Solving finite algebraic systems using numeric Gröbner bases and eigenvalues

Daniel Lichtblau

Wolfram Research, Inc.
100 Trade Centre Dr.
Champaign, IL 61820 USA
danl@wolfram.com

In: Proceedings of the World Conference on Systemics, Cybernetics, and Informatics (SCI2000). Volume **10** (Concepts and Applications of Systemics, Cybernetics, and Informatics). M. Torres, J. Molero, Y. Kurihara, and A. David, eds. 555−560. International Institute of Informatics and Systems. July 2000.

Abstract

Systems of algebraic equations with finitely many solutions arise in many areas of applied mathematics. Among these are motion planning, robotics, computer−aided design, and graphics. We will discuss the design and implementation of a hybrid symbolic−numeric method, and a *Mathematica* implementation thereof, that finds all solutions to an algebraic system. It makes use of numeric Gröbner bases and arbitrary−precision numeric eigenvalue computation. We explain in outline how this works, and give a few examples that demonstrate how this can be useful technology independent of Newton's method local solvers.

Keywords: Gröbner basis, eigenvalues, hybrid symbolic−numeric solving, algebraic equations

Introduction

The need to obtain many or all solutions to a system of algebraic equations is becomming ever more apparent. Problems in computer−aided design, graphics, the physical sciences, biochemistry, robotics, and elsewhere often lead to nonlinear systems of equations with finitely many solutions. Over the past dozen years or so several hybrid symbolic/numeric approaches have appeared. In this paper we discuss one that the author has implemented in version 4 of *Mathematica*.

Suppose we are given a system of polynomial equations, for which the solution set is finite, and we wish to find good numeric approximations to all solutions. We might try local methods such as Newton's and relatives, giving different starting points, in the hope of coming across all solutions in this manner. There are several problems with such an approach. First, unless we know a priori how many roots there are, it is virtually impossible to know whether all have been found (using a Bezout bound can help, but in typical situations it is only an upper bound and one will not know from this how many finite solutions there are). Moreover getting good starting points is difficult.

Another alternative is to use an exact solver. These typically use technology that involves the computation of something called a Gröbner basis from the polynomials [Buchberger 1985; Cox, Little, and O'Shea 1992; Becker, Weispfenning, and Kredel 1993]. The simplest approach computes such a basis for what is called a lexicographic term order. One obtains a new set of polynomials generating the same ideal (and hence having the same solutions with the same multiplicities) but in "triangulated" form. There is a univariate polynomial in the "last" variable, for which one may obtain solutions, and these may be plugged into the next polyno−mial, which involves only the last two variables. One now can solve for the next−to−last variable, plug in values, etc. A difficulty with this approach is that computation of such bases in general, and with respect to such term orderings in particular, can be very expensive. One uses exact integer arithmetic and this can suffer from coefficient swell. A partial remedy involves a technique known as basis conversion [Faugére, Gianni, Lazard, and Mora 1993]. In short, one computes with respect to a typically less expensive term order and later converts to the desired order using linear algebra. The conversion can still be a bit time consuming, at least in high dimensions, and moreover even the cheaper basis construction can be expensive when using exact arithmetic. Moreover exact root extraction and back−substitution steps can be computationally expensive.

There are two specific improvements that make this general idea tractable as a hybrid symbolic−numeric method. The first is to compute the basis using finite precision arithmetic. This in effect eliminates the intermediate growth at the expense of introducing the potential for failure (this will be discussed later). The second modification is to use a numeric eigensystem method to extract roots of the system directly from this basis, without recourse to basis conversion; this tactic was introduced, in a slightly different setting, in [Auzinger and Stetter 1988] with similar ideas also appearing in [Yokoyama, Noro, and Takeshima 1992], [Cellini, Gianni, Traverso 1991] and perhaps other places. We will discuss the technique that emerged, and some details of its implementa−tion in the *Mathematica* kernel [Wolfram 1999] (*Mathematica* (TM) is a registered trademark of Wolfram Research, Incorporated).

Background

Gröbner bases are a tool used universally in computational commutative algebra. Among several excellent references we single out [Buchberger 1985], [Cox, Little, and O'Shea 1992], and [Cox 1998] because they cover various aspects of equation solving via Gröbner bases.

