A Hybrid Approach for Determinant Signs of Moderate-Sized Matrices ⋆

Tim Culver John Keyser Dinesh Manocha

Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599

Shankar Krishnan

AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07392

Abstract

Many geometric computations have at their core the evaluation of the sign of the determinant of a matrix. A fast, failsafe determinant sign operation is often a key part of a robust implementation. While problems such as three-dimensional convex hull or Delaunay triangulation of points typically require determinants of order three or four, non-linear problems involving algebraic curves and surfaces involve larger matrices. Furthermore, the matrix entries often exceed machine precision, while existing approaches focus on machine-precision matrices. In this paper, we outline the earlier methods for computing exact determinant signs, and evaluate their effectiveness on moderate-sized matrices. We describe a hybrid method for computing the sign of the determinant. The stages include a floating-point filter based on the singular value decomposition of a matrix, an adaptive-precision implementation of Gaussian elimination, and a modular arithmetic determinant algorithm. We demonstrate our method on a number of examples encountered while solving polynomial systems.

Key words: Sign of matrix determinant; robustness; subresultant algorithm; singular value decomposition; adaptive precision arithmetic.

⋆ Supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract DAAH04-96-1-0257, NSF Career Award CCR-9625217, and ONR Young Investigator Award (N00014-97-1-0631)

1 Introduction

Determinant sign evaluation is an important problem for geometric applications. For some applications, a common technique for alleviating robustness problems is to express algorithmic decisions as determinant signs, and then focus on a fast, exact implementation of this operation. Researchers have developed many specialized techniques for determinant signs of small matrices, often focusing on 6×6 matrices and smaller.

In non-linear geometric computation, robustness requires effective handling of algebraic numbers. The key tool is often the Sturm sequence of a polynomial. One of the most effective ways to evaluate the Sturm sequence is with the subresultant algorithm, which expresses the terms of the sequence as matrix determinants. Often only the signs are needed. However, the matrix size grows with the degree of the polynomials. Evaluating a degree-n polynomial's Sturm sequence using the subresultant algorithm requires a determinant of order $2n - 1$ (and also the determinants of smaller matrices). For example, a three-dimensional algorithm requiring intersections of quadric surfaces involves algebraic numbers of degree eight, leading to subresultant matrices of order 15. Polynomials of much higher degree may be used as well. Of the known specialized methods for fast, exact determinant signs, most are ineffective for matrices of this size.

A further difficulty is the bit length of the matrix entries. In some linear problems, the matrix entries are simple formulae in the input data. But in non-linear computation, the matrix entries tend to have a bitlength significantly greater than the input bitlength, simply because the matrix entries are further removed from the input data. Specialized determinant sign algorithms often have a bit length restriction which is not satisfied in practice.

1.0.0.1 Overview. In section 2, we summarize the known techniques for computing the sign of the determinant of a moderate-sized matrix. In section 3, we describe several applications for determinant sign evaluation. The examples come from non-linear geometric computation in two and three dimensions.

Sections 4 through 6 present determinant sign algorithms. In section 4, we review the technique of modular arithmetic, including recent advances. In section 5, we present a new floating-point filter based on singular value decomposition. In section 6, we present a determinant sign method based on multiprecision interval arithmetic.

In section 7, we examine the performance of several algorithms on a variety of matrices encountered in non-linear geometric computation. In section 8, we weigh the strengths and weaknesses of the different methods and demonstrate that the meth-

ods may be combined into a powerful, efficient multistage method. We conclude in section 9.

2 Previous work

Many interesting techniques have been proposed for fast, exact determinant sign computation. The most powerful general technique computes the determinant of an integer matrix using modular arithmetic. The matrix determinant is computed modulo several primes, and the complete determinant is reconstructed from the residues. Brönnimann et al. [3] have recently improved the reconstruction step by avoiding the use of multiprecision arithmetic. The authors have also released an efficient implementation of their algorithm, which works for integer matrices with 53-bit entries. Wiedemann [25] computes the determinant of a sparse matrix in a finite field (modulo a prime number, for example) by computing the characteristic polynomial.

Avnaim et al. [2] give a specialized determinant sign algorithm for 2×2 and 3×3 matrices. The algorithm computes the determinant sign using arithmetic on b or $b+1$ bits, where b is a bound on the entry bitlength. Brönnimann and Yvinec [4] extend this algorithm, obtaining the "lattice method" for $n \times n$ matrices using arithmetic on $O(b + n)$ bits. Clarkson [6] proposes a determinant sign algorithm based on Gram-Schmidt orthogonalization. The orthogonalization preconditions the matrix for safe determinant sign computation by Gaussian elimination. For larger matrices, the algorithm requires arithmetic at a higher precision than machine precision. It is unclear how this algorithm would be implemented so as to be effective for larger matrices. Brönnimann and Yvinec [4] implement a variant of Clarkson's algorithm and demonstrate its performance primarily on matrices of order up to 6.

Much of the recent research is focused on *filtered computation*. For instance, one may compute the determinant in interval arithmetic. If the interval contains zero, the computation is repeated in higher precision or a slower exact method is used as a fallback. Many filtered methods are variations on this idea. Fortune and Van Wyk [11,12] analyze the structure of the determinant computation at compile time to obtain static error bounds, minimizing the runtime penalty of error analysis. The LEDA system [19] and the CORE system ([15], see also Yap [26]) provide numeric datatypes supporting runtime error analysis and automating higher-precision recomputation. For determinant sign computation, LEDA improves this model by incorporating static error bounds (Burnikel et al. [5]). Shewchuk [24] uses an unusual form of multiprecision arithmetic to compute signs of matrices up to 4×4. In this model, the actual error in each arithmetic operation is carried along, rather than a bound on the error magnitude. All of these methods are based on analyzing the forward error in a numerical computation.

