The Singular Value Decomposition for Polynomial Systems

Robert $\mathcal{M}$. Corless Patrizia $\mathcal{M}$. Gianni Barry $\mathcal{M}$. Trager Stephen $\mathcal{M}$. Watt*

IBM T. J. Watson Research Center
Yorktown Heights, NY 10598 USA

Abstract

This paper introduces singular value decomposition (SVD) algorithms for some standard polynomial computations, in the case where the coefficients are inexact or imperfectly known. We first give an algorithm for computing univariate GCD's which gives *exact* results for interesting *nearby* problems, and give efficient algorithms for computing precisely how nearby. We generalize this to multivariate GCD computation. Next, we adapt Lazard's u-resultant algorithm for the solution of overdetermined systems of polynomial equations to the inexact-coefficient case. We also briefly discuss an application of the modified Lazard's method to the location of singular points on approximately known projections of algebraic curves.

1 Introduction

We consider here the computation of useful and familiar algebraic quantities from sets of input polynomials whose coefficients are only imperfectly known. To do this, we reduce problems involving polynomials with floating point coefficients to more well-understood problems of numerical linear algebra, to take advantage of the well-developed backward error analysis of that field of study. We also use existing high-quality numerical linear algebra software, such as LAPACK [1], wherever possible, as the numerical stability and robustness of these codes is very well understood and tested.

Similar and related works include [2, 10, 13, 14, 15, 16, 17, 20, 21]. In addition, an anonymous referee has informed us of the μ-analysis toolbox in Matlab, written by Doyle, Packard, Tits, and others, which apparently uses optimization techniques to solve problems in the same spirit as this paper. It is now becoming apparent through all these works that symbolic and numeric computation can be usefully done together, combining the speed and low memory usage of numeric computation with the mathematical veracity of symbolic computation, in a sense to be made clearer below.

In section 2 we introduce our notation, discuss the singular value decomposition (SVD), and show how this can be used to compute univariate GCD's. This section also develops the idea of 'backward error analysis' in this context. This idea is also used in subsequent sections. This work is similar to that in [10, 17], and particularly to the former. That work was brought to our attention quite late in the process of preparing this paper, and so a full comparison of the significance of the differences between our work and theirs (principally that we use the 2-norm and the SVD whilst they use the 1-norm and a stabilized polynomial remainder sequence) cannot be attempted at this time. In section 3 we extend the univariate GCD computation to the multivariate case; this requires a reformulation of the standard algorithms to improve numerical stability, which seems to be new. In section 4 we look at the solution of possibly overdetermined homogeneous multivariate problems with only finitely many solutions at infinity. To do this we provide a constructive reformulation of an algorithm of Lazard [12], changing his determinantal algorithm to a generalized eigenvalue problem. This work is similar to that in [2, 13, 14, 15, 16, 20, 21], but differs in that the method of this paper can handle the overdetermined case.

2 Univariate GCD

Suppose we are asked to compute the GCD of two polynomials whose coefficients are given to only 2 decimal places, and expected to produce a 'satisfactory' answer. This is in contrast with the notion of 'quasi-GCD' of Schönhage [18], where the input polynomials are supposed to have 'exact' coefficients which can be known to arbitrary accuracy by some oracle.

For example, suppose we are given the polynomials $p = x^2 + 1.99x + 1.00$ and $q = x + 1.00$ and asked to compute the GCD of p and q. In the quasi-GCD approach of Schönhage, we would need to be able to refer to an oracle to get more figures of accuracy for the coefficients on demand, and the concept of quasi-GCD itself makes reference to the 'exact GCD' of the infinitely-precise input polynomials. We do not take this approach here. Instead, we work with the data we are given, but make a certain assumption as to its accuracy.

The fact that the coefficients are given above to two dec-

*This research was supported by IBM T. J. Watson Research Center and in part by the Natural Science and Engineering Research Council of Canada. Special thanks go to R. D. Jenks for helping to arrange the sabbatical visit of RMC to the Center. This research also benefited from the contribution of C.N.R., M.U.R.S.T, CEC contract ESPRIT B.R.A. n.6846 POSSO and the European Communities Science Plan Project "Computational Methods in the Theory of Riemann Surfaces and Algebraic Curves."

Reprinted from pp. 195–207 *Proc. 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC 95)*, A. H. M. Levelt editor, ACM Press 1995.

imals means by one convention[1] that the third and subsequent decimals are unknown, and may imply that we do not place much reliance on the second place. Thus there is an implied error tolerance (call it ε) on the *input*—we may be perfectly happy with an answer which is *exact* for some $p + \Delta p$ and $q + \Delta q$, where each of Δp and Δq is smaller (in some sense) than ε. For the problem given above, it seems reasonable to take $\varepsilon = 0.01$, or perhaps some moderate multiple of that.

Now it is obvious that for 'most' Δp and Δq the GCD will just be $(p+\Delta p, q+\Delta q) = 1$. It is equally obvious in this case that this is unsatisfactory—for $\Delta p = .01x$ and $\Delta q = 0$ we have $(p + \Delta p, q + \Delta q) = x + 1.00$, and this would be preferred in many cases.

However, there are in fact an infinite number of perturbations Δp and Δq which produce a monic degree 1 GCD; for example, if we restrict Δp to be $\Delta p_0 + \Delta p_1 x$ (i.e. leave the perturbed polynomial monic) and likewise Δq to be $\Delta q_0 + 0 \cdot x$, then the GCD of the perturbed polynomials will be $x + 0.99 + \Delta p_1 - \Delta q_0$ provided that the perturbations lie on a subspace determined by

$$0.01 + \Delta p_0 - \Delta p_1 + 0.01\Delta q_0 - \Delta q_0(\Delta p_1 - \Delta q_0) = 0\,.$$

Which of these possibly interesting GCDs should we compute?

We take as the answer to this question that the GCD we compute will be the one of highest possible degree with perturbations $||\Delta p|| \le \varepsilon$ and $||\Delta q|| \le \varepsilon$; if there are more than one such, then we take the one with the minimum 2-norm (Euclidean norm) of the perturbation $||\Delta p||_2^2 + ||\Delta q||_2^2$. This choice of norms is not arbitrary—it is chosen to facilitate computation. As previously mentioned, the work of [10] uses the 1-norm $||p||_1 = \sum |p_k|$ to measure polynomial size and a stabilized polynomial remainder sequence. Here we use the SVD to obtain precise results on the perturbation; these may well be equivalent to the precise results obtained in [10].

Some generalizations to the use of the 2-norm are possible at no real extra complication: in particular we can handle the case where different weights are given to the perturbations of different coefficients, corresponding to the case when different coefficients are known to different accuracies. Such elaboration will usually be left for the reader.

This approach to computing GCD's is based on a 'backward error' point of view. We wish to compute the *exact* answer to a possibly different question to the one posed. Note this is a more general usage of the term 'backward error' than that used in the context of proving that simple rounding errors in a computation can be interpreted as perturbing the problem; rounding errors will not be considered in this paper, as they are usually far less significant than the errors in the input data that we are allowing for. This approach is very useful provided that the input data is not known to great precision (or can be efficiently found to great precision). It may not be useful if the notion of 'exact answer' makes any sense for the problem at hand—in that case, the notion of 'quasi-GCD' of Schönhage [18] may be more useful. The algorithms considered in this work are intended for use in the case where a satisfactory answer is desired in the face of imperfect data.

Note that the problem of computing the GCD of two polynomials over the reals is 'ill-posed', in that arbitrarily small changes in the coefficients can make large changes in the answer. Thus we may expect the traditional techniques for dealing with ill-posed problems to be of some use. In particular, we take the approach here of projecting the problem onto the nearest degenerate problem (that is, the nearest problem with a non-trivial or higher-degree GCD), and then doing a *minimization*.

The main tool that we use for this is the Singular Value Decomposition. We explain this tool in some detail here, to make the exposition self-contained. To motivate the discussion below, note that the computation of GCDs can be phrased as a matrix problem: the degree of the GCD of two polynomials is related to the rank of their Sylvester matrix, and indeed the GCD itself can be read off from the last nonzero row of the reduced row-echelon form of the Sylvester matrix [6] as we will see.

2.1 Notation

Given a polynomial $p(x) = p_0 + p_1 x + \cdots + p_n x^n$ we associate with it the vector $p^* = [p_n, p_{n-1}, \ldots, p_0]^T$. We say that p has 2-norm $||p||$ equal to the 2-norm of its associated vector. Then $p(x) = p^* \tilde{x}_n$ where $\tilde{x}_n = [x^n, x^{n-1}, \ldots, 1]^T$.

We can represent polynomial multiplication by using the *Cauchy matrix*

$$C_q(p) = \begin{bmatrix} p_n & & & & & \\ p_{n-1} & p_n & & & & \\ \vdots & \vdots & \ddots & & & \\ p_0 & & & \ddots & & \\ & p_0 & & & p_n & \\ & & & \ddots & \vdots & \\ & & & & p_0 & \end{bmatrix},$$

and it is easy to see that $a(x) = p(x) \cdot q(x)$ if and only if $a^* = C_q(p)q^*$, if there are the right number of columns of $C_q(p)$, namely degree$(q) + 1$ (hence the subscript q).

