Exact and Efficient Determination of Geometric Predicates

IOANNIS Z. EMIRIS

INRIA Sophia-Antipolis, B.P. 93, 06902 Sophia-Antipolis, FRANCE, and
Department of Computer Science, University of Crete, Heraklio, GREECE.
emiris@sophia.inria.fr, http://www.inria.fr/saga/emiris

Abstract: - Typical geometric predicates, including orientation and in-sphere tests, reduce to a sign determination of multivariate polynomials. We propose an exact and efficient method that determines the sign of a multivariate polynomial expression with rational coefficients, including matrix determinants. Exactness is achieved by using modular computation. Although this usually requires some multiprecision computation, our techniques enable us to carry out sign determination by using only single precision. In this, and to exploit modern day hardware, we exclusively rely on floating point arithmetic. We show how our method can be used to generate robust and efficient implementations of geometric algorithms including convex hull, Voronoi diagram computations and solid modeling. This method is easy to implement and compares favorably with known methods from a practical complexity point of view. These claims are substantiated by experimental results and comparisons to other existing approaches.

Key Words: - Geometric predicate, sign determination, single precision, determinant test, fast implementation.

1 Introduction

In computational geometry, computer-aided design (CAD), geometric modeling and computer graphics, most geometric predicates can be expressed by computing the sign of an algebraic expression. We are typically interested in the exact determination of such a test with the additional goal of speed. The latter goal motivates us to use fixed-precision floating-point (f.p.) arithmetic since it has benefited from important infrastructural support and extremely efficient hardware implementations. F.p. arithmetic is only approximate, however. While this may be acceptable in performing numerically stable computations, it is unacceptable in deciding geometric predicates because the roundoff errors may easily lead to the wrong sign, causing the algorithm to fail. This problem is often referred to as the *robustness* problem [13].

This paper surveys methods that determine exactly the sign of a multivariate polynomial with rational coefficients evaluated at a rational point. It uses no operations other than modular arithmetic and f.p. computations with a fixed finite (single) precision. The key feature is the veracity of exact computation, combined with the efficiency of f.p. arithmetic. Our methods can be used in exact geometric predicates, as well as whenever numerical techniques need an exact test.

More specifically, our algorithms perform rational algebraic computations modulo several primes, which require only single-precision. The Chinese remainder theorem enables us to combine the resulting values together in order to recover the desired output value. This is not a new trick: such a representation of integers by their moduli is known as *Residue Number Systems* (RNS) and is popular because it provides a cheap and highly parallelizable version of multiprecision arithmetic. The latter stage of combining the moduli to reconstruct the explicit answer, however, has always been the bottleneck of this approach because higher precision computations were required [16]. Our method enables us to greatly accelerate this phase, since it only needs some simple single precision computations.

Most of the results surveyed here have appeared in [4, 5]. The closest predecessors of this work are apparently [10, 14, 3]. More recently, other approaches in the same general vein have appeared, such as [12].

2 Exact sign computation using modular arithmetic

Our model of a computer is that of an f.p. processor that performs operations at unit cost by using b-bit precision (e.g., in the IEEE 754 double precision standard, we have $b = 53$). For all four arithmetic operations, the computed result is always the f.p. representation that best approximates the exact result. This means that the relative error incurred by an operation returning x is at most 2^{-b-1}, and that the absolute error is at most $2^{\lfloor \log |x| - b - 1 \rfloor}$; logarithms in this paper are base 2. In particular, operations performed on pairs of integers smaller than 2^b are performed exactly as long as the result is also smaller than 2^b. We systematically ignore underflows and overflows, by assuming that the range of exponent is large enough. Given any real number x, it is *representable* over b bits if $x = 0$ or if $x2^{-\lfloor \log x \rfloor + b}$ is an integer; $\widetilde{x}$ denotes the representable f.p. number closest to x (with any tie-breaking rule if x is right in-between two representable numbers).

Let $m_1, \ldots, m_k$ be k pairwise relatively prime integers and let $m = \prod_i m_i$. For any number x (not necessarily an integer), we let $x_i = x \bmod m_i$ be the only number in the range $\left[-\frac{m_i}{2}, \frac{m_i}{2}\right)$ such that $x_i - x$ is a multiple of m_i. This operation can be extended modulo an f.p. number. Note that the result of truncating x to a power of two is always representable if x is representable. To perform arithmetic modulo m_i on integers by f.p. arithmetic, we assume $m_i \le 2^{b/2+1}$.