To our knowledge, few authors have attempted a backward error analysis of a determinant algorithm. In two related articles, Pan et al. [21,22] compute the sign of the determinant by Gaussian elimination, and then validate the computation by estimating the distance from the given matrix to the nearest singular matrix.

3 Determinant signs in non-linear computation

The Sturm sequence is a basic tool for isolating the real roots of a univariate polynomial. Given a square-free polynomial $f(u)$ with integer coefficients, the Sturm sequence is $\{f_1(u), \ldots, f_m(u)\}$ where $f_1(u) = f(u)$, $f_2(u) = f'(u)$, and $f_i = -\operatorname{rem}(f_{i-1}, f_{i-2})$. Here $\operatorname{rem}(f, g)$ gives the remainder on polynomial division of f by g. Evaluating $\{f_i\}$ at a real number a and counting the number of sign changes in the sequence gives the *variation operator* $\operatorname{var}_f(a)$. The variation operator has the useful property that $\operatorname{var}_f(b) - \operatorname{var}_f(a)$ computes the number of real roots of f in the interval $[a, b)$. Multiple roots are counted only once. Using $\operatorname{var}_f()$, one may easily construct disjoint intervals, each containing one of the real roots of f. Such an interval, together with f, is a complete specification of an algebraic number.

Computing the Sturm sequence is an effective approach in some situations. But the length of the coefficients of the polynomials f_i can grow exponentially with i. In fact, the coefficient growth is avoidable. The integer coefficients of f_i share a large common factor. The *subresultant algorithm* of Collins [7] computes the coefficients of f_i/c_i, where c_i is a positive integer dividing the coefficients of f_i. The coefficient length of the modified sequence grows linearly. Further, the algorithm expresses the coefficients as matrix determinants. The matrices are submatrices of the resultant matrix of f and f'. Each of the coefficients may be computed individually, so one may compute only the constant terms of the $\{f_i\}$ and obtain $\operatorname{var}_f(0)$. In fact, we need only the signs of the constant terms. Thus, $\operatorname{var}_f(0)$ can be computed by evaluating determinant signs. The number of variations at a is computed by expanding $g(x) = f(x - a)$ symbolically and computing $\operatorname{var}_g(0)$.

For systems of polynomial equations, our approach is to construct a univariate polynomial whose roots are related to the roots of the system. Several techniques can reduce the dimension of the system. In this section, we describe three multivariate algebraic problems with geometric applications, and give a method for reducing each problem to a univariate problem.

3.1 Curve-line intersection

In the (s, t) plane, a line can be parametrized by u as $(a + bu, c + du)$. The line is intersected with the algebraic curve $f(s, t) = 0$. The intersection points correspond to the roots of $r(u) = f(a + bu, c + du)$. In this case, the substitution operation reduces a two-dimensional problem to a univariate problem.

This approach for curve-line intersection is used in the curve-curve intersection algorithm presented in Keyser et al. [16]. Each of two curves $f(s, t) = 0$, $g(s, t) = 0$ is intersected with each of the four walls of an axis-aligned box. The algorithm examines the configuration of the curve-line intersections and infers the presence or absence of a root.

3.2 Curve-curve root projection

The resultant of a pair of polynomials is a projection operation. Given two curves $f(s, t) = 0$, $g(s, t) = 0$, the Sylvester resultant with respect to s is a univariate polynomial $r(s) = \mathrm{res}_s(f, g)$ whose roots are the s-coordinates of the intersections of f and g. The roots are effectively projected onto the s-axis. This operation may also be viewed as eliminating the variable t from the system. "Elimination" here carries the same meaning as it does in Gaussian elimination.

Projection is used in the curve-curve intersection algorithm (Keyser et al. [16]). The x-resultant and the y-resultant are used together to compute possible locations for

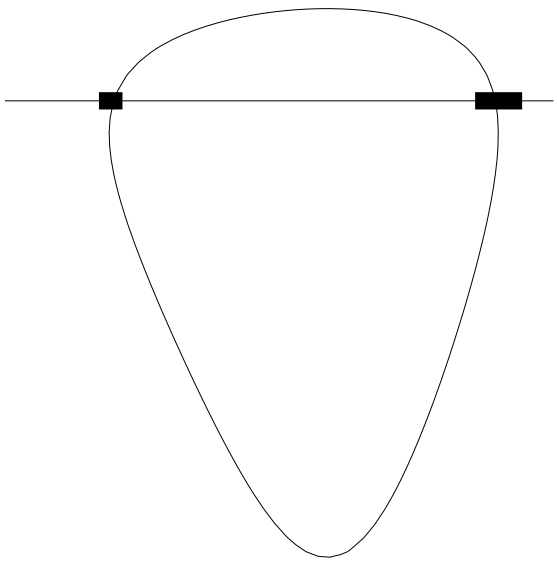

Fig. 1. Curve-line intersection. The curve $f(s, t) = 0$ is a torus-ellipsoid intersection curve, pulled back to the domain of one of the surfaces. The line is $t = \mathrm{const}$.

