, \quad i = 1, 2, \cdots, n,$$

find the solutions (common zeros) of the system (assuming the consistency of the system).

The standard attack of numerical analysis on this problem is by an iterative method based on Newton's method or the method of steepest descent. These methods assume a good estimate for the solutions and they may, in fact, fail to converge otherwise. Such an estimate will, in general, be required for each desired solution. Obtaining the estimates can be a very time-consuming process and obtaining sufficiently good estimates may be a prohibitively costly operation.

The approach considered here is the one taken by the algebraists of the last century which led to the elimination method. The elimination method reduces the problem to that of finding the zeros of a polynomial in a single variable. It is very similar to the Gaussian elimination method used in solving linear equations.

In the Gaussian elimination procedure (Figure 1) the variable x_i is eliminated in the step labelled A. The corresponding step B for the polynomial case (Figure 2) is a call to a subroutine (Figure 3) which produces two polynomials which we shall call the *divisor* and the *eliminant*. The eliminant is independent of x_i and the divisor is usually of lower degree in x_i than the original equations. Note the similarity of the form of the transformation involved in step C of the subroutine and that in step A.

```
    for $i$ := 1 step 1 until $n-1$ do
    for $j$ := $i + 1$ step 1 until $n$ do
A     $P_j := a_{ii}P_j - a_{ji}P_i$
```

FIG. 1. Gaussian elimination for linear equations

$$P_i = \sum_{k=1}^{n} a_{ik}x_k + a_{i,n+1} = 0, \quad P_j = \sum_{k=1}^{n} a_{jk}x_k + a_{j,n+1} = 0, a_{ii}a_{ji} \neq 0$$

```
    for $i$ := 1 step 1 until $n-1$ do
    for $j$ := $i+1$ step 1 until $n$ do
B     $P_i$ := divisor of $P_i$ and $P_j$ with respect to $x_i$
      $P_j$ := eliminant of $P_i$ and $P_j$ with respect to $x_i$
```

FIG. 2. Elimination for polynomial equations

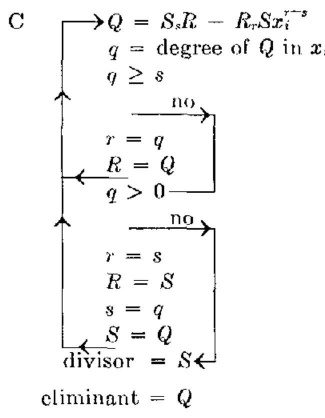

FIG. 3. Subroutine for eliminating x_i from polynomial equations R and S

$$\left.\begin{aligned} R &= \sum_{k=0}^{r} R_k(x_{i+1}, x_{i+2}, \cdots, x_n)x_i^k = 0 \\ S &= \sum_{k=0}^{s} S_k(x_{i+1}, x_{i+2}, \cdots, x_n)x_i^k = 0 \end{aligned}\right\}, R_rS_s \neq 0, \quad 0 < s \leq r$$

Presented at an ACM Symposium on Symbolic and Algebraic Manipulation, Washington, D. C., March 29–31, 1966.

* Present address: Research Laboratory of Electronics, MIT, Cambridge, Mass.

This work was supported in part by the Joint Services Electronics Program (Contract DA 36-039-03200(E)), the National Science Foundation (Grant GP-2495), and the National Aeronautics and Space Administration (Grant NsG-496).

While step A completely eliminates the variable, step C can only eliminate the highest power of the variable appearing in the two equations. In the linear case the transformation in A is best done by an algebraic processor such as FORTRAN. However, in the nonlinear case a processor with polynomial manipulation capabilities is essential in order to efficiently perform the transformation indicated in step C. We shall call the resulting system of equations, P_i, $i = 1, 2, \cdots, n$, the *reduced system.*

The following theorem [1] provides the justification for the elimination procedure used in the nonlinear case.

THEOREM. *A common zero of P_i and P_j is a common zero of the eliminant and divisor of P_i and P_j with respect to x_i.*

Hence, in order to find the solutions to the original system of equations we need find only the solutions to the reduced system. In the reduced system each equation P_i is independent of x_j if $j < i$. Typically the reduced system also has the following character: The reduced equations P_i, $i = 1, 2, \cdots, n-1$ are linear in variable x_i, and equation P_n is of high degree in x_n, where the degree reflects the number of solutions of the system. Thus P_n can be solved (in principle) by a root-finding program and the complete solutions can be obtained by successively solving $P_{n-1}, \cdots, P_2, P_1$ for $x_{n-1}, \cdots, x_2, x_1$ as equations in one unknown in a backward substitution process similar to that used in the Gaussian elimination procedure.