Problem 1 *Let k, b, $m_1, \ldots, m_k$ denote positive integers, $m_1, \ldots, m_k$ being pairwise relatively prime, such that $m_i \le 2^{b/2+1}$, and let $m = \prod_{i=1}^{k} m_i$. Let x be an integer whose magnitude is smaller than $\lfloor m/2 \rfloor$. Given $x_i = x \bmod m_i$, compute the sign of x by using only modular and floating-point arithmetic both performed with b-bit precision.*

We will solve this problem, even though x can be huge and, therefore, not even representable by using b bits. In the worst case, our solutions require $O(k^2)$ operations and therefore do not improve asymptotically over the standard multiprecision approach. They are simple, however, and require little or no overhead. In practice, they only perform $O(k)$ operations. Thus they are very well suited for implementation.

3 Lagrange's method

According to the Chinese remainder theorem, x is uniquely determined by its residues x_i, that is, Problem 1 is well defined and admits a unique solution. Moreover, this solution can be derived algorithmically from a formula due to Lagrange.

If x is an integer in the range $\left[-\frac{m}{2}, \frac{m}{2}\right)$, x_i stands for the residue $x \bmod m_i$, $v_i = m/m_i = \prod_{j \ne i} m_j$, and $w_i = v_i^{-1} \bmod m_i$, then

$$x = \left(\sum_{i=1}^{k} ((x_i w_i) \bmod m_i)\, v_i \right) \bmod m.$$

Trying to determine the sign of such an integer, we compute the latter sum approximately in fixed b-bit precision. Computing a linear combination of large integers v_i with its subsequent reduction modulo m can be difficult, so we prefer to compute the number

$$S = \frac{x}{m} = \operatorname{frac}\left(\sum_{i=1}^{k} \frac{(x_i w_i) \bmod m_i}{m_i} \right),$$

where frac(z) is the fractional part of a number x that belongs to $\left[-\frac{1}{2}, \frac{1}{2}\right)$.

If S were computed exactly, then we would have $S = x/m$, due to Lagrange's formula. If we compute S by incrementally adding the ith term and taking fractional part, the error bound follows the induction $\varepsilon_i = \varepsilon_{i-1} + 2^{-b-1} + 2^{-b}$, where the term 2^{-b-1} accounts for the error on computing the ith term of S, and the term 2^{-b} accounts for the error on computing the incremental sum. Moreover, $\varepsilon_1 = 2^{-b-1}$. A technical problem can arise if S is too close to a half-integer, because the fractional part may not be computed properly. We circumvent this by assuming that $|x|$ is less than $\frac{m}{2}(1 - \varepsilon_k)$. In this way, we can ensure that S approximates x/m within an absolute error bound $\varepsilon_k = (3k - 2)2^{-b-1}$.

Therefore, if $|S|$ is greater than ε_k, the sign of x is the same as the sign of S, and we are done. Otherwise, $|x| \le 2\varepsilon_k m$. Since $m_k \le 2^{b/2+1}$, we can say conservatively that $2\varepsilon_k m$ is smaller than $\frac{m}{2m_k}(1-\varepsilon_{k-1})$, for all practical values of k and b, and hence we may recover x already from $x_i = x \bmod m_i$ for $i = 1, \ldots, k-1$, that is, it suffices to repeat the computation using only $k-1$ moduli. Recursively, we will reduce the solution to the case of a single modulus m_1.

We will present our resulting algorithm by using additional notation: $m^{(j)} = \prod_{1 \leq i \leq j} m_i$, $v_i^{(j)} = \prod_{\substack{1 \leq \ell \leq j \\ \ell \neq i}} m_\ell$, $w_i^{(j)} = (v_i^{(j)})^{-1} \bmod m_i$,

$$S^{(j)} = \text{frac}\left(\sum_{i=1}^{j} x_i w_i^{(j)} \bmod m_i / m_i\right),$$

so that $m = m^{(k)}$, $v_i = v_i^{(k)}$, $w_i = w_i^{(k)}$ and $S = S^{(k)}$. We must assume that $x/m^{(k)}$ is sufficiently far from half-integers, hence we assume that $|x| \leq \frac{m^{(k)}}{2}(1-\varepsilon_k)$. This assumption is violated with very low probability ε_k for random x_i and can be remedied by computing one more residue x_{k+1}.