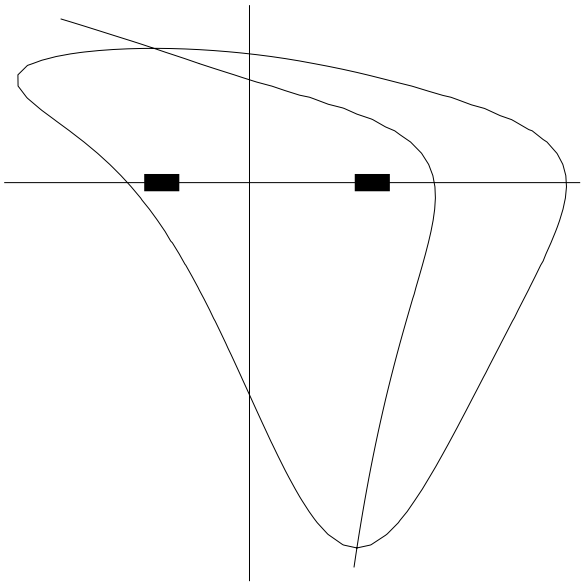

Fig. 2. Curve-curve root projection. The closed curve $f(s,t) = 0$ is a torus-torus intersection curve, pulled back to the domain of one of the surfaces. The other curve is $\partial f/\partial s = 0$. The roots are projected onto the s-axis by taking the resultant with respect to t.

roots.

3.3 Surface-surface-surface intersection

To intersect three algebraic surfaces $f(x,y,z) = 0$, $g(x,y,z) = 0$, $h(x,y,z) = 0$, one may use *multivariate Sturm sequences* to isolate the roots of the system. Multivariate Sturm theory was developed by Pedersen [23] and Milne [20]. Milne's method introduces a fourth variable u and a carefully-chosen fourth equation in x, y, z, and u. The variables x, y, z are then eliminated using a higher-order resultant. The remaining equation is a univariate polynomial $V(u)$, called the *volume function*, whose roots are in one-to-one correspondence with the roots of the three-dimensional system. In effect, the system is promoted to four dimensions, and then the roots are projected in such a way as to ensure that distinct roots have distinct projections.

Multivariate Sturm sequences are used in the polyhedral medial axis algorithm presented by Culver et al. [8]. For this application, f, g, and h are quadrics, and the roots of the system are points which are equidistant from four vertices, edges, or faces.

4 Modular arithmetic

The modular arithmetic approach computes $|A|$ in modular arithmetic over a collection of relatively prime numbers $\{p_i\}_{i=1}^m$. The determinant residues $d_i = |A| \mod p_i$ can be used to reconstruct the determinant residue over the much larger finite field of order $P = \prod p_i$. That is, $D = |A| \mod P$ is computed from the $\{d_i\}$. If D is then reinterpreted as an integer in the interval $(-P/2, P/2)$, and the actual determinant $|A|$ is known to be within the same range, then $D = |A|$.

The $\{p_i\}$ are chosen so that arithmetic modulo p_i can be implemented efficiently in a hardware-supported data type. For instance, a machine with fast IEEE double-precision floating point operations can use that data type for 53-bit integer arithmetic. In this system, one may use a modulus as large as 2^{27} while avoiding lost precision on multiplication. One may choose p_1 to be the largest prime less than 2^{27}, p_2 the largest prime less than p_1, and so on. Further, the $\{p_i\}$ must be chosen so that $\prod p_i > |\det(A)|$. This requires an a priori bound on the size of the determinant. Hadamard's bound states that

$$|\det(A)| \leq \prod_{i=1}^{n} \Big(\sum_{j=1}^{n} A_{ij}^2 \Big)^{1/2}$$

and so one uses just as many primes as is necessary so that their product exceeds twice this bound. See Knuth [17].

The usual choice for computing the modular determinant is fraction-free Gaussian elimination, although Wiedemann's algorithm [25] may be more efficient for large, sparse matrices. For the reconstruction step, the implementor has several options. An algorithm attributed to Lagrange appears to be the fastest in the typical case. An iterative algorithm attributed to Newton has the advantage that it allows early termination. The determinant may have significantly fewer bits than Hadamard's bound predicts. If $|\det(A)| < \prod_{i=1}^{m'} p_i$ with $m' < m$, then only $d_1, \ldots, d_{m'+1}$ need be computed. The case of a small determinant can be detected, albeit without absolute certainty, during reconstruction by noticing that $d_1, \ldots, d_{m'}$ yield the same answer as $d_1, \ldots, d_{m'+1}$. If this happens, reconstruction can be halted early. There remains the possibility that the answer is incorrect, with a probability of $1/p_{m'+1}$. Newton's algorithm is more efficient than Lagrange's when the determinant magnitude is significantly less than Hadamard's bound. See the cited work and Knuth [17] for the details of these reconstruction algorithms. A third reconstruction algorithm is given by Brönnimann et al. [3], based on Lagrange's algorithm. Theirs is different from all other known methods in that it requires only single-precision operations (assuming single-precision input), yet computes the sign correctly.

Modular arithmetic has two main advantages. First, the modular computation can be carried out in a fixed-precision arithmetic, chosen for its efficiency on the target platform. Second, the intermediate coefficients in Gaussian elimination grow expo-

nentially in length, typically exceeding Hadamard's bound on the ultimate result. As a consequence, a "bigint" implementation suffers from coefficient explosion, which modular arithmetic avoids.

5 A floating-point filter

We propose a floating-point filter based on the singular value decomposition of a matrix. Like most useful floating-point matrix decompositions, the SVD leads to an estimate of the determinant and its sign. However, the SVD has two additional advantages. First, it can be computed by a backward-stable algorithm (Golub and van Loan [13]). The error in the decomposition can be measured without assuming that the maximum error occurs at each arithmetic operation. Second, a tight bound is known on the error in the smallest singular value, which is the distance to the nearest singular matrix. The SVD assists in the computation of this bound. These properties allow us to use the SVD both to find the determinant sign and to decide whether the computed sign is reliable. As we will show, the computed sign is reliable whenever the matrix is well-conditioned—a criterion that is decided by a threshold on the condition number κ of the matrix.