We give a brief synopsis. One defines a term order on the exponent vectors of power products of a polynomial (these are simply products of powers of variables, e.g. $x^2\, y\, z^3$). Using an algorithm developed by Buchberger (as in the references above) one then rewrites the given set of polynomials to obtain a Gröbner basis. This new set generates the same polynomial ideal and hence has the same solution set. If computed with respect to a lexicographic term ordering, one has in effect triangulated the system, in a form analogous to a row reduced system of linear equations. If the system has finitely many solutions, and if certain generic technical assumptions are fulfilled (as specified in [Gianni and Mora 1988]), one will obtain a univariate polynomial in the lowest ordered variable, and polynomials that relate each of the remaining variables linearly to that lowest ; if the generic assumptions are not fulfilled, one still has something triangular but without linear polynomials in remaining variables. Extracting solutions is then a matter of solving that univariate and back−substituting each solution into the remaining polynomials. A progenitor of this method was apparently first discovered by Trinks in the late 1970's (see [Buchberger 1985]) although several niceties, such as obtaining linear polynomials in the generic case, were not uncovered until later.

A problem with this technique is that computation of Gröbner bases with respect to lexicographic term orders is typically strenu− ous. Generally it is much faster to compute with respect to a term order that is graded by total degree of monomials (that is, one first compares degree and only if equal does one look, say, at lexicographic order of the power products). So now one must somehow use a basis with respect to such an order to get the roots of the system. One method, outlined in [Gianni and Mora 1988] and refined in [Faugére, Gianni, Lazard, and Mora 1993], involves basis conversion. This uses linear algebra to solve a system of equations. There is another method, now well known, that was developed in [Auzinger and Stetter 1988] (similar ideas appeared in [Manocha and Canny 1991]): using any Gröbner basis or by other means, one sets up and extracts eigenvalues from a particular matrix. Under certain circumstances a combination of the two approaches can be useful.

As noted above, the initial step of finding a Gröbner basis can be faster if one works with finite precision. This is difficult because of a "zero recognition problem" wherein values that need to be set to zero do not exactly cancel. The first attempt to handle this, using fixed precision arithmetic, was discussed in [Shirayanagi 1993]. This appeared at about the time that `GroebnerBasis` was being rewritten for version 3 of *Mathematica*. As it happens, the engine for arbitrary finite precision arithmetic in *Mathematica* utilizes what is called "significance arithmetic" [Keiper 1992; Sofroniou 1998; Sofroniou and Spaletta 2000]. Rather than do arithmetic in fixed precision, it keeps an estimate of error and adjusts precision during the course of operations. Hence at the end one knows that certain digits are valid. The ideas behind this have been around for some decades, but pessimistic error estimates rendered it little more than a curiousity. More recently it became understood how to control these esimates, and beginning around 1987 the late Jerry Keiper implemented this model of arithmetic in the *Mathematica* kernel; this may have been the first serious large−scale implementation thereof.

Using significance arithmetic the problem of recognizing zero is now reversed from the situation one faces with fixed precision. Rather than an inability to obtain zero due to round−off error, one now faces the hazard of deciding that a result is "zero" due to catastrophic cancellation that eradicated all "good" bits. This will be discussed in more detail later. The fact is that it works quite well in numerical Gröbner basis computations. The *Mathematica* implementation was coded in late 1993 and discussed in [Lichtblau 1996]; at that time the numeric Gröbner basis interface was crude, allowing one to use only 100 digits of initial preci− sion. This has since been extended to allow a user specification, with 100 digits being the default. When working over inexact numbers one either obtains a Gröbner basis or else the computation fails due to too much loss of precision along the way, in which case an error is issued. Hence the procedure is safe.

The numeric solver

The idea is to put together all of the speed improvements at our disposal. We work with numeric rather than exact Gröbner bases. We compute with respect to a degree based order because it is faster than lexicographic; an added advantage is that, because it does less work, one tends to lose less precision along the way. We then need an inexpensive way to use the resulting Gröbner basis.