Suppose $p(x)$ and $q(x)$ are polynomials in x, with the degree of p equal to n and the degree of q equal to m. Let $S(p, q)$ be the Sylvester matrix (see e.g. [6]) of p and q. Note it is of dimension $m + n$ by $m + n$, and is linear in the polynomial coefficients.

Remark 1. We can see the significance of the Sylvester matrix for polynomial computations most easily by considering the matrix-vector multiplication $S(p, q)\tilde{x}_{m+n-1}$. Linear combinations of the rows of S can be written as row-vector times matrix multiplications; if there are several combinations to be considered we can use matrix-matrix multiplication. If we multiply the row vector v^T by the Sylvester matrix $S(p, q)$ and then by $\tilde{x}_{m+n-1}$, we find $v^T S(p, q)\tilde{x}_{m+n-1}$ is

$$\begin{bmatrix} a_{m-1}, a_{m-2}, \ldots, a_0, b_{n-1}, \ldots, b_1, b_0 \end{bmatrix} \begin{bmatrix} x^{m-1}p(x) \\ x^{m-2}p(x) \\ \vdots \\ q(x) \end{bmatrix}$$

[1] The purely mathematical convention that all unstated digits are zero is perhaps more common in computer algebra; indeed assuming the input is exact is a commonplace in numerical analysis. We choose to follow in this article a different convention, the experimentalists' convention, to see if we can gain in simplicity of analysis and in speed of the algorithms. This convention has semantic implications—that is, it really alters the arithmetic model we are using, in the sense that we are really specifying both a number and a tolerance on input. Although not every arithmetic operation needs to be cognizant of this tolerance (as for example would happen in interval arithmetic) we will see that certain 'blocks' of operations will need this tolerance, and without it progress cannot be made.

or

$$\begin{aligned} &\left[(a_{m-1}x^{m-1} + \cdots + a_1 x + a_0)p(x)\right. \\ + \quad &\left.(b_{n-1}x^{n-1} + \cdots + b_1 x + b_0)q(x)\right], \end{aligned}$$

where the a_i and b_i are the entries of v^T, partitioned conformably with the rows of S. Thus we see that linear combinations of the rows of S are in a one-to-one correspondence with polynomial combinations of $p(x)$ and $q(x)$. Thus if we can find the linear combination of the rows of S that gives a row with the most leading zeros (while still having *some* nonzero entries) then we will have found the coefficients of the GCD of p and q. But this is just the last row of the row echelon form of S.

Thus for the example we began with, we must decide the rank of the matrix

$$\begin{bmatrix} 1.00 & 1.99 & 1.00 \\ 1.00 & 1.00 & 0.00 \\ 0.00 & 1.00 & 1.00 \end{bmatrix}.$$

and to compute the last row of its 'correct' row-echelon form.

2.2 Elementary Numerical Analysis and the SVD

The main tool in numerical analysis for deciding the rank of a matrix in the face of data perturbations (or, indeed, in the face of the usually much more trivial roundoff perturbations) is the Singular Value Decomposition, or SVD. We refer the reader to [8] for a geometric interpretation of the SVD, as well as details on how to compute it. We note that many packages exist for the reliable and efficient computation of the SVD, and in particular the LAPACK project paid particular attention to it.

Before we begin we describe the usual notation.

The *norm* of a vector v is, here, the Euclidean length of v, denoted by $||v||_2 = \sqrt{v_1^2 + \cdots v_n^2}$ or more simply by $||v||$. The 2-norm (Euclidean norm) of a *matrix* is

$$||A||_2 = \max_{||v||=1} ||Av||$$

or, equivalently,

$$||A|| = \max_{x \neq 0} \frac{||Ax||}{||x||}.$$

This implies that $||Ax|| \leq ||A|| \cdot ||x||$ for any vector x. Note that this norm is different from the more easily computed *Frobenius norm*

$$||A||_F = \sqrt{\sum a_{ij}^2}.$$

It can be shown that $||A||_2 \leq ||A||_F$.

The lengths of the semi-axes of the ellipsoidal image of the unit sphere under the mapping $x \to Ax$ are called the *singular values* of A, and are usually denoted by $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ in the n-dimensional case. $||A||_2$ is just σ_1.

A can be factored as $A = U\Sigma V^T$, where U and V are orthogonal and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a diagonal matrix. The orthogonal matrices U and V can be computed along with the σ_j if desired.

The most important property of the SVD for our algebraic purposes is that σ_k is the 2-norm distance to the nearest matrix of rank strictly less than k. This is Corollary 2.3-3 in [8]. In other words, if A has singular values σ_j, $j = 1, 2, \ldots, n$ arranged in the conventional decreasing order, and $A + E$ has rank strictly less than k, then $||E||_2 \geq \sigma_k$. Further, there is a matrix E with $||E||_2 = \sigma_k$ such that the rank of $A + E$ is strictly less than k. This matrix E is easily constructible from the SVD of A, as follows. Put $A + E = U\mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_{k-1}, 0, 0, \ldots, 0)V^T$. Then it is obvious that $A + E$ has rank strictly less than k. A simple computation shows that the norm of $E = A + E - A$ is the norm of $U\mathrm{diag}(0, 0, \ldots, 0, \sigma_k, \sigma_{k+1}, \ldots, \sigma_n)V^T$ which is σ_k since multiplication by orthogonal matrices does not alter the 2-norm.

2.2.1 Relationship with eigenvalues

Singular values are often confused with eigenvalues. However, the singular values σ_k of A are in fact the *square roots* of the eigenvalues of $A^T A$. If A happens to be symmetric positive definite, then indeed the singular values of A are the same as the eigenvalues of A; but in general they are not the same. Geometrically, singular values measure the amount of stretching A induces, without worrying about direction changes (i.e. rotation). In contrast, eigenvalues (if complex) also measure the amount of rotation that A induces.

2.3 Algorithms for Univariate GCD

Several algorithms exist to compute floating-point GCD's. We discuss only three here: Schönhage's quasi-GCD algorithm, which assumes the input is inexact but arbitrarily precise, Noda and Sasaki's scaled Euclidean algorithm [17], and Karcanias and Mitrouli's associated matrix pencil algorithm [11]. As mentioned previously, the method of Hribernig and Stetter [10] was brought to our notice too recently to permit a real comparison with our method.

Schönhage's algorithm does not do what we want. Here we are assuming that the input is inaccurate and the inaccuracy cannot be removed. Since the hypotheses on which the algorithm is based are assumed not to apply, we do not discuss this approach further.

Noda and Sasaki's scaled Euclidean algorithm is simple and efficient, but again is designed to do something slightly different than what we want. It can produce unsatisfactory answers on some simple examples, including the following one from Schönhage: $p = z^4 + z + 1$, $q = z^3 - \eta z$. For this example, for any small η, the GCD is 1. More than that, every polynomial set within 0.27 of this polynomial (with $\eta = 0$) in 2-norm also has GCD 1. We can, however, find values of η and reasonable tolerances (say $\eta = 10^{-3}$ and $\varepsilon = 10^{-6}$) such that Noda and Sasaki's algorithm computes a degree-1 GCD. This degree-1 GCD is completely spurious, in light of the above. We remark, however, that their algorithm often produces satisfactory answers, particularly for the approximate square-free decomposition, and it is efficient.

The algorithms of Karcanias and Mitrouli [11] are phrased in the jargon of automatic control and hence less accessible to a general audience. Their first algorithm, which they call the associated pencil algorithm, is equivalent to the algorithm of Lazard (see section 4) for the solution of polynomial systems, specialized to the univariate case. Their second algorithm, which they really advocate as being more efficient, algorithm 3.1 in [11], is quite different from the associated pencil approach and from the approach used here. However, their error analysis for their algorithm 3.1 is apparently incomplete: they analyze each step but this does

not guarantee the stability of the whole process (for example one can stably compute the characteristic polynomial of a matrix (using extra precision if necessary in intermediate steps and then rounding to working precision, for example), and then stably compute the roots of that polynomial, but it is well known that first computing the characteristic polynomial and then computing its roots is *not* a stable method for computing eigenvalues). Finally, one of their error bounds contains the term $||\hat{A}||_\infty^{d-1}$, where $\hat{A}$ is the companion matrix for one of the input monic polynomials, and d is the maximum degree of the input polynomials. This term (and hence the stated error bound) can be extremely large. Indeed for their example (4.2), this number is larger than 10^{120}, which induces a gross over-estimate of the actual error achieved by their algorithm.

The algorithm we present here for GCD computations is simpler and may be more reliable, but may also be more expensive than any of these. On the other hand, the expensive part of our algorithms can be carried out in purely numerical subroutines, perhaps in FORTRAN, and if the computer algebra system can take sufficient advantage of connections to good numerical libraries, as $A^\sharp$ can [24], the actual performance of this approach may not be bad at all.