Although the above is the usual case there are some exceptions. For one, the first $n-1$ equations of the reduced system need not be linear. This complicates the backward substitution somewhat, but normally poses no real problem. Another possibility is that solutions of the reduced system may be extraneous. This is usually noted in the backward substitution phase, when an equation of the reduced system becomes identically zero. Such solutions are then ignored. Finally, we may find that one of the eliminants generated during the elimination process will vanish identically. This occurs when equation P_i and P_j have a factor in common involving x_i. This difficulty may sometimes be overcome by rearranging the equations and starting the process over again. Otherwise, the equations can be divided by the common factor, thereby generating subsystems of equations each of which must be solved. This difficulty is troublesome, but fortunately it does not occur often in practice.

Williams [1] described how the complete procedure could be programmed for a digital computer using a polynomial manipulation system, but he lacked access to a sufficiently large machine on which to carry out his ideas. The procedure has been programmed both in LISP [2] and FORMAC [3] for the IBM 7094. An example of the solution of a small system is given in Figure 4. The solution of another system is described in Figure 5. The program uses single precision arithmetic throughout except when finding the roots of equations. The latter process is performed in double precision with a program using Muller's method [4]. It is interesting to note that Bareiss' root-finding program [5] makes use of elimination in a limited context in extending the Graeffe root-squaring technique.

If a system has more variables than equations, we call the variables which are not eliminated *parameters* of the system. J. Israel has written a program in FORMAC which obtains solutions in closed form to systems containing parameters in which the equations in the reduced system are of degree 4 or less in their respective variables. The solutions (in terms of the parameters) are obtained from the formulas which are available in such cases.

The experience gained with the programs leads to the following conclusions.

(1) The method yields fast and accurate results for small systems of equations (e.g., three quadratics). No starting values are needed and all the solutions (complex, in general) are obtained except in degenerate cases.

(2) When the reduced system is of degree 4 or less in each of its variables, standard formulas can be used to obtain solutions in closed form for the original system in terms of its parameters.

(3) For slightly larger systems (e.g., three quartics) the degree of the final equation in the reduced system may be so large that one cannot obtain sufficiently accurate solutions for it with existing root-finding processes. With an estimate for a zero of the reduced equation P_n such a program can find any single desired solution. Backward substitution yields the complete solution. In such a case the elimination method is superior to existing numerical procedures in that it requires an estimate for only one variable instead of all.

Original system	
A	$x^2 - y^2 = 0$
B	$x^2 - x + 2y^2 - y - 1 = 0$
Intermediate equations	
$C = 1 \cdot A - 1 \cdot B$	$x - 3y^2 + y + 1 = 0$
$D = 1 \cdot B - 1 \cdot C \cdot x$	$(3y^2 - y - 2)x + 2y^2 - y - 1 = 0$
$E = (3y^2 - y - 2)\,C - 1 \cdot D$	$9y^4 - 6y^3 - 6y^2 + 2y + 1 = 0$
Reduced system	
C	$x - 3y^2 + y + 1 = 0$
E	$9y^4 - 6y^3 - 6y^2 + 2y + 1 = 0$
Solutions of E:	$y = 1, -\frac{1}{3}, \sqrt{\frac{1}{3}}, -\sqrt{\frac{1}{3}}$
Substitution in C:	$x = 1, -\frac{1}{3}, -\sqrt{\frac{1}{3}}, \sqrt{\frac{1}{3}}$

Time for solution with FORMAC program: approx. 8 sec.

FIG. 4

Original system	
A	$x^2y + 3yz - 4 = 0$
B	$-3x^2z + 2y^2 + 1 = 0$
C	$2yz^2 - z^2 - 1 = 0$
Reduced system	
D	$x^2y + 3yz - 4 = 0$
E	$2yz^2 - z^2 - 1 = 0$
F	$36z^8 - 96z^7 + 42z^6 + 10z^4 + 6z^2 + 2 = 0$

FIG. 5. The above system has 16 solutions. These solutions were obtained with the FORMAC program in 75 sec with a 7-digit accuracy.

(4) For large systems of equations, the final polynomial in the reduced system can be of such high degree that it will be virtually impossible to solve for all the roots, and the solution for a single root will face a roundoff error problem. In many situations, however, the large final polynomial can be avoided by overdetermining the system, i.e., by adding more equations without any additional unknowns. A nontrivially overdetermined system usually has far fewer solutions than the original system. The price for obtaining a final equation of small degree is the added size of the intermediate polynomials and the possible increased error due to roundoff.