This is provided in [Auzinger and Stetter 1988]. Choose a particular variable in the polynomial system, call it x. We refer to the ideal generated by our polynomials as $\langle I \rangle$. The assumption of a finite solution set implies that the polynomial alge− bra $\mathbb{C}[x, y, \ldots] / \langle I \rangle$ is a finite vector space [Buchberger 1985; Cox, Little, and O'Shea 1992]. One uses the basis to set up an endomorphism matrix that represents multiplication by x (with respect to this basis); a nice description of how one might do this may be found in [Corless 1996]. The idea is to use polynomial reduction of various power products with respect to the Gröbner basis just constructed, in order to rewrite products of a monomial basis set (for the polynomial algebra) with the variable x.

As discussed in [Auzinger and Stetter 1988], the eigenvalues of that matrix are exactly the roots of the polynomial system in that variable. We outline a proof. First one shows that for a generic ideal (radical, and in general position with respect to x), and using a lexicographic Gröbner basis, the endomorphism matrix is simply a companion matrix as with univariate polynomial quotient algebras. Linear algebra texts show that the eigenvalues give the roots of that polynomial. If the ideal did not satisfy the require− ments of genericity, one may perturb polynomial coefficients infinitesmally so that it does. This works because, loosely speaking, the set of polynomial coefficients that give rise to finite solution sets is Zariski−open, as are the configurations that give generic (by the above requirements) ideals. Moreover since roots are continuous in the coefficients of the system, one now can apply a limit argument to handle the nongeneric situation. Last, to see why we do not actually require a lexicographic term ordering for our Gröbner basis, simply note that eigenvalues of the endomorphism are independent of its matrix representation. We remark that, not surprisingly, these endomorphism matrices often go by the term "generalized companion matrices".

We now have extracted roots of the system with respect to one variable. At this point one might proceed in any of several ways. For one, we might plug in each value and repeat the process of finding an inexact Gröbner basis, etc. While seemingly indirect, in practice this can work reasonably well. Generally subsequent basis computations are trivial or nearly so because generally the ideal structure is such that the back−substitution phase is simple. In our experience it is quite rare to get endomorhism matrices that are not `1 x 1` after having substituted for two variables. Note that such substitutions would be extrememly problematic when working in fixed precision arithmetic, because typically the basis, after substitution of a value for `x`, will have more polynomials than indeterminates and hence will be overdetermined.

Another idea is to use the fact that in most cases we have linear polynomials in all remaining variables. So we might instead try to find these using the basis already computed. This method works quite well exactly when we know these linear polynomials are in the ideal, and this in turn happens exactly when there are no multiple roots in `x`. The technique for computing these polynomials is a special case of the conversion algorithm of [Faugére, Gianni, Lazard, and Mora 1993].

Yet another way to get solutions in the remaining coordinates involves computation of eigenvectors associated to each eigenvalue. It is effective for finding those solutions of multiplicity one. We will not discuss the details; a good exposition may be found in [Cox 1998, section 6].

At present the technique used by `NSolve` is to find linear polynomials when there are no multiple solutions, and to use back−substitution otherwise. It may be better to use eigenvectors but this would seem to be costly given that many real−life problems are not "generic" and so multiple solutions are not that uncommon; in such situations the vector computations would be costly and many or all would not be useful. Moreover there is a mechanism whereby one might discard unwanted roots (say, if one is only interested in those with real values), and hence one will not want to compute all eigenvectors only to ultimately use but a few of them. When back−substitution is necessary, experience indicates that each new subsystem so spawned will be much simpler than the original problem and hence the computational effort is not increased by much.

Generalizations

Thus far we have assumed that we have a system of polynomial equations in some set of unknowns. We also assume that the solution set is finite. If it is not, the method will make this determination, hence this need not be known in advance. Note however, that within *Mathematica* the fallback is to defer to older code which will very likely not produce a result in reasonable time; it is not clear how to handle the case of a numerical solution set of nontrivial dimension. Note also that, unlike the case for typical Newton’s method solvers, we need not assume the system has the same number of equations as indeterminates.

There are some other modest extensions we can handle. For one, suppose that we have rational functions in the input. Then for each denominator we make an auxiliary equation in an auxiliary variable, that forces it not to vanish (something like `den * reciprocal == 1`), and we use a variable order that eliminates the new variable.