It is possible that some of the ideas presented here may be adapted to improve instead, say, Noda and Sasaki's algorithm (or at least provide an *a posteriori* error analysis for it); however, the ideas used in this algorithm also prove useful in the approximate solution of polynomial systems, which is discussed in section 4.

2.4 Description of the SVD GCD algorithm

Input: Polynomials p and q, with $\deg(p) \geq \deg(q)$, and an error tolerance $\varepsilon > 0$.

Processing:

1. Form the Sylvester matrix S of p and q.
2. Compute the SVD of $S = U\Sigma V^T$.
3. Find the maximum k such that $\sigma_k > \varepsilon\sqrt{m+n}$ and $\sigma_{k+1} \leq \varepsilon$ (if all $\sigma_j > \varepsilon\sqrt{m+n}$ then set $d = 1$, and if there is no such 'gap' in the singular values then report failure). The index k is the declared rank of S, and the degree of d will be $n_d = n - k$.
4. Compute d by any of the following methods.
   (a) Compute d by the ordinary Euclidean algorithm (perhaps scaled as Noda and Sasaki do it [17]), terminating when the degree of the remainder is n_d.
   (b) (This rational method is speculative.) Form the top k rows of $U^T S$ or, equivalently, of ΣV^T. Compute the row-echelon form of this matrix by Gaussian elimination with partial pivoting. The numerical behaviour of this algorithm can be bad, depending on the conditioning of the matrix F which transforms this matrix to row-echelon form. It is possible that this problem may be dealt with by a careful variation of iterative refinement, taking care to simultaneously iterate for a perturbed Sylvester matrix.
   (c) Solve the minimization problem defined below, in section 2.6, by standard optimization techniques. This has the advantage that the backward error analysis (also discussed below) is all done at the same time, and is numerically stable.
   (d) Use the modified Lazard algorithm detailed in section 4, specialized to the univariate case, to find all the *roots* of the GCD and hence the GCD. This is equivalent to the matrix pencil algorithm of [11].

Output: A polynomial d of degree n_d which satisfies the following properties:

1. The polynomial d is the exact GCD of some pair of polynomials $p + \Delta p$ and $q + \Delta q$, where $||\Delta p|| \leq \hat{\varepsilon}$ and $||\Delta q||_2 \leq \hat{\varepsilon}$. We will discuss the quantity $\hat{\varepsilon}$ below, but note now that it is very easily bounded and only slightly less simply computed exactly *a posteriori*.
2. The degree of d satisfies
$$\text{degree}(d) = \max \text{degree}(\text{GCD}(r, s)) \, .$$
where the maximum is taken over all polynomials $r \in \mathcal{N}_{\hat{\varepsilon}}(p)$ and $s \in \mathcal{N}_{\hat{\varepsilon}}(q)$. By $\mathcal{N}_\alpha(p)$ we mean a closed α-neighbourhood of p in the 2-norm:
$$\mathcal{N}_\alpha(p) = \{q(x) | \deg(q) = \deg(p) \text{ and } ||p - q||_2 \leq \alpha\} \, .$$
3. Among all polynomials $(d^*, p + \Delta\hat{p}, q + \Delta\hat{q})$ satisfying the first two properties, the associated polynomials $p + \Delta p$ and $q + \Delta q$ are the closest to p and q in the least-squares sense.

Remark 2. Given an error bound, we have a well-defined concept of GCD. If an error bound is not known in advance, this algorithm can be used in several ways. For example:

- We can ask if a set of polynomials has a nontrivial GCD, and be assured that no polynomial within distance σ_k has one;
- We can instead ask how far away is the first set of polynomials with nontrivial GCD;
- We can ask for the polynomials *closest* to the given ones having a GCD of a given degree; or
- We can inspect the sequence $\{\sigma_k\}$ for jumps to determine candidates for a "natural" GCD.

Other variations will no doubt occur to the reader.

We point out that the negative results (e.g., no polynomials within ε have nontrivial GCD) are the easiest to obtain—the algorithm produces this type of result directly. On the other hand, to explicitly find the nearest polynomials with a GCD of specific degree requires the additional solution of two least-squares problems, but as we will see the problems involved give rise to banded, symmetric, Toeplitz, positive definite systems, which can be solved very efficiently.

2.5 More precise statements

Lemma 1. If $E = S(\Delta p, \Delta q) = S(p + \Delta p, q + \Delta q) - S(p, q)$, then $||\Delta p||_2 \leq ||E||_2$, $||\Delta q||_2 \leq ||E||_2$, and
$$||E||_2^2 \leq ||E||_F^2 = m||\Delta p||_2^2 + n||\Delta q||_2^2 \, ,$$
where $||E||_2$ is the 2-norm of the matrix E and $||E||_F$ is the Frobenius norm of the matrix E. Recall that the 2-norm of

the matrix E is, in contrast, the maximum value of $||Ex||_2$ that can be obtained for any unit vector x.

Proof. The last inequality is obvious from the definitions. The first two are consequences of the fact that $||E||_2 = ||E^T||_2$ and taking as unit vector $x = [1, 0, \ldots, 0]^T$ we have $E^T x = [\Delta p_n, \Delta p_{n-1}, \ldots, \Delta p_0, 0, \ldots, 0]$ which implies $||E||_2 \geq ||E^T x||_2 = ||\Delta p||_2$ by definition. A similar argument establishes the inequality for $||\Delta q||_2$.

Lemma 2. If the singular values of $S(p, q)$ are σ_j, $j = 1, 2, \ldots, m+n$, and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > \varepsilon\sqrt{m+n} > \varepsilon \geq \sigma_{k+1} \geq \cdots \sigma_{m+n}$ (in other words, the numerical ε-rank of S is k), and if $\hat{d}$ is a common divisor of $p + \Delta p$ and $q + \Delta q$ with degree$(\hat{d}) \geq m + n - k + 1$, then one of $||\Delta p|| > \varepsilon$ or $||\Delta q|| > \varepsilon$.

Proof. The nearest rank $m + n - k + 1$ matrix $S + E$ has $||E||_2 \geq \sigma_k > \varepsilon\sqrt{m+n}$. Thus $S(p + \Delta p, q + \Delta q) - S(p, q) = S(\Delta p, \Delta q)$ has $||S(\Delta p, \Delta q)|| \geq \sigma_k > \varepsilon\sqrt{m+n}$. But if both $||\Delta p|| \leq \varepsilon$ and $||\Delta q|| \leq \varepsilon$, then by Lemma 1 $||S(\Delta p, \Delta q)||_2^2 \leq m\varepsilon^2 + n\varepsilon^2$ or $||S(\Delta p, \Delta q)||_2 \leq \varepsilon\sqrt{m+n}$, a contradiction.

Lemma 3. Given a candidate approximate divisor d of p, we define $p + \Delta p$ to be the (least-squares) 'closest exact product' if there exists a polynomial $Q_p(x)$ such that $d \cdot Q_p = (p + \Delta p)$ exactly and Δp is as small as possible in the least squares sense. We can compute Δp by solving the following linear least squares problem: minimize $||\Delta p^*||$ where

$$\Delta p^* = D_p Q_p^* - p^* ,$$

with the $n + 1$ by $k + 1$ Cauchy matrix $D_p := C_p(d)$ (see section 2.1) whose columns are the coefficients of d. If d has degree $n_d = m + n - k$, where k is defined as the numerical ε-rank of $S(p, q)$, then we can bound the norm of $||\Delta p||$ by

$$\sigma_{k+1} \leq ||\Delta p|| \leq ||r_p||$$

where r_p is defined by $p = \hat{Q}_p d + r_p$ (i.e. ordinary polynomial division).

Proof: Immediate after the observation that polynomial multiplication of d by $Q_p = Q_0 + Q_1 x + \cdots + Q_k x^k$ can be written in linear algebra terms as the matrix-vector product $D_p Q_p^*$. The final inequality arises because if $p = \hat{Q}_p d + r_p$ then

$$-r_p^* = D_p \hat{Q}_p^* - p^*$$

which must have norm greater than or equal to the minimum value attainable, i.e. $||r_p|| \geq ||\Delta p||$.

Remark 3. Solution of this least-squares problem by the method of normal equations gives us a banded symmetric Toeplitz (positive definite because $A = D_P^T D_P$) matrix whose first column is $[d_0^2 + d_1^2 + \cdots d_{n_d}^2, d_0 d_1 + d_1 d_2 + \cdots d_{n_d - 1} d_{n_d}, \ldots, d_0 d_{n_d}, 0, \ldots 0]^T$ where only the first $n_d + 1$ entries are nonzero (this presumes $k > n_d$). Thus the bandwidth of the system is directly related to the degree of the putative GCD, while the size of A is $k + 1$ by $k + 1$. This system can be solved in $O(kn_d)$ floating point operations by the standard recurrences [8], which are particularly apt for this system with its banded structure. There are faster algorithms for solving non-banded symmetric positive definite Toeplitz systems, which take $O(k \log^2 k)$ operations, but in the usual case where the degree of the GCD n_d is small we can expect the standard (and in any event simpler) algorithm to be faster still.