The singular value decomposition of a square matrix $A_{n\times n}$ is a factorization $A = P\Sigma Q$, where P and Q are orthogonal, and Σ is a diagonal matrix with positive entries σ_i. By convention, the σ_i are placed in nonincreasing order $\sigma_1 \geq \cdots \geq \sigma_n$. The largest singular value σ_1 is equal to the Euclidean matrix norm $\|A\|_2$. The smallest singular value σ_n is equal to the Euclidean distance to the nearest singular matrix to A. The SVD has many useful properties and many applications. For details, and an algorithm based on QR iteration, see Golub and van Loan [13].

The SVD can be computed in floating-point arithmetic with the following backward error bound. The computed matrices $P\Sigma Q$ are the exact singular value decomposition of a perturbed matrix $A + E_{SVD}$, where

$$\|E_{SVD}\|_2 \leq f_1(n) \cdot \epsilon \cdot \|A\|_2.$$

The value ϵ is machine epsilon. The function $f_1(n)$ is less than $100n^3$ (Demmel [9]) but experience suggests that $f_1(n) = n$ is a safe assumption for the problem at hand. The appropriate value for $f_1(n)$ is discussed further at the end of this section.

Demmel and Kahan [10] improve the traditional SVD algorithm. Their modified QR algorithm is forward stable as well as backward stable. The authors prove fairly tight error bounds on all of the singular values computed by their algorithm, whereas the traditional QR algorithm computes the smaller singular values with less accuracy. In the modified QR algorithm, the relative error in each singular

value is on the order of $n\epsilon$. This can be expressed:

$$|\sigma_i - \tilde{\sigma}_i| \leq f_2(n) \cdot \epsilon \cdot \sigma_i \tag{1}$$

where σ_i is a singular value of A, $\tilde{\sigma}_i$ is its computed value, and $f_2(n)$ is a modestly-growing function of n, which may be safely assumed to be $O(n)$ for the values of n we consider. The modified QR algorithm is implemented in the LAPACK system [1].

By combining the forward and backward error bounds, we can obtain conditions for the correctness of the determinant sign as computed by the SVD. Let A be an exact integer matrix, and $A + E_A$ its floating-point representation, with $\|E_A\| \leq \epsilon\|A\|$. By backward stability, the SVD algorithm computes the sign of the determinant of a matrix $A + E_A + E_{SVD}$ for a small perturbation E_{SVD}. On the other hand, by forward stability, the distance from $A + E_A$ to the nearest singular matrix is computed accurately as $\sigma_n(A + E_A)$. If the net perturbation of A is less than the distance to the nearest singular matrix, then the computed sign is correct. That is, the sign is correct if

$$\|E_A + E_{SVD}\| < \sigma_n \tag{2}$$

Neither of these quantities is available exactly. However, the following inequality can be tested, and if it holds, it is sufficient to establish equation (2) and the correctness of the sign.

$$(f_1(n) + 1) \cdot \epsilon \cdot \sigma_1 < \sigma_n. \tag{3}$$

The sufficiency of (3) is established using the backward stability of the SVD algorithm, which ensures that $\|E_{SVD}\|$ is small.

$$\begin{aligned} \|E_A + E_{SVD}\| &\leq \|E_A\| + \|E_{SVD}\| \\ &\leq \epsilon \cdot \|A\| + f_1(n) \cdot \epsilon \cdot \|A\| \\ &= (f_1(n) + 1) \cdot \epsilon \cdot \|A\| \\ &= (f_1(n) + 1) \cdot \epsilon \cdot \sigma_1 \end{aligned}$$

Rearranging equation (3), we find that it expresses a static bound on the condition number $\kappa = \sigma_1/\sigma_n$:

$$\sigma_1/\sigma_n < \frac{1}{(f_1(n) + 1) \cdot \epsilon}. \tag{4}$$

In other words, the SVD algorithm dependably computes the sign of the determinant whenever the matrix is well-conditioned. Moreover, the notion of "well-conditioned" is explicitly defined in terms of a largest allowable condition number, determined by the matrix size and the working precision.

To evaluate the test (4), we use the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_n$. These are not the exact singular values, but they are close. We need to weaken the test slightly to

account for the error present in the singular values. Taking advantage of forward stability (1), we find absolute error bounds on the actual singular values σ_1, σ_n in terms of the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_n$. We use the upper bound on σ_1 and the lower bound on σ_n. The computed version of (4) is

$$\tilde{\sigma}_1/\tilde{\sigma}_n < \frac{1}{(f_1(n)+1)\cdot\epsilon} \cdot \frac{1 - f_2(n)\cdot\epsilon}{1 + f_2(n)\cdot\epsilon} \tag{5}$$

The procedure for computing the determinant sign is straightforward.

(1) Approximate the integer matrix A with a double-precision floating-point matrix and compute the singular value decomposition $P\Sigma Q$.
(2) Compute the condition number from Σ and the determinant sign from P and Q.
(3) If κ is small (as defined by equation (5)), accept the determinant sign.

If κ is large, the filter fails, and we must use another method to find the sign.

Many systems offer some form of extended precision arithmetic implemented in software, with a precision of ϵ'. If extended precision is available, the same value of κ can be compared against the looser static bound involving ϵ'. That is, we know whether the extended-precision SVD will give a dependable answer before we compute it. Lastly, the determinant magnitude can be estimated by $\prod \tilde{\sigma}_i$, so even when the filter fails, the SVD reveals a good guess as to whether Lagrange-style or Newton-style reconstruction will be more efficient if modular computation follows. This idea is developed in section 8, in which we propose a multistage strategy.

Reliability of filters based on backward error estimation. It should be noted that backward error bounds in numerical analysis are usually viewed as "approximate error bounds." In other words, the error bounds are an accurate estimate of the *actual* net error committed. The bounds are not mathematically proven to be strict upper bounds. Often such bounds are known only up to a constant factor.