Algorithm 1 : Compute the sign of x knowing $x_i = x \bmod m_i$

```
Precomputed data: $m_j$, $w_i^{(j)}$, $\varepsilon_j$, for all $1 \leq i \leq j \leq k$
Input: integers $k$ and $x_i \in \left[-\frac{m_i}{2}, \frac{m_i}{2}\right)$, for all $1 \leq i \leq k$
Output: sign of $x$, the unique solution of $x_i = x \bmod m_i$ in $\left[-\frac{m^{(k)}}{2}, \frac{m^{(k)}}{2}\right)$
Precondition: $|x| \leq \frac{m^{(k)}}{2}(1-\varepsilon_k)$

1. Let $j \leftarrow k+1$
2. Repeat $j \leftarrow j-1$,
     $S^{(j)} \leftarrow frac\left(\sum_{i=1}^{j} \frac{x_i w_i^{(j)} \bmod m_i}{m_i}\right)$
   until $|S^{(j)}| > \varepsilon_j$ or $j = 0$
3. Return sign of $S^{(j)}$
```

Lemma 3.1 [5] *Algorithm 1 computes the sign of x knowing its residues x_i by using at most $\frac{k(k-1)}{2}$ modular multiplications, $\frac{k(k-1)}{2}$ f.p. divisions, $\frac{k(k-1)}{2}$ f.p. additions, and $k+2$ f.p. comparisons. All of these operations can be implemented in f.p. arithmetic.*

If $|x| > 2\epsilon_k m$ then only step k is performed, involving only at most k f.p. operations of each kind. This is to be contrasted with full reconstruction, which requires $\Theta(k^2)$ operations. Thus algorithm 1 is of great practical value.

If actually $x = 0$, the algorithm can be greatly sped up by testing if $x_j = 0$ in step 2, in which case we may directly pass to $j-1$. Furthermore, stage 3 is not needed unless $x = x_j = 0$ for all j, which can be tested beforehand.

4 Newton's method

An incremental version of Chinese remainder reconstruction, named after Newton, is described in this section. The main advantage is that it can be adapted to a probabilistic algorithm that does not require an *a priori* bound on the magnitude of x. This is the subject of section 4.1.

Let $x^{(j)} = x \bmod m^{(j)}$, for $j = 1, \ldots, k$, so that $x^{(1)} = x_1$ and $x = x^{(k)}$. Let $y_1 = x_1$, and for all $j = 2, \ldots, k$,

$$\begin{aligned} t_j &= w_j^{(j)} = (m^{(j-1)})^{-1} \bmod m_j, \\ y_j &= \left(x_j - x^{(j-1)}\right) t_j \bmod m_j \quad \in \left[-\frac{m_j}{2}, \frac{m_j}{2}\right). \end{aligned}$$

Then, for all $j = 2, \ldots, k$,

$$x^{(j)} = \left(x^{(j-1)} + y_j m^{(j-1)}\right) \bmod m^{(j)}. \qquad (1)$$

All computation can be kept modulo m_j, and no floating-point computation is required, in contrast to the previous section. The y_j define the mixed-radix representation of x, which would offer an alternative way to perform arithmetic on long integers; see also [15, 16]. It is obvious, that when $y_j \neq 0$, then the sign of $x^{(j)}$ is the same as the sign of y_j since $|x^{(j-1)}| \leq m^{(j-1)}/2$. If $y_j = 0$, the sign of $x^{(j)}$ is the same as that of $x^{(j-1)}$, for $j \geq 2$, whereas the sign of $x^{(1)} = x_1 = y_1$ is known. If $y_j = 0$ for all j, then this is precisely the case when $x = 0$.

For $1 \leq i < j \leq k$, we introduce integers $u_i^{(j-1)} =$

$$\left(m^{(i-1)} t_j\right) \bmod m_j = \left(\prod_{l=i}^{j-1} m_l\right)^{-1} \bmod m_j.$$

Then $t_j = u_1^{(j-1)}$. Unrolling equation (1) in the definition of y_j shows that the quantities y_j verify the following equality for all $j = 2, \dots, k$:

$$y_j = \left((x_j - x_1) u_1^{(j-1)} - \sum_{i=2}^{j-1} y_i u_i^{(j-1)} \right) \bmod m_j .$$

Therefore, they can be computed by using modular arithmetic with bit-precision given by the maximum bit-size of the m_j^2. Here it suffices to assume that the absolute value of x is bounded by $m^{(k)}/2$.