We find that there are two types of limitations to the computer implementation of solutions to problems in non-numerical mathematics. One is due to recursive unsolvability, the other is due to the explosive growth in the intermediate variables in recursive problems. It is a familiar fact that deciding the theorems in the lower predicate calculus is a recursively unsolvable process and hence necessitates the use of heuristics in the general case. Richardson [6] has shown that the problem of matching elementary functions [7] is also recursively unsolvable. The elementary functions contain the usual arithmetical, trigonometric, exponential and logarithmic functions of real variables, and the matching problem asks whether an elementary function is equivalent to zero.

It is interesting to note that the difficulties found in matching programs such as Martin's [8], and Tobey's [3] occur with the introduction of the same functions (log and square root) that are critical in Richardson's proof.

Simplification has properties which make it, in general, an even more insidious problem than matching. The criteria for simplicity are not fixed but may vary with the situation. For instance, it may be easier to integrate $4x^3[x^4/((x^4)^3+1)]$ than the equivalent and apparently simpler $4x^7/(x^{12}+1)$. Suppose that the criteria could be fixed (e.g., zero is the simplest form of a function equivalent to it), then simplification behaves like a minimization problem in a none-too-well structured space. It is difficult to tell whether one has the minimum (i.e., simplest) expression, or what transformation to apply to get down to the minimum. In fact, the minimum cannot always be recognized in a finite number of steps because by finding it we could solve the matching problem.

Richardson's unsolvability result for matching also carries over to the integration problem for the elementary functions. It is hard to say how much Slagle's integration program [7] suffers from this difficulty, but the result certainly explains the difficulties mathematicians have had in solving problems in integration and differential equations.

The limitations of procedures designed to perform manipulations (e.g., inversion) on matrices with nonconstant entries are certainly not due to recursive unsolvability. The difficulties in these problems are due to the highly nonlinear growth of the intermediate and final results, in general. Similarly, the difficulties we are faced with in the solution of systems of polynomial equations are due to the explosive growth in the degree of intermediate equations and their coefficients in large systems.

Further remarks about the difficulties due to the coefficients are in order. If the coefficients of the original system of equations are not rational (hence introducing a starting error in the computer realization of the solution), then the instability of the problem will lead to gross errors in the solution of large systems. Suppose we truncated the coefficients in some stage of the solution in order to gain space and time, then the solution for sufficiently large systems will be in great error due to the roundoff error introduced coupled with the instability of the problem. Hence in order to obtain meaningful solutions in the elimination of large systems, the coefficients should be exact at all stages. In Williams' paper these difficulties seem to have been underestimated.

Recently Collins [8] obtained some major results in this area. He derived a small bound on the growth of the length of the coefficients in the problem of finding the greatest common divisor (GCD) of two polynomials. Usually GCD algorithms are essentially extensions of the Euclidean algorithm to the polynomial case, and are similar to the subroutine in Figure 3. Collins shows that the growth of the coefficients is linear in the degree of the original equations and the length of their coefficients, and is not nearly as large as was previously estimated. He furthermore has programmed a method for obtaining a reduced system to a system of polynomial equations which damps the growth of the degree of the intermediate equations and their coefficients. We believe these results constitute a breakthrough in the solution of this problem by the techniques of algebraic manipulation. However, we doubt that any single procedure can change the exponential character of the problem. One should keep in mind that the number of solutions and their absolute values are, in general, exponential functions of the coefficients, degrees and number of equations of the original system. While ideally one would like the degrees of each of the polynomials of the reduced system to be as small as possible (the danger then being the accumulation of errors during the backward substitution, forcing the solution of polynomial equations with inexact coefficients), it seems unlikely that one can avoid, in general, simulating a large part of the search process necessary to locate the solutions in an n-dimensional complex space. In numerical analysis this search is explicit; the implicit methods in algebraic manipulation tend to lead to increasingly large intermediate variables.[1] We venture a guess that Kutta's system [10] of 16 equations in 15 unknowns (used in constructing a fifth-order

[1] These are systems of equations whose reduced system must contain exponentially large polynomials. Consider the system $x_i = x_{i+1}^2$, $i = 1, 2, \cdots, n-1$, $2x_n = x_1^2$. The reduced system must contain a polynomial divisible by one of the polynomials $x_k^{2^n-1} - 2^{2^{n-k}}$, $k = 1, 2, \cdots, n$

Runge-Kutta method), which has, to the best of our knowledge, withstood solution by the techniques of numerical analysis, is going to remain unsolved by the techniques of algebraic manipulation.