The conditioning of A is also important. In the common case when the degree of the computed GCD is 1, then A is tridiagonal, symmetric, and Toeplitz; in that case we can explicitly calculate the eigenvalues in terms of the roots of the Chebyshev polynomials of the second kind. This gives that the condition number of A grows in the worst case like $4n^2/\pi^2 + O(n)$ where n is the degree of p; clearly for large n this could be a problem (say $n = 1000$, depending on the problem and the precision desired in the estimate for Δp). This is obviously not a critical problem, though for larger-degree GCD's the conditioning may be worse.

If conditioning of A prevents a satisfactory answer from being obtained, one can alter the Modified Gram-Schmidt algorithm for QR factorization to take advantage of the special form of D_p; if $D_p = QR$ then Q is upper trapezoidal in shape and R is upper $n_d + 1$-diagonal; still, this gives a more expensive algorithm than the normal equations. But the condition number of D_p grows only like $2n/\pi$ (in the degree one case), which may allow more accurate answers than the normal equations approach.

Remark 4. For the degree-one case, there is an explicit analytical formula for the minimum 2-norm Δp. We derive it as follows (this result may not be original to this paper). Suppose r is the root of our computed degree-one GCD. Then the minimum 2-norm distance from $p(x)$ to a polynomial $p(x) + \Delta p(x)$ with this root can be found by solving the following rank-deficient least-squares problem with the SVD [8].

$$p(r) + [1, r, r^2, \ldots, r^n] \begin{bmatrix} \Delta p_0 \\ \Delta p_1 \\ \vdots \\ \Delta p_n \end{bmatrix} = 0$$

But the SVD of the 1 by $n + 1$ matrix R for this linear system is trivial: $U = [1]$, $\sigma_1 = ||R||_2$ (as a vector), and V is some orthogonal completion of R/σ_1. This gives us as the minimum 2-norm solution

$$\Delta p^* = -p(r) R^T / \sigma_1^2 .$$

We see that the distance to the nearest polynomial with this zero is proportional to the *residual* $p(r)$.

There is a slightly simpler formula cited in [15] which gives the 1-norm distance to the nearest polynomial with the given root.

Definition. Suppose Δp has been computed by solving the associated least-squares problem. Suppose also that Δq has been computed. We define $\hat{\varepsilon} = \max(||\Delta p||, ||\Delta q||)$. This is our computable *a posteriori* bound on the distance to the nearest set of input polynomials with nontrivial GCD.

Remark 5. This definition and Lemma 3 provide two *a posteriori* backward error bounds, one cheap and crude and one slightly more expensive but wholly precise. It would also be interesting and useful to have a good *a priori* bound. We conjecture that one can replace the upper bound $\hat{\varepsilon}$ in the Lemma 2 with something like $M\varepsilon$ for a suitably moderate constant M, possibly depending on n and m.

2.6 Optimization Algorithm

If we define the near-GCD as the solution to the following minimization problem

$$\min_d ||C_p(d) q_1 - p||^2 + ||C_q(d) q_2 - q||^2$$

where $C_p(d)$ is the $n+1$ by $n - n_d + 1$ Cauchy matrix defined by the unknown coefficients of the near-GCD d of degree n_d, and similarly $C_q(d)$ is the $m+1$ by $m - n_d + 1$ Cauchy matrix,

and q_1 and q_2 are the unknown vectors of coefficients of the corresponding quotient polynomials, then we see we have a (quadratic) nonlinear least-squares problem to solve to find $d(x)$.

This can be done with standard optimization algorithms, and has as byproduct the backward error computations discussed in the previous sections, because clearly once $d(x)$ has been specified, the minimum is achieved on each piece of the function by choosing q_1 and q_2 appropriately. We are experimenting with this algorithm at present to see if it can be competitive.

2.7 Example

Suppose

$$p = x^5 + 5.503x^4 + 9.765x^3 + 7.647x^2 + 2.762x + 0.37725$$

and

$$q = x^4 - 2.993x^3 - 0.7745x^2 + 2.0070x + 0.7605$$

are given.

If we compute the GCD by the SVD algorithm above, we find that the singular values of the Sylvester matrix are approximately 23.1, 14.6, 7.62, 4.68, 3.59, 2.72, 1.11, .000141, and .611E−5. From this we can decide that there is a perturbation of the data not much bigger than $1.4 \cdot 10^{-4}$ such that there is a degree 2 GCD, and that the required perturbation to make a degree 3 GCD is at least 1.1—that is, the 2-norm of the vector of coefficients of Δp or Δq must be at least 1.1 to get a degree 3 GCD. Since the coefficients of the input were given only to four significant figures, we conclude that it is reasonable to look for a degree-2 GCD.

Now we continue with the algorithm and compute the candidate GCD

$$d = x^2 + 1.007x + 0.2534\,.$$

Then as we have seen in theory, d exactly divides some $p+\Delta p$ with $||\Delta p|| \leq ||r||$, where r is the remainder on division of p by d. This gives an upper bound on $||\Delta p||$ of roughly $9.7 \cdot 10^{-5}$. In fact the minimum possible perturbation to p that allows exact division by d has norm about $6.6 \cdot 10^{-6}$, as we discover on solving the appropriate least-squares problem.

Similarly, d exactly divides $q + \Delta q$ with $||\Delta q|| \leq ||r||$, where r is the remainder on division of q by d. This gives an upper bound on $||\Delta q||$ of roughly $5.7 \cdot 10^{-4}$, whereas the minimum possible perturbation to q that allows exact division by d has norm about $1.6 \cdot 10^{-4}$. Note that this is only slightly larger than the discriminating singular value of the Sylvester matrix, which was $1.4 \cdot 10^{-4}$.

The 2-norm of p is about 14, while the 2-norm of q is about 4. Thus we see that our divisor d is a better divisor of p than it is of q, but in both cases *relative changes in the input of less than* 10^{-4}, that is, of the *unknown digits of p and q*, allow us to say that our calculated GCD is exact.

Thus the algorithm provides a *proof* that there is no more satisfactory GCD than the one given, in the sense that the one computed has the highest possible degree consistent with the data. Further, we can explicitly compute the smallest perturbation of the initial data which makes the answer exact, so that we can verify that the problem we have actually solved is reasonable.

Contrast this with a 'forward error' approach, which requires you to know something more *a priori* about the 'exact' answer to the given problem.

2.8 Difficult cases

What do we do if there is no clear separation amongst singular values? This is likely to happen for large-degree polynomials. The algorithm above will simply report that it has failed; the user then must examine the singular values and make her or his own decisions. This is the best that can be done, in principle.

3 A first approach to multivariate GCD

It is often maintained that computation of GCD's is essentially a univariate problem, since we can compute multivariate GCD's by interpolation. Substantial modification of the standard exact algebraic interpolation algorithm is necessary, however, for a satisfactory algorithm in the approximate case. We describe this modification below.

For simplicity, let us consider a bivariate problem first. Suppose that the degrees of the input polynomials in x are less than the degrees in y; now consider taking a random value of x, say $x = \alpha$, and then consider the reduced problem of computing the GCD of $p(\alpha, y)$ and $q(\alpha, y)$. We use the SVD algorithm to find the GCD of these polynomials. Because of the randomness of α this will tell us both the degree and the sparsity pattern of the true GCD as a polynomial in y, with probability 1.

Now we take x as our main variable; we put x back as an indeterminate, and choose a certain number of random values β_i of y. For each β_i we compute the GCD of $p(x, \beta_i)$ and $q(x, \beta_i)$. Note that these GCD's will all be monic in x, whereas the true GCD may have both x and y in its leading term; hence the coefficients of the computed GCD's will be *rational* functions of β. However, we can take the denominator of these rational functions to be the *same* in every coefficient, and this allows us to do the (sparse) interpolation at little more than the cost of polynomial interpolation. Note that taking more points than necessary allows us to use ideas from least-squares approximation theory, and may allow us to compute better answers. The problem of 'unattainable' or 'near-unattainable' points for rational interpolation may be avoided in this fashion, as well.

In exact computation, each coefficient can be interpolated separately. For approximate computation this might allow slightly different denominators in each case, which would be unsatisfactory. So we interpolate all the coefficients simultaneously. We do this by setting up (conceptually) the linear system describing the interpolation problem, and then *semi-analytically* solving the problem—that is, reducing it to a sequence of smaller-order matrix problems.

Suppose there are T_x nonzero terms in each of the GCDs of $p(x, \beta_i)$ and $q(x, \beta_i)$. Suppose that we know that the GCD of $p(\alpha, y)$ and $q(\alpha, y)$ has T_y nonzero terms (and we know their powers also: e.g. $GCD(p(\alpha, y), q(\alpha, y)) = c_1 y^3 + c_2 y^5$ so $T_y = 2$ and the powers are known).