We can similarly treat radicals, by adding new variables/equations to include the polynomial dependencies e.g. $\texttt{newvar}^2 == x + 1$ to handle input that contains $\sqrt{x+1}$, and again we eliminate the new variable. In this situation one might get what are called "parasite" solutions, hence we must verify the results numerically and discard bad ones. Thus we have extended the basic system to handle radicals and reciprocals.

Moreover it is easy to handle elimination of specified variables, as is done in the *Mathematica* `Solve` function. This is useful for finding finite intersections of objects that are given parametrically, a situation that may arise, for example, in computer−aided design. In all cases where elimination is done, a term order is used that is as close as possible to the degree based order that would otherwise be used; this tends to keep down the cost of the Gröbner basis computation.

Another generalization of the technology is to work in infinite precision, in the case where input is exact. Indeed this hardly qualifies as an extension because it works the same as the numeric algorithm, except one computes a Gröbner basis, eigenvalues, etc. using exact arithmetic. A drawback is that it frequently produces an undersimplified result, insofar as combinations of univari−ate algebraic roots can often be reduced to fairly simple form. Experience confirms that this will sometimes give results more complicated than those produced by `Solve`, but it is often significantly faster.

Often one is interested not in all roots but only a subset, say those that are real−valued. Clearly one could first find them all and a posteriori exclude those not wanted, but this tends to be inefficient. Within `NSolve` there is a limited ability to restrict the set of desired roots and, in so doing, save on computation. To explain how this is used, recall that roots are generated variable by variable. Say we only want real roots. Then we can, after solving for all roots in the last variable, select only the real ones to propagate in subsequent steps. Thus, while the initial `GroebnerBasis` and `Eigenvalues` computations are not made faster, the remaining work will often be significantly cheaper (particularly in cases where back−substitution methods are used).

One might moreover want restrictions to apply only to particular variables. For example, one may be looking for curve intersec−tions that lie in the positive quadrant. if the curves are given parametrically and one is eliminating the parameters, clearly the positivity requirements should be applied only to the variables of interest and not to the parameters. Hence the selection capability passes both a variable and a value to a user−specified function. Specifically, one may give `NSolve` an option of the form

```
SelectCriterion → (♯[[1]] ≠ x || ♯[[1]] ≠ y || (Head[♯[[2]]] === Real && ♯[[2]] >= 0) &)
```

to check that solutions in `x` and `y` are nonnegative. Simple examples of `SelectCriterion` usage are presented later.

Technical issues

One problem that may arise is in detection of "multiplicity" in the solution set. Many real−world problems either have multiple solutions or solutions wherein some variables take on the same value more than once. In the latter case it is generally best to perform a linear change of variables to put the solution set into general position with respect to a new variable; specifically, we want no repeated values in that variable unless there is actuall multiplicity in the solution set.

A useful way to proceed is to take a random linear combination of the variables, find its endomorphism matrix, extract eigenvalues to obtain solutions for that combination, and use them to solve for the actual variables. (Note that we need not compute a new Gröbner basis with respect to the changed coordinate system; this is important because such changes often destroy sparsity which in turn makes for a more laborious computation). This process is not as yet implemented in the `NSolve` code but will be in a future version. It is the only reliable way to determine whether there is actual multiplicity or just a particular variable with repeated (or close by) solutions, and moreover in the latter case it allows one to use the linear polynomial rather than back−substitution method to find roots in remaining variables.

A useful tactic to use in the poresence of near−multiplicity is to detect when several values are sufficiently "close" (this invariably involves some heuristics, as eigencomputations at multiplicities have numeric problems). In this case a useful strategy is to average the values of each cluster [Corless, Gianni, and Trager 1997].

Using significance arithmetic we are unlikely to mistake (what should be) a zero value for something nonzero. Unfortunately the opposite may happen. That is, catastrophic cancellation of significant digits may cause a variable to be deemed zero when in fact it should simply have been much smaller than the original values that were subtracted. This can cause trouble during the computation of a Gröbner basis whenever a leading coefficient is deemed zero that should have been nonzero (in the sense that an exact computation would have given a nonzero value). This is analogous to losing a leading term in a univariate polynomial that has a small coefficient compared to other terms. In terms of consequences, one loses solutions that are "large" compared to the ones that are not lost. In the case where inputs are exact (or perhaps after rationalizing inexact inputs), a possible workaround would be to do a simultaneous modular Gröbner "trace" computation [Traverso 1988]. Even simpler, one might instead do an after−the−fact probabilistic verification by computing a Gröbner basis of the exact system, modulo some large prime, and checking the size of the solution set to make certain the same number were found (counting multiplicity). If not, then some were lost due to improper zero recognition, and the computation can be retried at higher precision.