Therefore, the possibility remains that a determinant sign filter based on backward error estimation, like our SVD filter, could return an incorrect sign and certify it as correct. Using the most conservative error bounds, such as $f_1(n) = 100n^3$ in the backward error in LAPACK's SVD algorithm, does not remove this possibility.

Yet in practice, the error bounds are accurate. In our tests, the SVD filter always returned either the correct sign or a warning that the sign was not to be trusted. Experimentation showed that the less conservative bound $f_1(n) = n$ was still conservative enough for all matrices in our tests.

6 Multiprecision interval arithmetic

In this section, we describe a means for computing the determinant sign using multiprecision floating-point arithmetic with explicit maintenance of lost precision. This method complements the methods in sections 4 and 5. The arithmetic system is described in detail in Krishnan et al. [18]. The system forms the foundation of the PRECISE library for matrix operations and polynomial root finding.

The numerical data structure consists of a sign, a mantissa, an exponent, and a bound δ on the absolute error. The mantissa is represented as an arbitrary-length array of digits. Only significant digits are stored, so as the error increases through the computation, the mantissa length decreases. This strategy has two principal advantages. One is that arithmetic speeds up as precision is lost. The other advantage is that the error always lies entirely in the last digit, allowing δ to be represented by a single-precision quantity. Our number type behaves like interval arithmetic over a "bigfloat" type (such as LEDA's), though it requires only one multiprecision computation per operation, and a single precision computation to maintain δ, rather than two multiprecision operations.

To perform a computation over our numerical type, we first specify a *fixed* maximum precision P for the computation. First the input data are rounded off to P bits, and then as the computation progresses, numbers are computed only to a precision of P bits. The bounds δ keep track of the forward error in the computation. If the result has enough precision, the computation terminates. For instance, if one is interested in the sign of an expression, the result's interval is checked to see whether it contains zero. If the result is too imprecise, P is increased and the computation restarts. By controlling P, the system adapts the precision to the forward stability of the computation. The performance of any algorithm in this framework clearly depends on the number of times the computation has to be repeated.

The possibility remains that the result is exactly zero. In general, this situation is difficult to resolve. But if the result is known to be an integer, as it is in our application, the sign is established whenever the interval length is less than one.

We compute the determinant sign by performing interval Gaussian elimination with partial pivoting. The algorithm starts by fixing the initial precision P for the computation. Ideally, P should be set to its ideal value P_{ideal}: the smallest value such that the sign of the diagonal entries after Gaussian elimination can be determined unambiguously. A larger value for P results in extra work at each arithmetic operation. A smaller value also results in extra work, in the form of restarts. The ideal value P_{ideal} is difficult to compute, so we describe an estimation method which has been successful in practice. We begin by setting

$$P = \alpha \cdot n \cdot \log \|A\|_F,$$

where α is a constant less than one (0.1 in our implementation). It is easy to see that if $\alpha = 1$, this bound is similar to the determinant bitlength predicted by Hadamard's bound. During the course of Gaussian elimination, if any diagonal element loses enough precision so that its sign is indeterminate, we increase P and redo all the computation. Suppose that the pivot after i steps of Gaussian elimination ($0 \leq i \leq n-1$) is found to have an ambiguous sign. We then increase P, noting that the closer i is to n, the closer P is to P_{ideal}. The increase is given by

$$\Delta P = \alpha \cdot i \cdot \log \|A_i\|_F.$$

where $\|A_i\|_F$ is the absolute value of the largest element in the lower right-hand submatrix of A before column i is eliminated. (A_i is therefore a matrix of size $(n-i) \times (n-i)$.) In our experiments, this estimate is good especially when the determinant size predicted by Hadamard's bound is pessimistic.

The principal difference between PRECISE's number type and adaptive precision schemes like LEDA's `real` type and CORE's `Expr` type is that PRECISE does not store the structure of the determinant expression at run time. The disadvantage to our approach is that PRECISE's numbers are not drop-in replacements for built-in numerical datatypes. The outer loop for repeating the computation at higher precision must be explicitly programmed. The advantage is that the structure is not stored. As most determinant algorithms perform $O(n^3)$ arithmetic operations, the cost of maintaining the expression structure is appreciable for even a moderate-sized matrix.

7 Empirical results

To evaluate our filter system, we chose several representative algebraic problems of the type described in section 3. The table in figure 3 describes the problems in terms of their algebraic degree and bit complexity, and characterizes the matrix determinants in terms of their size and bit complexity. In all cases, the curves and surfaces are derived from inputs with reasonable bit lengths. The algebraic curves are intersection curves of various ellipsoids and tori. An ellipsoid is defined in terms of a center point $\mathbf{p}$ and three axis vectors $\{\mathbf{v}_i\}$, while a torus is defined in terms of a vector frame $\{\mathbf{v}_i\}$ and two radii r_1, r_2. The surface-surface-surface problem intersects three quadric surfaces. The surfaces are line-line bisectors, where each line is defined by two points $\mathbf{p}_1, \mathbf{p}_2$. For the examples considered here, all of the quantities $\mathbf{p}, \mathbf{p}_i, \mathbf{v}_i, r_i$ are given as 32-bit integers, and most require fewer than eight bits.

The SVD algorithm of James Demmel is implemented in the LAPACK library [1]. For the modular algorithm, we use Patrick Theobald's implementation in the LiDIA library [14], which uses Lagrange's interpolation algorithm. We have modified the

Problem	Degree	Deg $f(u)$	Coef bits $f(u)$	Max matrix n	Matrix entry bits
Curve-line	4	4	32	7	34
intersection	8	8	56	15	59
	16	16	54	31	58
Curve-curve	4,4	8	105	15	107
root projection	8,7	24	156	47	160
	8,7	28	157	55	161
Surface-surface-surface	2,2,2	8	2820	15	2822
root isolation					

Fig. 3. Algebraic and numerical complexity of the geometric problems used to generate the test matrices.

library to support Newton's algorithm as well. Sylvain Pion's implementation of the Brönnimann et al. algorithm [3] is very efficient for matrices with 53-bit entries, but unfortunately our matrices do not fit this restriction. The multiprecision interval arithmetic is our implementation in the PRECISE library [18].