Then we must be able to fit rational functions $p_i(y)/p_0(y)$ to the T_x terms of the GCD's of $p(x, \beta_i)$ and $q(x, \beta_i)$. Note that the coefficient of the monic term is just $p_0(y)/p_0(y)$ and thus we get, for each β_i, only $T_x - 1$ equations in the $T_x T_y$ unknown coefficients of the interpolating polynomials. This tells us that if $T_x = 1$ we have to do something special, and in fact this case is easy—since we know there is only one term in x, and we know its power, just solve the reduced polynomial GCD problem with $x = 1$ and multiply the result by x^ℓ where ℓ is the known power of the GCD. If $T_x > 1$, then we can take M to be any number greater than or equal to $T_y T_x/(T_x - 1)$ (in particular $2T_y$ will do but may require

too many points if $T_x > 2$), then we need only solve M GCD problems in $n-1$ variables.

Consider a specific example for clarity. Suppose we are trying to recover the GCD $1+3y+3y^2x$ from its sampled values $x+6$ at $y=1/3$ (remember the GCD process takes a *monic* GCD), $x+9/4$ at $y=2/3$, $x+0$ at $y=-1/3$, $x-3/4$ at $y=-2/3$, $x+4/3$ at $y=1$, and $x+2/3$ at $y=-1$. Rationally interpolating the constant term of each GCD by $(p_0+p_1y+p_2y^2)/(q_0+q_1y+q_2y^2)$ (the degrees and nonzero terms are known by random sampling), we get the following system of linear equations.

$$p_0+\beta_i p_1+\beta_i^2 p_2-\gamma_i(q_0+\beta_i q_1+\beta_i^2 q_2)=0$$

for (β_i,γ_i) equal to $(1/3,6)$, $(2/3,9/4)$, ..., $(-1,2/3)$.

This is a sixth-order homogeneous system: we are looking for *an eigenvector corresponding to the zero eigenvalue*. If zero isn't an eigenvalue, then there is no rational system with the same denominator fitting the data.

For larger systems we quickly notice that the matrices multiplying the numerator coefficients are all the same Vandermonde matrix. This permits economical solution. Placing the denominator coefficients last in the list of unknowns leads to a block-bordered diagonal system that looks like

$$\begin{bmatrix} B & & & Y_1B \\ & B & & Y_2B \\ & & \ddots & \vdots \\ & & B & Y_pB \end{bmatrix},$$

where B is the M by T_x Vandermonde matrix associated with the chosen collocation points β_i raised to the known powers, and Y_i is the diagonal matrix of the known values of the i-th coefficient at each β_j. Calling this matrix A, then we are looking for a vector that makes $A\{p\}=0$. Such a vector will also make $A^TA\{p\}=0$, so we may confine our search to eigenvectors of the (square) matrix A^TA. Indeed since $A^TA=V\Sigma^2V^T$ if $A=U\Sigma V^T$ we see that the nullspace of A^TA is exactly the nullspace of A. This matrix is also sparse, and we can, by stable block row reduction using the SVD on B^TB (which doesn't alter the eigenvector we are looking for), reduce it to a system of the form

$$\begin{bmatrix} I & & & G_1 \\ & I & & G_2 \\ & & \ddots & \vdots \\ & & & M \end{bmatrix}.$$

Explicit formulas for the G_i and M are as follows.

$$G_i=(B^TB)^{-1}B^TY_iB$$

and

$$M=\sum_{i=1}^{T_y-1}(Y_iB)^T(Y_iB)-(Y_iB)^TBG_i\ .$$

We can find an eigenvector of *this* system by first finding an eigenvector of the lowest-order submatrix, M. This eigenvector gives the coefficients of the denominator polynomial. Finding the coefficients of the numerator polynomials is then simply a matter of matrix multiplication.

We find the zero eigenvector of M by using the SVD again. We simply take the last vector of V where $M=U\Sigma V^T$. It is possible that more than one vector will make $A^TA\{p\}=0$ (if for example there is a common factor $d(x)$ among all coefficients, which will obviously cancel after dividing out the leading coefficient), in which case we take the one of largest degree possible.

This process can be recursively iterated for the case of more than two variables. Some questions remain: how many points should we use for a good least-squares fit, and how should we choose them? Randomness is necessary only for avoiding coincidences (e.g. $(x^2+yx+1,x+1)=1$ if $y\neq 2$) once the degree of the GCD in y is known.

Using a purely random choice of interpolation points usually produces an unacceptably high condition number for the interpolation/approximation problem: we often observed condition numbers as high as 100, and on a problem with only three digits of accuracy in the input this means that only one digit will be accurate on the output. However, we have experimentally observed that it seems sufficient to randomly scale each variable and choose the Chebyshev points in the new variable (that is, put $x_k=\beta u_k+\alpha$ for some randomly chosen α and β, and then choose the M values $\cos(\pi j/M)$, $j=0..M-1$ for u_k). This typically produces condition numbers of 5 to 10, though if we are unlucky with our choice of β we can have very large condition numbers indeed so this must be monitored. We originally chose α and β as random variables on $(0,1)$ but later modified this to $(1/2,1)$. A good theory for the choice of points is needed.

We note that as implemented, there are two ways in which this algorithm can fail: due to unlucky random choices, or due to poor conditioning of the Vandermonde systems. So we must be able to check the answers that the program produces.

3.1 Backward error analysis

As in the univariate case we can compute the nearest polynomials to the input whose exact GCD is the one we have computed. If we assume that the nearby polynomials are *dense*, then there is no real difficulty in performing the calculation. We give an example below, which is example 7 in [17].

Put

$$f0=0.25-0.25x^2-0.25y^2-0.9999xy+xy^3+yx^3$$

and

$$f1=-0.00001+y-1.00001x+xy^2-yx^2+x^3-y^3\ .$$

If the coefficients are rounded in the obvious way the GCD is x^2+y^2-1. The algorithm as implemented computes slightly different answers on different runs (because of the randomness). A typical result is (with tolerance 0.0001)

$$\hat{d}=-0.99997+0.99937x^2+y^2\ .$$

Finding the quotient that makes $d(x,y)q_0(x,y)=f_0(x,y)$ is easily set up as a set of (incompatible) linear equations for the unknown coefficients of q_0. We get $Dq^*=f_0^*$, where the matrix D is sparse and similar in structure to the symbolic matrices in Lazard's algorithm (section 4). It is not printed here, for space reasons. We solve that set of equations in the least-squares sense to find that the smallest change that we can make in f_0 to make $\hat{d}$ an exact divisor has 2-norm of about 0.0000816. Similarly, we must (and can) change f_1 by 0.00001125 to make it exactly divisible by $\hat{d}$.

Thus we see that the algorithm has found the exact answer to a slightly different problem, a nearby dense one

within the specified tolerance. The fact that $\hat{d}$ is not so very different from $x^2 + y^2 - 1$ tells us that the original problem is not very sensitive to the errors in the input data.

If sparsity is important for the error analysis, that is, if the user is really only interested in getting the exact solution to nearby sparse problems, then it may not be true that the problem solved exactly is nearby (or even exists at all).

4 Solution of Multivariate Systems

The algorithms we discuss here for the solution of polynomial systems are similar to those of [2, 13, 14, 15], in that they use the SVD and generalized eigenvalue computations on a resultant-like matrix; however, the details of the computation are different. Aside from complexity differences, the main difference is that the algorithm we adapt, Lazard's algorithm, is capable of solving over-determined systems. (We note, however, that for the n equation in n unknown case, earlier methods [13] are superior.) In principle we can take some advantage of sparse matrix computations, for the algorithm considered in the present paper, but we have not yet done so, and in any event the matrices used here are not as sparse as in the $n \times n$ case.

We note that if we treat the input data as being exact, then overdetermined systems with approximately-known coefficients will, in general, have no solutions. Thus we are forced to treat the input data as being inexact, and allow the algorithm to pick out nearby interesting problems to solve.

The method is based in part on the recognition of certain matrices as being representations of multiplication by each of the variables in an affine ring, and hence that the matrices are commutative. This leads to an interesting approach to the computation of linear bases for polynomial ideals. See [7] for details.

4.1 Lazard's algorithm for solution of polynomial systems

In a following section we will be modifying Lazard's algorithm [12] to the present case of approximately-known input polynomials. This section contains a brief exposition by example of Lazard's method, for easy reference. We use Lazard's own example, with some additional remarks to prepare for the transition to the approximate case.

Consider the very simple system of equations

$$f_0 = -1 + 2x + y + x^2 + xy = 0 \tag{1}$$

$$f_1 = -1 + 3x + 2y + x^2 - y^2 = 0\,. \tag{2}$$

These equations have three finite solutions, $(0, 1)$, $(1, -1)$, and $(-3, 1)$, and one solution at infinity in the common asymptotic direction $(-1, 1)$.

Now put

$$f_2 = u + vx + wy\,, \tag{3}$$

where u, v, and w are scalar indeterminates. We now form a matrix system out of these three polynomials, from the coefficients of the monomials that occur in f_0, xf_0, yf_0, f_1, xf_1, yf_1, f_2, xf_2, yf_2, x^2f_2, xyf_2, and y^2f_2. Lazard's Theorem 3.3 [12] tells us how many such polynomials to construct, and what the maximum degree of the resulting monomials is (in this case, $D = 3$). Taking the monomials in lexicographic order, the resulting table is shown in Figure 1.