Algorithm 2 : Compute the sign of x, knowing $x \bmod m_i$, by Newton's incremental method.

```
Precomputed data: $m_j, u_i^{(j-1)}$, for all $1 \le i < j \le k$
Input: integers $k$ and $x_i \in \left[-\frac{m_i}{2}, \frac{m_i}{2}\right)$ for all $i = 1, \dots, k$
Output: sign of $x$, where $x$ is the unique solution of $x_i = x \bmod m_i$ in $\left[-\frac{m^{(k)}}{2}, \frac{m^{(k)}}{2}\right)$
Precondition: None.

1. Let $y_1 \leftarrow x_1$, $j \leftarrow 1$. Set $s$ to $-1, 0$ or $1$, if $y_1$ is negative, zero or positive, respectively.
2. Repeat $j \leftarrow j + 1$,
       $y_j \leftarrow \left( (x_j - x_1) u_1^{(j-1)} - \sum_{i=2}^{j-1} y_i u_i^{(j-1)} \right) \bmod m_j$,
       Set $s$ to $1$ or $-1$, if $y_j$ is positive or negative, respectively.
       If $y_j = 0$ then $s$ does not change.
   until $j = k$.
3. Return sign of $s$
```

Lemma 4.1 [5] *Algorithm 2 computes the sign of x knowing its residues x_i using exactly $\frac{k(k-1)}{2}$ modular multiplications, $\frac{k(k-1)}{2}$ modular additions, and $2k$ comparisons. All of these operations can be implemented in f.p. arithmetic.*

To compare with algorithm 1, realistically assume that a modular addition is equivalent to 3/2 f.p. additions and one comparison, on the average. Then, algorithm 1 requires at most $\frac{k(k-1)}{2}$ f.p. divisions (which are essentially multiplications with precomputed reciprocals) more than algorithm 2, whereas the latter always requires $\frac{k(k-1)}{4}$ extra f.p. additions and $\frac{k(k+1)}{2}$ additional comparisons.

4.1 A probabilistic variant

We propose below a probabilistic variant of algorithm 2 which, moreover, removes the need of an *a priori* knowledge of k. The principal feature of Newton's approach is its incremental nature. In our variant, this may lead to faster termination, before examining all k moduli. Informally, this should happen whenever the magnitude of x is significantly smaller than $m^{(k)}/2$, in which case we would save the computation required to obtain x_j for all larger j. This saves a significant amount of computation if termination occurs earlier than the static bound indicated by k.

Step 2 is modified to include a test of y_j against zero. Clearly, $y_j = 0$ precisely when $x^{(j)} = x^{(j-1)}$. Then we may deduce that $x^{(j)} = x^{(k)} = x$, with a very high probability, and terminate the iteration. In terms of mixed-radix representation, this assumes that when $y_j = 0$ then all more significant y_i's will also be zero. This is no different from escaping in multiprecision arithmetic when some digit (or sequence of consecutive digits) turns out to be zero, assuming then that the higher order digits also turn out to be zero.

By lemma 3.1 of [10], this algorithm terminates with a failure with probability bounded by $(k-2)/m_{\min}$, where $m_{\min} = \min\{m_1, m_2, \dots, m_k\}$.

5 Experimental results

5.1 Sign reconstruction

We present several benchmark results of our diverse methods for reconstructing the sign of an integer $x \in \left[-\frac{m^{(k)}}{2}, \frac{m^{(k)}}{2}\right)$ represented by its residues $x_i = x \bmod m_k$, $i = 1, \dots k$. Most of these are due to S. Pion at INRIA Sophia-Antipolis [4, 5, 6]. The measurements are performed on a 200MHz Sun Ultra Sparc workstation. We see for instance that they are negligible with those of the following determinant sign computation, showing that sign determination in RNS using our methods becomes a negligible portion of the determinant sign computation.

5.2 Determinant sign

We present several benchmark results of the described methods for computing the sign of a determinant and compare them with different existing packages, so as to assert the practical interest of our algorithms: Method FP is a straightforward f.p. implementation of Gaussian elimination which, of course, cannot guarantee correctness of the result. In particular, FP fails for ill-conditioned matrices. Method MOD is our implementation of modular Gaussian elimination. Method PROB is an implementation of modular Gaussian elimination using the probabilistic Newton variant described in section 4.1, where the computation is stopped when the probability of having a bad result is about 2^{-53}. In all the random matrices we tested, PROB never failed. Method CL has been implemented by us based on [9, 7]. As we compare with methods that handle arbitrary dimensions, we did not specialize the implementation for small dimensions as is done in [7] (this would provide an additional speedup of approximately 3). Method GMP is an implementation of Gaussian elimination using the GNU Multiprecision Package, for dimension lower than 5, and an implementation of Bareiss' extension of Gaussian elimination, for higher dimensions. Method LEDA uses the routine for the determinant sign of an integer matrix of Leda [8].