Figures 4, 5, and 6 show running times of the determinant-sign algorithms on the matrices generated by our examples. The matrices from the various problems are pooled, then collated into three categories: well-conditioned, ill-conditioned but nonsingular, and singular.

One important observation from our experiments is that small determinants are rare. When we examined the determinant bitlength, we found that it was always at least 80% of the length predicted by Hadamard's bound, and usually above 95%. Newton reconstruction takes about the same amount of time as Lagrange reconstruction when the determinant bitlength is large. Figures 4 and 5 omit the Newton algorithm for this reason. As the determinant magnitude grows, the running time is dominated by Gaussian elimination, and the choice of reconstruction algorithm matters less.

Singular matrices, on the other hand, do occur, and Newton's algorithm far outperforms Lagrange's algorithm on these matrices. The matrices are ill-conditioned by definition, so the SVD filter fails. Yet the SVD does reveal an estimate of the determinant magnitude which can be used to choose between Newton and Lagrange reconstruction. Since full Lagrange reconstruction is unnecessary on these matrices, Lagrange time is absent from figure 6. PRECISE requires significantly more time on singular matrices—so much more that it goes from an order of magnitude faster than modular arithmetic to an order of magnitude slower. Singular matrices are typically a worst-case input for methods based on forward error.

In all cases, the floating-point SVD algorithm takes a fraction of the time of the modular algorithms. In the data in figures 4 and 5, the ratio of SVD time to mod-

ular time never exceeds 0.28, and is less than 0.06 for larger matrices ($n > 10$). In figure 6, the SVD algorithm runs in about one-fourth the time of the Newton algorithm.

Unfortunately, the SVD filter fails on all matrices above $n = 15$ in our experiment. One might think that this due to overly pessimistic bounds on the error in the SVD algorithm, but in fact, the SVD computes an incorrect sign for about one-third of the ill-conditioned matrices in the experiment. Large matrices are not necessarily ill-conditioned in general, but all of the large matrices generated by our test cases are ill-conditioned.

In a separate experiment, we ran the algorithms on matrices with $5 \leq n \leq 53$ with uniformly-distributed 100-bit random entries (figure 7). Such matrices tend to be well-conditioned, and the SVD filter succeeded on all of the matrices we tried. Such a distribution might occur in a high-dimensional convex hull or Delaunay triangulation algorithm.

Since the subresultant matrices have a sparse structure, we implemented Wiedemann's algorithm [25]. The algorithm computes the characteristic polynomial over a finite field by iteratively multiplying the matrix by random vectors. Since the input matrix is used only for matrix-vector multiplication, the running time depends on the number of nonzero entries. We found that it ran significantly slower than the others we examined. This may reflect a flaw in our implementation strategy. It is also possible that our matrices are not large enough for the better asymptotic complexity of the sparse algorithm to take over.

The performance of the PRECISE-based algorithm depends on the determinant magnitude. From our data, PRECISE's performance appears to be independent of the condition number.

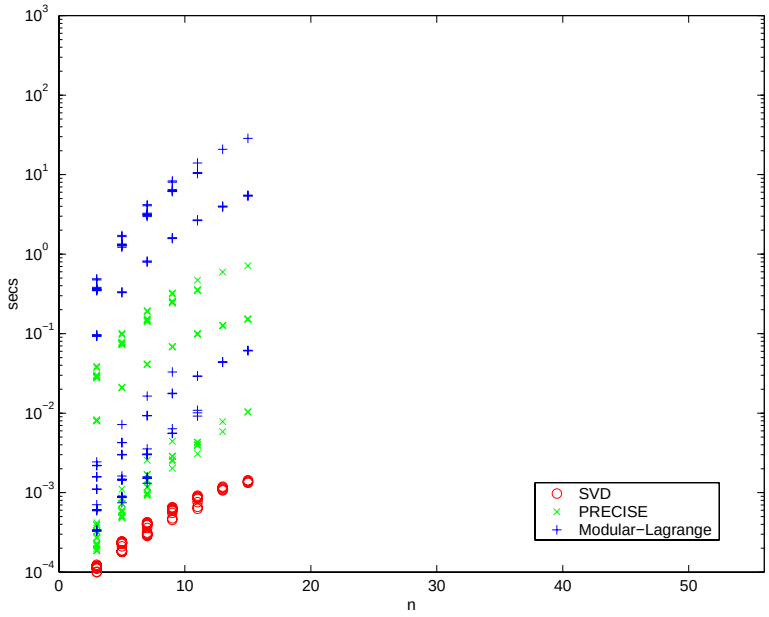

Fig. 4. Well-conditioned matrices. Each symbol represents a matrix for which the SVD filter succeeded. SVD is many times faster than PRECISE. PRECISE is also many times faster than modular arithmetic, though this is not expressed in the graph. The SVD filter fails on all matrices above $n = 15$. The symbols fall into curves, since the matrices are not evenly distributed with respect to entry bitlength.