The table was constructed as follows. The exterior left-hand column lists the monomials of degree less than or equal to $D = 3$. The three interior parts correspond to the polynomials f_0, f_1, and f_2. The top row is a list of the monomials of degree less than or equal to $D-\text{degree}(f_i)$. If m is a monomial from the left-hand exterior column, and n is a monomial from the top row, then the entry in the table in that row and column intersection is the coefficient of m in the polynomial nf_i. For example, 3 is the coefficient of xy in yf_2.

It is clear that this is a generalized Sylvester matrix, and that if we put $\tilde{x} = [1, x, y, x^2, xy, \ldots, xy^2, y^3]$, then if Z is the matrix above then the components of $\tilde{x}Z$ are $f_0(x, y)$, $xf_0(x, y)$, $yf_0(x, y)$, $f_1(x, y)$, ..., and $y^2f_2(x, y)$. For the general problem, this matrix can get very large (though it is sparse).

Lazard works with this table as one large matrix. We will find it simpler to split it into four matrices, all purely numerical. For the moment, however, we split it only into two: the left-hand purely numerical part, which we will call Z, and the right hand symbolic part, which we will call M. Later we will split $M = uM_u + vM_v + wM_w$ into three numerical parts.

The algorithm proceeds as follows. First, we perform Gaussian elimination on Z; that is, we factor $Z = PLUR$ into its row-echelon factorization [4]. Here, Z is the first six (numeric) columns of the matrix in Figure 1, and it factors into $PLUR$ where P interchanges rows 5 and 7 and U is a 10 by 10 upper-triangular matrix, and the row-echelon form of Z is a 10 by 6 matrix with its main diagonals all equal to 1. We really need only P, L, and R for our purposes. R tells us that the *rank* of Z is $k = 6$, which we need, and we will apply $L^{-1}P^{-1}$ (or, rather, the bottom rows of this, corresponding to the nontrivial part of the result) to the symbolic matrix M. This gives Z_1, as given below (some entries are not printed, to save space).

$$Z_1 = \begin{bmatrix} -u - v + w & * & * & * & u & -u \\ -v + w & * & * & * & v & 0 \\ u - v + 2\,w & * & * & * & w & u + v \\ u + w & * & * & * & 0 & u + w \end{bmatrix}$$

We then go on to apply more Gaussian elimination, in order to compute a determinant. This determinant *factors into factors linear in u, v, and w*, which is the crucial result that the algorithm is based on. It turns out that this determinant is precisely the determinant of U in the row echelon factorization of $Z_1 = PLUR$, where P is a permutation matrix, L is unit lower triangular, U is a *nonsingular* upper triangular matrix, and R is the row echelon form. We remark that this determinant of U is necessarily nonzero when the computed factorization is correct on specialization—this computation gives one of the simplest examples of a useful 'proviso' [5].

The determinant of U is

$$(v - w)\,(u + w)\,(u - 3v + w)\,(u + v - w)\,.$$

From this determinant, we can read off the roots by the *coefficients* of the linear factors:

$$(0, 1, -1), \qquad (1, 0, 1), \qquad (1, -3, 1), \qquad (1, 1, -1)\,.$$

The first set of coefficients corresponds to the solution at infinity. There are *three* coefficients here because the method is based on homogenization of the original polynomial set, and hence we want to set the first coefficient to 1 (by scaling) if we can.

	1	x	y	1	x	y	1	x	y	x^2	xy	y^2
1	-1	$\cdot$	$\cdot$	-1	$\cdot$	$\cdot$	u	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
x	2	-1	$\cdot$	3	-1	$\cdot$	v	u	$\cdot$	$\cdot$	$\cdot$	$\cdot$
y	1	$\cdot$	-1	2	$\cdot$	-1	w	$\cdot$	u	$\cdot$	$\cdot$	$\cdot$
x^2	1	2	$\cdot$	1	3	$\cdot$	$\cdot$	v	$\cdot$	u	$\cdot$	$\cdot$
xy	1	1	2	$\cdot$	2	3	$\cdot$	w	v	$\cdot$	u	$\cdot$
y^2	$\cdot$	$\cdot$	1	-1	$\cdot$	2	$\cdot$	$\cdot$	w	$\cdot$	$\cdot$	u
x^3	$\cdot$	1	$\cdot$	$\cdot$	1	$\cdot$	$\cdot$	$\cdot$	$\cdot$	v	$\cdot$	$\cdot$
x^2y	$\cdot$	1	1	$\cdot$	$\cdot$	1	$\cdot$	$\cdot$	$\cdot$	w	v	$\cdot$
xy^2	$\cdot$	$\cdot$	1	$\cdot$	-1	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	w	v
y^3	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	-1	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	w

Figure 1: Lazard's matrix

Remark 6. The matrix M can be usefully split into three numerical matrices, each multiplied by a scalar indeterminate. Then the rectangular matrix F formed from the bottom $n-k$ rows of $L^{-1}P^{-1}$ (where $n=(D+1)(D+2)/2$ is the row-dimension of Z, and k is the rank) can be used on each piece of M:

$$\begin{aligned} FM &= F(uM_u + vM_v + wM_w) \\ &= u(FM_u) + v(FM_v) + w(FM_w) \\ &= u\hat{M_0} + v\hat{M_1} + w\hat{M_2} \qquad \text{say,} \end{aligned}$$

and now we recognize this as a *generalized eigenvalue problem*: we must find scalars u, v, and w which make the determinant of some $r \times r$ submatrix of FM equal to zero. Note that the matrix F should not be formed explicitly, but that the matrices FM_s can be formed during the row echelon process by row operations.

For arbitrary matrices M_u, M_v, and M_w, this problem will have no solution; in fact, we are looking for *over*-generalized eigenvalues. But these matrices are special. If we form (in this example) 4 by 4 submatrices M_i of the $\hat{M_i}$ by simply deleting the last two columns (in general we will have to be more sophisticated but for this example this is good enough), and then form $B = \alpha M_0 + \beta M_1 + \gamma M_2$ for some randomly chosen scalars α, β, and γ, then the matrices $A_0 = M_0B^{-1}$, $A_1 = M_1B^{-1}$, and $A_2 = M_2B^{-1}$ *all commute with each other*: $A_0A_1 = A_1A_0$, $A_0A_2 = A_2A_0$, and $A_1A_2 = A_2A_1$. This means, by a well-known theorem [9, Theorem 2.3.3] that this guarantees the existence of a unitary matrix U such that U^*A_iU are *all* upper triangular. Thus

$$\begin{aligned} & \det(uM_0 + vM_1 + wM_2) \\ = \; & \det\left(U^*(uM_0 + vM_1 + wM_2)B^{-1}U\right) \det B \\ = \; & \det\left(uT_0 + vT_1 + wT_2\right) \\ = \; & \det B \prod_{i=1}^{4} (ur_i + vs_i + wt_i) \end{aligned} \tag{4}$$

where each T_i is upper triangular: hence the determinant is just the product of the diagonal entries r_i, s_i, and t_i. That is, the determinant splits into linear factors (which we knew from [12] already). This means that such 'over-generalized' eigenvalues do indeed exist.

This point of view leads to a useful method of solution, which involves neither formation of large symbolic determinants nor factorization of such. Indeed the point of view leads to a useful algorithm in the symbolic context also.

Remark 7. Computation of the simultaneous triangularizing matrix U: Recent work shows how to simultaneously diagonalize symmetric commuting matrices [3], and it is possible that these techniques may be adapted to the nonsymmetric case. This is currently under investigation.

Remark 8. Why do those matrices commute? For details, see [7]. The following brief sketch gives the main ideas. Choosing a generic combination of the M_i matrices which is invertible is like choosing a hyperplane at infinity such that no solutions lie on that hyperplane. Let y be that hyperplane, i.e. $y = \sum c_i x_i$ such that y doesn't vanish on any solution.

Since $x_0, \ldots, x_n$ are homogeneous coordinates, $x_0/y, \ldots, x_n/y$ become affine coordinates of an affine ring which is also a finite dimensional vector space, whose dimension is the number of solutions of the system. Multiplication by x_i/y is represented by a matrix on this vector space. Since the ring is commutative, these matrices commute.

It turns out that M_iB^{-1} is similar to this multiplication matrix, i.e. it is the same transformation acting on a vector space with a different basis. So these matrices also commute.

Remark 9. It is crucial to be able to identify the rank of Z correctly, and it is here that the algorithm as stated first breaks down in the presence of data error. We will use the SVD to rectify this problem in the next section.

Remark 10. What goes wrong when there are an infinite number of solutions at infinity for the homogenized problem? We know that the theorems guaranteeing success do not go through; it is likely that what happens is that there are *no* $r \times r$ submatrices of $\hat{M_0}$, $\hat{M_1}$ or $\hat{M_2}$ that have full rank, which makes the pencil determining the eigenvalues singular.