In regards to efficiency, a reasonable question is how to proceed once one has a Gröbner basis. While it is not well understood how to (accurately) compute such a basis using only machine arithmetic, it is often possible to find eigenvalues accurately at machine precision. If this is all that one requires, well and good. In the case where high precision result is desired, a question arises as to whether it is more efficient to compute eigenvalues at high precision, or instead to get a machine precision approximation, likewise find values for remaining coordinates at machine precision, and use this as a starting point for a Newton−type iteration. `NSolve` currently uses the first approach because (i) In principle the eigenvalues computation can likewise take advantage of low precision approximations (in practice this is of course heavily dependent on implementation details of the eigenvalues code). (ii) A Newton iterative refinement procedure will not work in cases where there are more equations than unknowns. (iii) A low precision estimate may not be good for determining possible multiplicity, especially as eigenvalue extraction is often sensitive to multiplicity.

In comparison to other methods the algorithm described above tends to have greater need for high precision arithmetic. While many benchmark problems require perhaps a few dozen digits, it is not uncommon to need up to several hundred digits in order to get a Gröbner basis. It remain an open problem how to reduce this burden. Moreover the method of this paper usually cannot handle overdetermined systems at low precision, unlike e.g. that discussed in [Corless, Gianni, Trager, and Watt 1995]. An open question is whether one can merge ideas from this technology with that described in [Kravanja and Van Barel 2000] to get the high speed of machine arithmetic and the simplicity/flexibility of the type of solver described herein. A related question is whether ideas from those notes can be adapted to give all real−valued solutions, entirely bypassing complex ones. This cannot readily be done using eigenvalue methods.

These issues notwithstanding, the method implemented has much to recommend it. For one, it is largely achieved with building blocks that already exist e.g. Mathematica's `GroebnerBasis`, `PolynomialReduce`, and `Eigenvalues`, hence needs relatively little special code. (An exception might be the computation of the normal set to form an endomorphism matrix. It is aptly noted in [Corless 1996] that this is simple in principle but a bit tricky to code.) Another advantage is that the technology works well for a wide range of problems that ellude exact solvers, and allows people to solve many "real−world" problems that may otherwise be inaccessible. A nice example is provided in [Chao 2000] wherein accurate solutions to pairs of bivariate twelth order equations are obtained in real time to do semi−thin lens approximation magnetic optical beam envelope matching.

Examples

Below we present several examples of `NSolve` use. In some cases we suppress output for brevity and instead check that the results give small residuals. These were run on an 300 Mhz pentium processor with 128 Mb RAM, using the Linux operating system.

(1) Trinks system.

```
polys =
  {-36 - 165 B + 45 P + 35 S, 35 P - 27 S + 25 T + 40 Z, -165 B^2 + 25 P S - 18 T + 15 W + 30 Z,
   15 P T - 9 W + 20 S Z, -11 B^3 + P W + 2 T Z, 3 B^2 - 11 B S + 99 W};
vars = {P, S, B, Z, T, W};
Timing[sol = NSolve[polys == 0, vars];]

{3.16 Second, Null}
```

We check by substituting solutions into the polynomials and chopping residuals smaller than 10^{-8}.

```
Chop[polys /. sol, 10^-8]

{{0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0}}
```

(2) This problem is due to Michael Trott (private communication). Find all positive numbers x that satisfies the radical equation below.

```
Timing[sols = NSolve[-5 - x + 2 Sqrt[1 + x^2] - 3 (1 + x^3)^(1/3) + 5 (1 + x^5)^(1/5), x,
   Internal`SelectCriterion → (Head[#[[2]]] === Real && #[[2]] >= 0 &)]]

{7.85 Second, {{x → 1.54208}}}
```