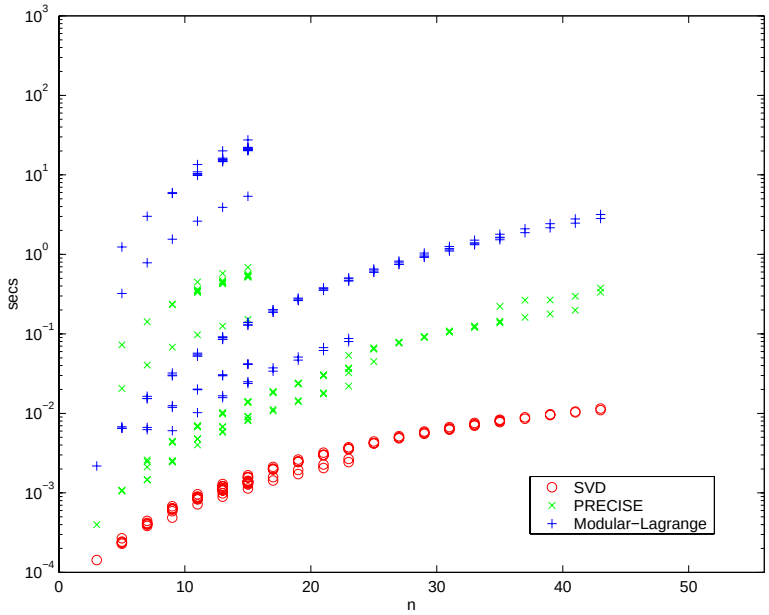

Fig. 5. Ill-conditioned, but nonsingular, matrices. Each symbol represents a matrix for which the SVD failed, but which turned out to be nonsingular. Since the SVD's sign is not dependable, we rely on the next fastest method, which is PRECISE in all cases. The SVD time is included on this graph to show that not much time is wasted on computing the SVD.

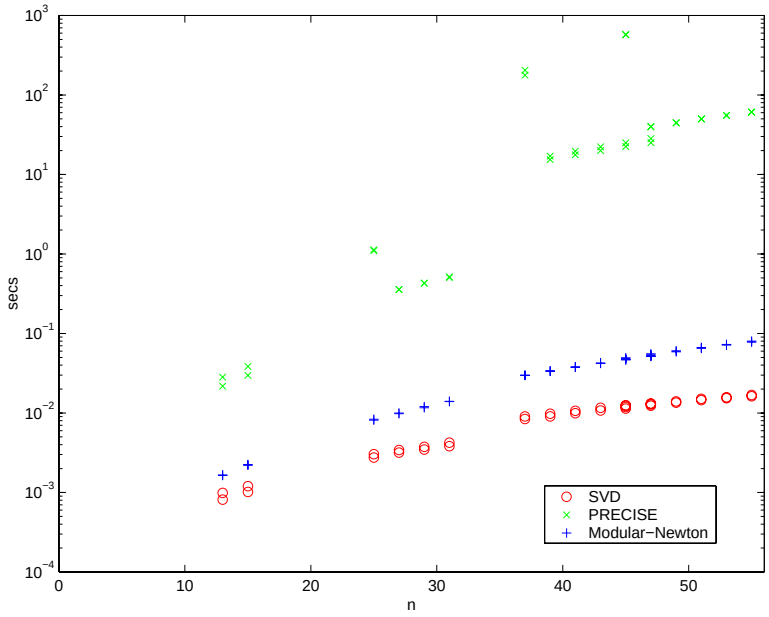

Fig. 6. Singular matrices. The SVD filter fails. For these matrices, our Newton implementation ceases after discovering that the determinant is zero modulo a single prime. As a consequence, modular arithmetic performs very well. PRECISE, on the other hand, performs poorly.

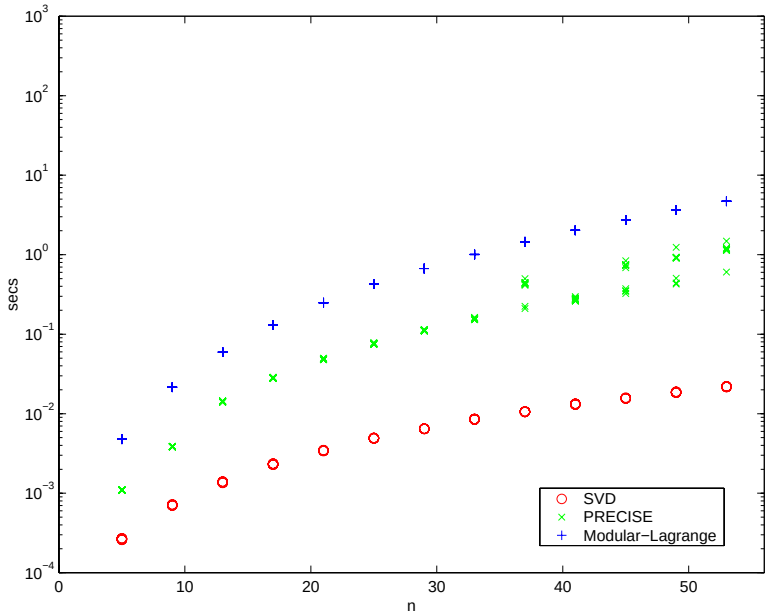

Fig. 7. Random matrices with uniformly-distributed 100-bit entries. The SVD filter succeeds on all of the matrices in this test.

8 Hybrid methods

The algorithms presented here perform differently on different matrix classes. The following table indicates the most efficient algorithm for each class. Note that in the geometric literature, the term "near-singular" usually refers to a matrix with a small determinant magnitude, while in the numerical computational literature, the same term refers either to an ill-conditioned matrix or a matrix with a tiny singular value σ_n. Both concepts are important, so we avoid the term "near-singular."

	Well-conditioned	Ill-conditioned but nonsingular	Singular
Large $\lvert \det \rvert$	SVD	PRECISE	—
Small $\lvert \det \rvert$		Modular-Newton	

Two modular reconstruction algorithms are not mentioned in the table. Lagrange reconstruction is useful when the tiny probability of failure inherent in early-exit Newton reconstruction is not tolerable, and the algorithm of Brönnimann et al. is useful for machine-precision matrices.