4.2 Inexact coefficient version

Suppose that instead of the polynomials used in the example of the previous section, we are given the same polynomials divided by 3 and rounded to four decimal places, simulating input data error. What happens if we simply run the above algorithm?

It turns out that for this problem, it works fine, provided we don't attempt to take a determinant at the end and then factor it but rather solve an eigenvalue problem as below. But can we guarantee that it will always work well? No. Consider the problem of finding the roots of an overdetermined polynomial system (e.g. a GCD from polynomials with inexactly known coefficients). We know that Gaussian elimination on the Sylvester matrix (which is what

the matrix of Lazard's algorithm works out to be in the univariate case) will fail in the presence of data error—we will not be able to reliably determine the rank of the matrix and hence the number of solutions correctly. Other examples include linear systems in n variables whose matrices are ill-conditioned; in this situation it means they are systems close to ones with an infinite number of solutions.

How can we rescue this algorithm, in the presence of data error such as was discussed in the previous paragraphs? We use two ideas. The first is to use the SVD to correctly determine the rank of the numerical matrix; as in section 2 this will give us tight lower bounds on the necessary perturbations of the data to ensure that the solutions are correct.

The second idea is to solve the determinant factorization problem as a generalized eigenvalue problem. This avoids formation of the determinant as a polynomial in u, v, and w to begin with, which is well-known to induce an instability in the rootfinding process [25]. In effect, we will be generalizing the *companion matrix* method for finding roots of univariate polynomials: we will replace the polynomial rootfinding problem with a (generalized) eigenvalue problem. Philosophically, this approach is very similar to that of Manocha and Demmel [14, 15], and we hope therefore that it shares the good robustness properties of their algorithms.

We now re-solve the modification of Lazard's example to exhibit our modifications to the algorithm.

Since the coefficients of the polynomials were all divided by 3 and rounded to four digits, the numerical matrix Z is obtained from the previous Z by replacing 1/3 with 0.3333 and 2/3 by 0.6667. Instead of computing the row-echelon factorization of Z, we compute the SVD of Z, $Z = U\Sigma V^T$. U is a 10×10 matrix. Σ is a 10×6 matrix, the same shape as Z, and it turns out that the singular values range from 1.89 to 0.107, and we conclude that the rank of Z is indeed 6. V is a 6×6 matrix. We can form $U^TZ = \Sigma V$ and notice that the bottom four rows are all on the order of the machine roundoff.

But we really wish to form $U^TM = uU^TM_u + vU^TM_v + wU^TM_w$, and look at the last four rows of each of these. To do this it suffices to use the last four columns of U in the product directly to produce

$$\hat{M}_0 = U_1^T M_u$$

$$\hat{M}_1 = U_1^T M_v$$

and

$$\hat{M}_2 = U_1^T M_w\ .$$

As noted in passing in the previous section, for this example it suffices to define M_i as the submatrices obtained from the $\hat{M}_i$ by taking the first four columns of each. In general we can take r columns formed from random combinations of the columns of $\hat{M}_i$ (the same random combination for each matrix, of course). Here we get

$$M_0 = \begin{bmatrix} 0.1505 & 0.2816 & -0.1956 & -0.1529 \\ 0.3535 & 0.3554 & -0.3554 & 0.3510 \\ 0.4136 & -0.0574 & 0.3804 & 0.2388 \\ 0.2106 & -0.1312 & 0.5402 & -0.2652 \end{bmatrix},$$

and similarly for M_1 and M_2.

We then form a matrix B as a random combination of M_0, M_1, and M_2. Here we can for example choose $B = \alpha M_0 + \beta M_1 + \gamma M_2$, with $\alpha = 0.2190$, $\beta = 0.0470$, and $\gamma = 0.6789$, which gives a nonsingular matrix B (this happens with probability 1, and indeed we expect B to be well-conditioned, also). The nonsingularity of B, and its well-conditioning, may not matter if the eigenvalue problem is solved in a good way, as we will see, but it is important to take a generic combination to avoid spuriously multiple eigenvalues because multiplicity complicates the algorithm.

We now solve the *generalized eigenvalue problem*

$$\det(M_0 + wB) = 0 \tag{5}$$

In general we will be concerned with the conditioning of the eigenvalues and eigenvectors, and will wish to partition the eigenvector matrix into insensitive subspaces [14, 15], but for this example all eigenvalues are well-separated and the eigenproblem is very well conditioned. We thus get four linearly independent eigenvectors.

By Lemma 1.3.17 in [9] (modified for the generalized eigenvector case), these eigenvectors are *common to all the pencils* $M_i + \lambda B$. This means that if the matrix of eigenvectors is V, then $V^{-1}M_iB^{-1}V$ is upper triangular (in fact diagonal) for each matrix. Note that the rows of V^{-1} are left eigenvectors for each pencil also.

Thus our determinant becomes

$$\begin{aligned} & \det\left(V^{-1}(uM_0 + vM_1 + wM_2)B^{-1}V\right)\det B \\ = \quad & \det B \prod_{i=1}^{4}(ur_i + vs_i + wt_i)\ . \end{aligned}$$

Note that r_i may be expressed as $y_i^TM_1x_i$ where y_i is a left eigenvector and x_i is a right eigenvector corresponding to the i-th eigenvalue. Similarly $s_i = y_i^TM_1x_i$ and $t_i = y_i^TM_2x_i$. These formulas may also be arrived at by standard perturbation theory [22], and indeed that is how we first found them. These formulas can be expressed succinctly as the *Rayleigh quotient formula*: $r_{ij} = y_i^TM_jx_i$. If all of these quantities are small, then the root is *ill-conditioned*, and a system with a multiple root is nearby.

The following table shows the results of applying this formula to the current example.

i	$y_i^TM_0x_i$	$y_i^TM_1x_i$	$y_i^TM_2x_i$
1	0.3462	0.3460	−0.3460
2	0.3860	0.0000	0.3861
3	−0.1111	0.3332	−0.1110
4	0.0000	−0.1006	0.1006

(6)

The zero in the bottom left-hand corner was, in fact, $4\cdot10^{-15}$. Taking ratios, we get the (projective) roots (remember our input problem is about 10^{-4} different from Lazard's example)

$$\begin{matrix} 1.0000 & 0.9994 & -0.9993 \\ 1.0000 & -0.0001 & 1.0003 \\ 1.0000 & -3.0002 & 0.9999 \\ 0.0000 & 1.0000 & -1.0000 \end{matrix} \tag{7}$$

A decision was made for the last one that dividing by 10^{-15} would produce an 'effective infinity'.

Remark 11. The formulation of the coefficients of the factors as a generalized eigenvalue problem is quite independent of the method used to form the matrices M_u, M_v, and M_w; in particular this approach could be valuable in the purely algebraic case because the basic generalized eigenvalues and vectors that start the process off can be found by solving a *univariate* problem. Once that has been accomplished, the other roots can be read off directly by vector-matrix-vector products.

Remark 12. What if the matrix pencils are all singular? In this case we must have a nontrivial component at infinity; it is possible that this approach may shed some light on this

case as well. We are currently looking at this, and it seems that the Kronecker canonical form may play an essential role.

4.3 Multiple Roots

More work is necessary in the case when multiple roots are encountered. We must

- identify multiple eigenvalues by computing condition numbers (as in [15]);
- cluster them using standard techniques;
- (proposed, not yet tested) add one more linear equation to the original system (no more work needs to be done for this since matrices corresponding to linear equations have already been reduced in the formation of the matrix associated with a given variable);
- use the new matrices, which have constant eigenvalues, to find all values of the coordinates of the root by taking the average of the trace of the matrix;

however, none of these have yet been implemented. We anticipate success, but multiple roots are notoriously difficult.

4.4 Algorithm Overview

We indicate below just which parts of the algorithm have been actually implemented. The most important thing not yet implemented is the handling of multiple roots, which we did not need for our immediate application.

1 Form the numerical resultant matrix Z of the input polynomials.

2 Form the $n+1$ numerical M-matrices.

3 Compute the SVD of the resultant matrix, and its rank, k.

4 Form the bottom $r = \dim(Z) - k$ rows of $U^T M_{x_0}$, $U^T M_{x_1}$, ..., $U^T M_{x_n}$.

5 Form the r generic columns of each matrix (not yet implemented). Alternatively we could take an SVD of one matrix to find a basis for its column space (this is what is currently done), but the generic combination trick is cheap and effective. Call these r by r matrices M_0, M_1, ..., M_n.

6 Form a generic matrix B by a random combination of the M_i's. Genericity is important to avoid spurious multiplicities.

7 Find the left and right eigenvectors of $M_0 + \lambda B$.

8 Partition the returned eigenvectors into insensitive eigenspaces corresponding to clusters of multiple eigenvalues (not yet implemented). The left and right eigenvectors may be useful in this context.