(3) Katsura−5 system.

```
polys = {-x1 + x1^2 + 2 x2^2 + 2 x3^2 + 2 x4^2 + 2 x5^2 + 2 x6^2,
   -x2 + 2 x1 x2 + 2 x2 x3 + 2 x3 x4 + 2 x4 x5 + 2 x5 x6,
   x2^2 - x3 + 2 x1 x3 + 2 x2 x4 + 2 x3 x5 + 2 x4 x6, 2 x2 x3 - x4 + 2 x1 x4 + 2 x2 x5 + 2 x3 x6,
   x3^2 + 2 x2 x4 - x5 + 2 x1 x5 + 2 x2 x6, -1 + x1 + 2 x2 + 2 x3 + 2 x4 + 2 x5 + 2 x6};
vars = {x1, x2, x3, x4, x5, x6};
Timing[solns = NSolve[polys, vars, WorkingPrecision → 120];]

{46.79 Second, Null}

Max[Abs[Chop[polys /. solns, 10^-8]]]

0
```

(4) Find the (real−valued) intersection points of a pair of algebraically parametrized curves.

```
c1 = {x == Sqrt[-1 + t^2], y == -4 + t + t^2};

c2 = {x == 1 + s + s^2, y == Sqrt[-2 + 2 s + s^2]};
Timing[intersections = NSolve[Join[c1, c2], {x, y}, {s, t},
   Internal`SelectCriterion → (Head[#[[2]]] === Real &)]]

{0.41 Second, {{x → 2.50377, y → 0.572771}}}
```

(5) The following example is taken from [Lichtblau 2000]. It finds parameters that give six cylinders through a certain set of five points in $\mathbf{R}^3$. The configuration of these points is as follows. We start with a regular tetrahedron of edge length $\sqrt{3}$. Take another such tetrahedron and glue them together on one face. The five vertices are the points that the cylinders must contain. We do not show the entire solution set below but just the radii (which, as one expects from from symmetry considerations, are all the same).

```
polys =
  {1 - 2 b + b^2 + c^2 - 2 b c^2 + b^2 c^2 + 2 a c d - 2 a b c d + d^2 + a^2 d^2 - rsqr - a^2 rsqr - c^2 rsqr,
   4 + 3 a^2 + 4 b + 4 b^2 + 2 Sqrt[3] a c + 4 Sqrt[3] a b c + c^2 + 4 b c^2 + 4 b^2 c^2 - 4 Sqrt[3] d -
    4 Sqrt[3] a^2 d - 4 a c d - 8 a b c d + 4 d^2 + 4 a^2 d^2 - 4 rsqr - 4 a^2 rsqr - 4 c^2 rsqr,
   4 + 3 a^2 + 4 b + 4 b^2 - 2 Sqrt[3] a c - 4 Sqrt[3] a b c + c^2 + 4 b c^2 + 4 b^2 c^2 + 4 Sqrt[3] d +
    4 Sqrt[3] a^2 d - 4 a c d - 8 a b c d + 4 d^2 + 4 a^2 d^2 - 4 rsqr - 4 a^2 rsqr - 4 c^2 rsqr,
   2 a^2 + 2 Sqrt[2] a b + b^2 + 2 c^2 + b^2 c^2 + 2 Sqrt[2] c d - 2 a b c d + d^2 + a^2 d^2 -
    rsqr - a^2 rsqr - c^2 rsqr, 2 a^2 - 2 Sqrt[2] a b + b^2 + 2 c^2 + b^2 c^2 -
    2 Sqrt[2] c d - 2 a b c d + d^2 + a^2 d^2 - rsqr - a^2 rsqr - c^2 rsqr};
Timing[solns = NSolve[polys, {a, b, c, d, rsqr}, WorkingPrecision → Infinity];]
Sqrt[rsqr /. solns]

{1.3 Second, Null}

{9/10, 9/10, 9/10, 9/10, 9/10, 9/10}
```

Conclusions

We have presented a method to find the entire solution set, when it is finite, to a system of polynomial equations. The underlying technology uses numerical Gröbner bases computed with respect to a typically efficient term order, followed by formation of a multiplication matrix for an algebra endomorphism, eigenvalue extraction therefrom, and simple linear algebra and/or back−substitution techniques to reconstruct all solutions from this data. This can be done using machine arithmetic, extended precision, or even exact arithmetic. As indicated by several examples, this appears to be a powerful technique and it has considerably enhanced the capability of *Mathematica* to handle such problems.