Although there exists for the above two types of problems in algebraic manipulation no single program which will handle all possible inputs efficiently or at all, there are many special categories where such programs can and do exist. Even though simplification in general is not possible, the simplification of rational functions reduces to the GCD problem which is recursive. The integration of rational functions in closed form is performed by a program written by Manove [11]. Surprisingly, the integration problem for the algebraic functions (these include rational functions and roots of rational functions) appears to be recursive also. Hardy [12] conjectured that this problem could not be solved in a finite number of steps.

For the recursive problems which experience growth of the intermediate variables there exist similar categories. One category is composed of those instances which are so simple that reasonable algorithms do not manage to blow them up. The simplicity of the algorithm to be used in solving more difficult cases need not be a virtue. For example, in elimination one profits from factoring the equations because this inhibits their growth. This is the manner in which a high school student would proceed on the system in Figure 4. The ALPAK algorithm for the GCD problem [13] makes use of special cases of factoring. A more general factoring algorithm is implemented in Engleman's system [11]. Modifications can also be made to the basic elimination algorithm to prevent the introduction of any extraneous roots. In this connection it should be noted that Williams did not account for all the extraneous roots introduced by the elimination procedure.

Finally, each particular instance of a problem can be considered as a special category since the user may have some insight into it. Symbolic manipulation systems should have sufficient flexibility for the user to translate his insight into an algorithm simply. Since a user frequently cannot foresee the possible consequences of a proposed transformation, a system with capabilities for man/machine interaction has many advantages. Martin's display program for mathematical expressions [14] is a large step in this direction. We believe that difficult problems can be most efficiently solved, if they can be solved at all, on systems possessing such capabilities.

REFERENCES

1. Williams, L. H. Algebra of polynomials in several variables for a digital computer. *J. ACM 9* (Jan. 1962), 29–40.
2. LISP 1.5 programmer's manual. Res. Lab. of Electronics, MIT, Cambridge, Mass., 1962.
3. Bond, E. R. et al. FORMAC—An experimental formula manipulation compiler. Proc. ACM 19th Nat. Conf., Philadelphia, 1964, Paper K2.1-1.
4. Muller, D. A method for solving algebraic equations using a digital computer. *Math. Tables Other Aids Comput. 10* (1956), 208–215.
5. Bareiss, E. H. Resultant procedure and the mechanization of the Graeffe process. *J. ACM 7* (Oct. 1960), 346–386.
6. Richardson, D. Doc. Thesis, U. of Bristol, Bristol, England.
7. Slagle, J. R. A heuristic program that solves symbolic integration problems. Doc. Thesis, MIT, Cambridge, Mass., 1961.
8. Martin, W. A. Hash-coding functions of a complex variable. Artificial Intelligence Project Memo 70, MIT, Cambridge, Mass., 1964.
9. Collins, G. PM, a system for polynomial manipulation. *J. ACM 9* (Aug. 1966), 578–589.
10. Kutta, W. Beitrag zum näherungsweisen Integration totaler Differentialgleichungen. *Z. Math. Phys. 46* (1901), 435–453.
11. Engleman, C. Mathlab: a program for on-line machine assistance in symbolic computation. Rept. MTP-18, MITRE Corp., Bedford, Mass., 1965.
12. Hardy, G. H. *The Integration of Functions of a Single Variable.* 2nd. ed., Cambridge U. Press, Cambridge, England, 1916.
13. Brown, W. S., Hyde, J. P., and Tague, B. A. The ALPAK system for nonnumerical algebra on a digital computer-II. *Bell Sys. Tech. J. 43* (March 1964), 785–804.
14. Martin, W. A. Syntax and display for mathematical expressions. Artificial Intelligence Project Memo 85, MIT, Cambridge, Mass., 1965.

Informal Evening Sessions

One of the features of the Symposium was a procedure copied from SHARE, in which informal evening discussions were suggested and led by attendees themselves. Sessions were held on the following topics:

- Implementation Techniques
- Approximations to Solutions of Differential Equations
- Mathematics and Metamathematics of Algebraic Manipulation Languages
- LISP II
- Natural Languages Processing
- Symbol Manipulation Techniques As Applied to Graphical Languages and Topological Models
- FORMAC 360 External Specifications
- L^6 — A Computer-Made Movie Showing a Programming Example in L^6

It is worth noting that the LISP II meeting was attended by over 50 people, and the session on Metamathematics by a vigorous group of about 65 people, even though the time was 6–8 p.m.—J. E. S.