9 Use the Rayleigh quotient formula $r_{ij} = y_i^T M_j x_i$ to find all the distinct roots.

10 For each higher-dimensional invariant subspace X_i of dimension $k_i \geq 2$, for each $j = 2, \ldots, n$, follow the steps outlined in Section 4.3 (not yet implemented).

4.5 Application to computation of singular points

One application of the solution of imperfectly-known overdetermined polynomial systems is the computation of singular points of algebraic curves where the projections of the curves are only known by interpolation of data known to a certain fixed number of decimal places. Mika Seppälä and Robert Silhol [19] wished to do this for the following example. After interpolation, their curve was $p(x,y) = 0$ where $p(x,y)$ was

$$4.0y^4 + 17.0x^2y^2 + 1.307xy^2 - 19.572938y^2 + 4.0x^4 \\ +5.228x^3 - 18.29175x^2 - 5.228x + 15.29175\ .$$

The level of error in this interpolated polynomial is not immediately apparent. To discover it *a priori* would involve careful analysis of the interpolation process used to create it. An anonymous referee pointed out that this problem is quadratic in y^2, and we could have split this problem into two univariate ones and applied the techniques of the first part of this paper.

The system that we wish to solve is $p(x,y) = 0$, $p_x(x,y) = 0$, and $p_y(x,y) = 0$. Note that if the input coefficients are treated as being exact, then there are no real solutions.

We ran our algorithm on this set of polynomials. On the first run, we discovered that there is precisely one real root, if the tolerance is taken to be larger than 0.000284. This is in fact the smallest nonzero singular value of the generalized Sylvester system, so if the tolerance is smaller than this, there are no solutions. The next smallest singular value is 0.101. The largest singular value is about 187 (this gives a natural scale for the problem). Thus we see a clear separation between singular values, and this gives us a good idea how accurate the input data was. We see that a relative change in the input data of about 0.000284/187 or $1.5 \cdot 10^{-6}$ turns the system from one that has no solution to one that has exactly one solution.

The algorithm then computed the singular point $x = 1.1838$, $y = 1.700 \cdot 10^{-7}$. Approaches using Gröbner bases to this problem produced very unsatisfactory results.

4.6 Implementation

The above procedure has been implemented in the Axiom Version 2 library extension language [23]. This platform, known as $A^\sharp$ during its development, provides the necessary symbolic facilities while allowing efficient access to libraries written in other languages [24]. For this paper we have made use of mature Fortran libraries to handle the singular value decomposition and the generalized eigenvalue sub-problems.

From a top-level view, the program to solve multivariate polynomial systems is expressed in about a dozen pages of source code (780 lines). The program makes use of the base Axiom Version 2 stand-alone library, an Axiom library package to represent matrices in Fortran form, and the routines DGESVD and DGEGV from LAPACK [1].

One inconvenience was that the foreign function interface does not yet have native support for Fortran calling conventions. The scalar arguments had to be placed in `Record`s to pass them, by reference, as Fortran expects.

Our first implementation of the program used the IBM ESSL library to compute the SVD and the generalized eigenvalues. This had a more convenient interface for the SVD, but the generalized eigenvalue routine was not as convenient.

Remark 13. The code is compiled with an optimizing compiler. This, and the interconnection to efficiently compiled Fortran code, allows much faster computation than one normally finds in a computer algebraic environment.

5 Concluding Remarks

The algorithms for the univariate and multivariate GCD computations described in the first part of this paper have been implemented in a computer algebra language, and are thus not restricted to machine-size tolerances and coefficients.

The algorithm described in the second part of this paper solves any zero-dimensional homogeneous system of equations. The method may also be applied to affine systems by adding an extra "homogenizing" variable, provided that when the system is homogenized the system has a finite number of solutions.

We have employed a technique of randomizing rank-deficient matrices, producing smaller matrices with rows or columns consisting of randomized combinations of rows or columns from the original matrix. This can be useful in reducing the cost of algorithms on dense matrices when they are not of full rank, since a rank-sized randomized matrix will carry full information.

From the implementation perspective, we observe that this symbolic-numeric work was greatly facilitated by making direct use of Fortran libraries for the linear algebra subproblems. An important aspect, which we hope becomes more common in computer algebra systems, is that we were not restricted to some predetermined set of foreign libraries. This allowed us to change the choice of linear algebra libraries, easily, quite late in the implementation.

Several research questions are left unanswered by this paper. In particular, we would like to see a good theory for the choice of collocation points in the multivariate GCD algorithm; an 'optimal' optimization method for the computation of univariate GCD's, perhaps using the SVD more directly; an implementation of the multiple root clustering heuristics in the solution of multivariate systems; proper use of sparse matrix technology; and an efficient algorithm for simultaneous upper triangularization of nearly commuting matrices. Finally, the connection of the Kroenecker Canonical Form to the case where the ideal is not zero-dimensional may be interesting to explore.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, 2nd. ed., SIAM, 1995.

[2] W. Auzinger and H. J. Stetter, "An Elimination Algorithm for the Computation of All Zeros of a System of Multivariate Polynomial Equations", in: Conference in Numerical Analysis, ISNM vol 86, Birkhaeuser, 1988, *pp. 11–30.*

[3] Angelika Bunse-Gerstner, Ralph Byers, and Volker Mehrmann, "Numerical Methods for Simultaneous Diagonalization", SIAM J. Matrix Analysis and Applications, **14**, no. 4, 1993, *pp.* 927–949.

[4] Robert M. Corless, David J. Jeffrey, and M. A. H. Nerenberg, "The Row Echelon Decomposition of a Matrix", *manuscript*, 1989.

[5] Robert M. Corless and David J. Jeffrey, "Well, it isn't quite that simple...", SIGSAM Bulletin **26** vol 3, August 1992, *pp.* 2–6.

[6] K. O. Geddes, S. R. Czapor, and George Labahn, *Algorithms for Computer Algebra*, Kluwer, 1992.

[7] P.M. Gianni and B.M. Trager, *in preparation.*

[8] Gene Golub and Charles Van Loan, *Matrix Computations*, Wiley-Interscience, 1981.

[9] Roger A. Horn and Charles A. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[10] V. Hribernig and H. J. Stetter, "Detection and Validation of Clusters of Polynomial Zeros", *preprint, to appear in J. Symb. Comp. 1995.*

[11] N. Karcanias and M. Mitrouli, "A Matrix Pencil Based Numerical Method for the Computation of the GCD of Polynomials", IEEE Trans. Automatic Control, **39**, No. 5, May 1994, *pp.* 977–981.

[12] Daniel Lazard, "Resolution des systemes d'equations algebriques", Theoretical Computer Science **15**, (1981) *pp.* 77-110.

[13] Dinesh Manocha, "Computing selected solutions of polynomial equations", Proc. 1994 ISSAC, Oxford, UK, *pp.* 1-8.

[14] Dinesh Manocha, "Solving Systems of Polynomial Equations", IEEE Computer Graphics and Applications, March 1994.

[15] Dinesh Manocha and James Demmel, "Algorithms for Intersecting Parametric and Algebraic Curves II: Multiple Intersections", *preprint.*

[16] H. Michael Möller and Hans J. Stetter, "Multivariate polynomial equations with multiple zeros solved by matrix eigenproblems", *preprint, to appear in Numerisch Mathematik, 1995.*

[17] Matu-Tarow Noda and Tateaki Sasaki, "Approximate GCD and its application to ill-conditioned algebraic equations", Journal of Computational and Applied Mathematics, **38**, (1991), *pp.* 335-351.

[18] A. Schönhage, "Quasi-GCD Computations", J. Complexity, 1981.

[19] M. Seppälä and R. Silhol, private communication.

[20] Hans J. Stetter, "Multivariate Polynomial Equations as Matrix Eigenproblems", WSSIA **2**, World Scientific, 1993, *pp.* 355–371.

[21] H. J. Stetter, "Verification in Computer Algebra Systems", in: Validation Numerics, R. Albrecht, G. Alefeld, H. J. Stetter, eds., Computing Suppl. **9**, 1993, *pp.* 247–263.

[22] G. W. Stewart, "Perturbation theory for the Generalized Eigenvalue Problem," in *Recent Advances in Numerical Analysis*, ed. C. deBoor and G. H. Golub, Academic Press, New York, 1978.

[23] S. M. Watt, P. A. Broadbery, S. S. Dooley, P. Iglio, S. C. Morrison, J. M. Steinbach and R. S. Sutor, *Axiom Library Compiler User Guide*, NAG Ltd, 1994.

[24] S. M. Watt, P. A. Broadbery, S. S. Dooley, P. Iglio, S. C. Morrison, J. M. Steinbach and R. S. Sutor, "A first report on the $A^\sharp$ compiler," in *Proc. 1994 ISSAC,* ACM Press, 1994, *pp.* 25-31.

[25] J. H. Wilkinson, "The Perfidious Polynomial", in *Studies in Numerical Analysis*, M.A.A. Studies in Mathematics, **24**, Gene H. Golub, ed., *pp. 1-28*, 1984.

[26] Richard Zippel, *Effective Polynomial Computation*, Kluwer Academic Publishers, Boston, 1993.