We propose a hybrid method, based on combining these algorithms in a flowchart. The design of the flowchart depends on the efficiency of the implementation of the various algorithms. Just as importantly, the design depends on the distribution of the matrices in the application at hand. For example, if singular matrices are extremely rare in practice, then modular arithmetic could be omitted. One should also consider whether the unlikely possibility that Newton reconstruction fails is tolerable. One fact that seems constant is that the SVD is by far the cheapest algorithm, and even when it fails, it provides enough information to suggest which of the other algorithms may be most efficient.

A flowchart for our application follows. It is based on several observations about our matrix distribution: singular matrices occur with some regularity, and nonsingular matrices with small determinants do not occur. Further, we are willing to tolerate the possibility that Newton reconstruction may fail.

(1) Compute the SVD in machine-precision floating point arithmetic, with relative precision ϵ.
(2) If the matrix is well-conditioned with respect to ϵ, stop.
(3) If the matrix is ill-conditioned with respect to ϵ, but well-conditioned with respect to a system-supported extended arithmetic with precision ϵ', compute the SVD in this arithmetic and stop.
(4) If the matrix is very ill-conditioned:
    (a) Guess whether the matrix is singular by comparing the smallest singular

value to a threshold: is $\sigma_n < T$? (The value of T affects only efficiency, not correctness.)

(b) If the matrix appears nonsingular, evaluate the sign using adaptive-precision arithmetic.

(c) If the matrix appears singular, compute the determinant in modular arithmetic with Newton reconstruction. (If the determinant is nonzero modulo the first prime, the matrix is nonsingular; switch to adaptive precision.)

9 Conclusions

We have presented a hybrid method for exactly computing the sign of the determinant of a moderate-sized integer matrix. The method begins by computing the singular value decomposition of the matrix. If the matrix is well-conditioned, the sign is known. Otherwise, information from the SVD is used to choose between a multiprecision interval algorithm and a modular arithmetic algorithm.

The determinant-sign filter based on the singular value decomposition is effective for well-conditioned matrices. For such matrices, the SVD reveals the sign at a fraction of the cost of an exact algorithm.

For ill-conditioned matrices, the SVD filter fails, but the cost of the filter remains small compared to the fallback algorithm. Further, the SVD reveals useful information about the matrix that can aid in choosing a reconstruction algorithm. An efficient multiprecision interval arithmetic system is effective on ill-conditioned but nonsingular matrices. Modular arithmetic is effective at verifying that a matrix is singular.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0*. SIAM, Philadelphia, 1992.

[2] Francis Avnaim, Jean-Daniel Boissonnat, Olivier Devillers, Franco P. Preparata, and Mariette Yvinec. Evaluating signs of determinants using single-precision arithmetic. Research Report 2306, INRIA, BP93, 06902 Sophia-Antipolis, France, 1994.

[3] Hervé Brönnimann, Ioannis Emiris, Victor Pan, and Sylvain Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 174–182, 1997.

[4] Hervé Brönnimann and Mariette Yvinec. Efficient exact evaluation of signs of determinants. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 166–173, 1997.

[5] C. Burnikel, S. Funke, and M. Seel. Exact geometric predicates using cascaded computation. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 175–183, 1998.

[6] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 387–395, October 1992.

[7] G. E. Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM*, 14:128–142, 1967.

[8] Tim Culver, John Keyser, and Dinesh Manocha. Accurate computation of the medial axis of a polyhedron. In *Proc. Symposium on Solid Modeling and Applications*, pages 179–190, 1999.

[9] James Demmel. Personal communication, 1998.

[10] James Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Stat. Comput.*, 11(5):873–912, September 1990.

[11] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, May 1993.

[12] S. Fortune and C. J. Van Wyk. Static analysis yields efficient exact integer arithmetic for computational geometry. *ACM Trans. Graph.*, 15(3):223–248, July 1996.

[13] G. H. Golub and C. F. van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, 2nd edition, 1989.

[14] LiDIA Group. A library for computational number theory. Technical report, TH Darmstadt, Fachbereich Informatik, Institut fur Theoretische Informatik, 1997.

[15] Vijay Karamcheti, Chen Li, Igor Pechtchanski, and Chee Yap. *The CORE Library Project*, 1.2 edition, 1999. http://www.cs.nyu.edu/exact/core/.

[16] John Keyser, Tim Culver, Dinesh Manocha, and Shankar Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. In *Proc. 15th Annual ACM Symposium on Computational Geometry*, pages 360–369, 1999.

[17] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[18] Shankar Krishnan, Mark Foskey, John Keyser, Tim Culver, and Dinesh Manocha. PRECISE: Efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. Technical Report TR00-008, University of North Carolina-Chapel Hill, 2000.

[19] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[20] P. S. Milne. On the solutions of a set of polynomial equations. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102, 1992.

[21] V. Y. Pan, Y. Yu, and C. Stewart. Algebraic and numerical techniques for the computation of matrix determinants. *Comput. Math. Appl.*, 34(1):43–70, 1997.

[22] Victor Y. Pan and Yanqiang Yu. Certified computation of the sign of a matrix determinant. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 715–724, 1999.

[23] P. Pedersen. Counting real zeros. Technical Report 545-R243, New York Univ., New York, NY, 1991.

[24] J. Shewchuk. Adaptive precision floating point arithmetic and fast robust geometric predicates. Technical Report CMU-CS-96-140, Carnegie Mellon Univ., Pittsburgh, PA, 1996.

[25] Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, IT-32(1), January 1986.

[26] C. Yap. Towards exact geometric computation. *Comput. Geom. Theory Appl.*, 7(1):3–23, 1997.