All implementations are in C, except LEDA which is in C++. Note that all methods could also be filtered, which would yield running times comparable to those of FP, on random inputs.

All tests were carried out on a 200MHz Sun Ultra Sparc workstation. Each program is compiled with the compiler that gives best results. Each entry in the following tables represents the average time of one run in microseconds, with a maximum deviation of about 10%. We concentrated on determinant sign evaluation and considered three classes of matrices: random matrices, whose determinant is typically away from zero, almost-singular matrices with single-precision determinant, and lastly singular matrices with zero determinant. The coefficients are integers of bit-size $53-n$ (due to restrictions of Clarkson's method).

Among the methods that guarantee exact computation, our implementations are at least as efficient as the others, and for certain classes of input they significantly outperform all available programs. Furthermore, our approach applies to arbitrary dimensions, whereas methods that compute a f.p. approximation of the determinant value are doomed to fail in dimensions higher than 15 because of overflow in the f.p. exponent. The running times are displayed in tables 1–3. (For small dimensions, specialized implementations can provide an additional speedup for all methods.) Our code is reasonably compact and easy to maintain.

Some side effects may occur, due to the way we generate matrices. The code of the modular package is free, and anyone can benchmark it on the kind of matrices that he uses. It is available via

http://www.inria.fr/prisme/personnel/pion/progs.

References

[1] F. Avnaim, J.-D. Boissonnat, O. Devillers, F. Preparata, and M. Yvinec, Evaluation of a new method to compute signs of determinants, in: Proc. 11th Annu. ACM Sympos. Comput. Geom. (1995) C16–C17.

[2] D. Bini, V. Y. Pan, Polynomial and Matrix Computations. Vol. 1: Fundamental Algorithms (Birkhäuser, Boston, 1994).

[3] J.-C. Bajard, L. S. Didier, J.-M. Muller, A new Euclidean division algorithm for residue number systems, in: Proceedings of ASAP (1996) 45–54.

[4] H. Brönnimann, I. Z. Emiris, V. Pan, and S. Pion, Computing exact geometric predicates using modular arithmetic with single precision, in: Proc. ACM Symp. on Computational Geometry (1997) 174–182.

[5] H. Brönnimann, I. Z. Emiris, V. Pan, and S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science, Special Issue on Real Numbers and Computers*, 210:1 (1999) 173–197.

[6] H. Brönnimann and S. Pion, Exact rounding for geometric constructions, in: Proc. of GAMM/IMACS Int. Symp. on Scientific Computing, Comput. Arithm. and Validated Numerics (1997) XIII-1–XIII-3.

[7] H. Brönnimann, M. Yvinec, Efficient exact evaluation of signs of determinants, Research Report 3140 (INRIA Sophia-Antipolis, 1997).

[8] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig, Exact geometric computation in LEDA, in: Proc. 11th Annu. ACM Sympos. Comput. Geom. (1995) C18–C19. Package available at: http://www.mpi-sb.mpg.de/LEDA/leda.html.

[9] K. L. Clarkson, Safe and effective determinant evaluation, in: Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci. (1992) 387–395.

[10] I. Z. Emiris, A complete implementation for computing general dimensional convex hulls, *Internat. J. Comput. Geom. & Appl.* 8:2 (1998) 223–253.

[11] I. Z. Emiris and J. F. Canny, A general approach to removing degeneracies, *SIAM J. Computing* 24:3 (1995) 650–664.

[12] I. Z. Emiris, V. Y. Pan, and Y. Yu, Modular arithmetic for linear algebra computations in the real field, *J. Symbolic Computation* 26:1 (1998) 71–87.

[13] C. M. Hoffmann, How solid is solid modeling? in: Applied Computational Geometry, Lecture Notes in Computer Science, Vol. 1148 (Springer, Berlin, 1996) 1–8.

[14] C. Y. Hung, B. Parhami, An approximate sign detection method for residue numbers and its application to RNS division, *Computers Math. Applic.* 27:4 (1994) 23–35.

[15] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, Vol. 2 (Addison-Wesley, Reading, Massachusetts, 1981 and 1997).

[16] N. S. Szabo, R. I. Tanaka, Residue arithmetic and its applications to computer technology (McGraw Hill, 1967).