References

W. Auzinger and H. Stetter (1988). An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. International Series of Numerical Mathematics **86**, R. P. Agarwal, Y. M. Chow, and S. J. Wilson eds., 11−31. Birkhäuser Verlag.

T. Becker, W. Weispfenning, and H. Kredel (1993). *Gröbner Bases*: *A Computational Approach to Computer Algebra*. Graduate Texts in Mathematics **141**. Springer−Verlag.

B. Buchberger (1985). Gröbner bases: An algorithmic method in polynomial ideal theory. In *Multidimensional Systems Theory*, chap 6. N. K. Bose, ed. D. Reidel Publishing Company.

Y.−C. Chao (2000). A full−order, almost−deterministic optical matching algorithm. To be submitted to Particle Accelerator Conference, 2001, Argonne, IL.

R. Corless (1996). Editor’s Corner: Gröbner bases and matrix eigenproblems. SIGSAM Bulletin: Communications in Computer Algebra **30**(4),26−32.

R. Corless, P. Gianni, and B. Trager (1997). A reordered Schur factorization method for zero−dimensional polynomial systems with multiple roots. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC ’97), W. Küchlin, ed., 133−140. ACM Press.

R. Corless, P. Gianni, B. Trager, and S. Watt (1995). The singular value decomposition for polynomial systems. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC ’95), A. H. M. Levelt, ed., 195−207. ACM Press.

D. Cox (1998). Introduction to Gröbner bases. In *Proceedings of Symposia in Applied Mathematics* **53,** D. Cox and B. Sturmfels, eds. 1−24. ACM Press.

D. Cox, J. Little, and D. O’Shea (1992). *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Computer Algebra*. Undergraduate Texts in Mathematics. Springer−Verlag.

J.−C. Faugére, P. Gianni, D. Lazard, and T. Mora (1993). Efficient change of ordering for Gröbner bases of zero−dimensional ideals. J. Symbolic Computation **16**:329−344.

P. Gianni and T. Mora (1988). Algebraic Solutions of systems of polynomial equations using Gröbner bases. In *Applied Algebra, Algebraic Algorithms, and Error−Correcting Codes* (AAECC 5). L. Hugeut,

A. Poli, eds. Springer LCNS **356,** 247−257. Springer−Verlag.

J. Keiper (1992). Numerical computation with *Mathematica* (tutorial notes). An electronic version may be found at:
http://mathsource.com/Content22/Enhancements/Numerical/0204−028

P. Kravanja and M. Van Barel (2000). Computing the Zeros of Analytic Functions. Springer Lecture Notes in Mathematics **1727**.

D. Lichtblau (1996). Gröbner bases in Mathematica 3.0. The *Mathematica* Journal **6**(4): 81−88.

D. Lichtblau (2000). Finding cylinders through 5 points in $\mathbf{R}^3$. Manuscript.

D. Manocha and J. Canny (1991). Efficient techniques for multipolynomial resultant algorithms. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '91), S. M. Watt, ed., 86−95. ACM Press.

H. Murao, H. Kobayashi, and T. Fujise (1993). On factorizing the symbolic U−resultant—application of the ddet operator—. J. Symbolic Computation **15**:123−142.

K. Shirayanagi (1993). An algorithm to compute floating point Groebner bases. Mathematical Computation with Maple V, Ideas and Applications 95−106. T. Lee, ed. Birkhauser Boston.

M. Sofroniou (1998). High−precision computations. (Talk given at the 1998 World−Wide *Mathematica* User's Conference). An electronic version may be found at:
http://library.wolfram.com/conferences/conference98/abstracts/high_precision_computations.html

M. Sofroniou and G. Spaletta. Precise Numerical Computation. To appear.

B. Sturmfels (1998). Polynomial equations and convex polytopes. American Mathematical Monthly **105**:907−922.

C. Traverso (1988). Gröbner trace algorithms. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC '88), P. Gianni, ed. Springer LCNS **358**:125−138. Springer−Verlag.

K. Yokoyama, M. Noro, and T. Takeshima (1992). Solutions of systems of algebraic equations and linear maps on residue class rings. J. Symbolic Computation **14**: 399−417.

S. Wolfram (1999). The *Mathematica* Book (4th edition). Wolfram Media/Cambridge University Press.