American Journal of Computational Mathematics, 2014, 4, 464-473
Published Online December 2014 in SciRes. http://www.scirp.org/journal/ajcm
http://dx.doi.org/10.4236/ajcm.2014.45039

Scientific Research

Combining Algebraic and Numerical Techniques for Computing Matrix Determinant

Mohammad M. Tabanjeh

Department of Mathematics and Computer Science, Virginia State University, Petersburg, USA
Email: mtabanjeh@vsu.edu

Received 22 October 2014; revised 1 December 2014; accepted 9 December 2014

Abstract

Computing the sign of the determinant or the value of the determinant of an $n \times n$ matrix A is a classical well-know problem and it is a challenge for both numerical and algebraic methods. In this paper, we review, modify and combine various techniques of numerical linear algebra and rational algebraic computations (with no error) to achieve our main goal of decreasing the bit-precision for computing $\det A$ or its sign and enable us to obtain the solution with few arithmetic operations. In particular, we improved the precision $\lceil H\log_2 p \rceil$ bits of the p-adic lifting algorithm ($H = 2^h$ for a natural number h), which may exceed the computer precision β (see Section 5.2), to at most $\lceil \beta \rceil$ bits (see Section 6). The computational cost of the p-adic lifting can be performed in $O(hn^4)$. We reduced this cost to $O(n^3)$ by employing the faster p-adic lifting technique (see Section 5.3).

Keywords

Matrix Determinant, Sign of the Determinant, p-Adic Lifting, Modular Determinant, Matrix Factorization, Bit-Precision

1. Introduction

Computation of the sign of the determinant of a matrix and even the determinant itself is a challenge for both numerical and algebraic methods. That is, to testing whether $\det(A) > 0$, $\det(A) < 0$, or $\det A = 0$ for an $n \times n$ matrix A. In computational geometry, most decisions are based on the signs of the determinants. Among the geometric applications, in which the sign of the determinant needs to be evaluated, are the computations of con-

How to cite this paper: Tabanjeh, M.M. (2014) Combining Algebraic and Numerical Techniques for Computing Matrix Determinant. *American Journal of Computational Mathematics*, **4**, 464-473. http://dx.doi.org/10.4236/ajcm.2014.45039

vex hulls, Voronoi diagrams, testing whether the line intervals of a given family have nonempty common intersection, and finding the orientation of a high dimensional polyhedron. We refer the reader to [1]-[5], and to the bibliography therein for earlier work. These applications have motivated extensive algorithmic work on computing the value and particulary the sign of the determinant. One of the well-known numerical algorithms is Clarkson's algorithm. In [6], Clarkson uses an adaption of Gram-Schmidt procedure for computing an orthogonal basis and employs approximate arithmetics. His algorithm runs in time $O\left(d^3 b\right)$ and uses $2b+1.5d$ bits to represent the values for a $d \times d$ determinant with b-bit integer entries. On the other hand, the authors of [7] proposed a method for evaluating the signs of the determinant and bounding the precision in the case where $n \le 3$. Their algorithm asymptotic worst case is worse than that of Clarkson. However, it is simpler and uses respectively b and $(b+1)$-bit arithmetic. In this paper we examine two different approaches of computing the determinant of a matrix or testing the sign of the determinant; the numerical and the algebraic approach. In particular, numerical algorithms for computing various factorizations of a matrix A which include the orthogonal factorization $A=QR$ and the triangular factorizations $A=LU$, $A=P_rLU$, and $A=P_rLUP_c$ seem to be the most effective algorithms for computing the sign of $\det(A)$ provided that $|\det A|$ is large enough relative to the computer precision. Alternatively, the algebraic techniques for computing $\det A$ modulo an integer M based on the Chinese remainder theorem and the p-adic lifting for a prime p use lower precision or less arithmetic operations. In fact, the Chinese remainder theorem required less arithmetic operations than the p-adic lifting but with higher precision due to (9). We also demonstrate some effective approaches of combining algebraic and numerical techniques in order to decrease the precision of the computation of the p-adic lifting and to introduce an alternative technique to reduce the arithmetic operations. If $\det A$ is small, then obtaining the sign of the determinant with lower precision can be done by effective algebraic methods of Section 4. Although, the Chinese remainder algorithm approach requires low precision computations provided that the determinant is bounded from above by a fixed large number. This motivated us to generalize to the case of any input matrix A and to decrease the precision at the final stage of the Chinese remaindering at the price of a minimal increase in the arithmetic cost. Furthermore, in Section 5 we extend the work to an algorithm that computes $(\det A) \bmod M$ using low precision by relying on the p-adic lifting rather than the Chinese remaindering.

2. Definitions, Basic Facts, and Matrix Norms

Definition 1. *For an $n \times n$ matrix A, the determinant of A is given by either*

$$\det(A)=|A|=\sum_{j=1}^{n}(-1)^{i+j}a_{ij}\det(A_{ij}),\ \text{for } i=1,\cdots,n,\quad \text{or}\quad \det(A)=|A|=\sum_{i=1}^{n}(-1)^{i+j}a_{ij}\det(A_{ij}),\ \text{for } j=1,\cdots,n,$$

where A_{ij} is $(n-1)$-by-$(n-1)$ matrix obtained by deleting the i-th row and j-th column of A.

Fact 1. *Well-known properties of the determinant include* $\det(AB)=\det(A)\det(B)$, $\det(A^{\mathrm{T}})=\det(A)$, *and* $\det(cA)=c^n\det(A)$ *for any two matrices* $A,B\in\mathbb{R}^{n\times n}$ *and* $c\in\mathbb{R}$.

Definition 2. *If A is a triangular matrix of order n, then its determinant is the product of the entries on the main diagonal.*

Definition 3. *For a matrix A of size $m \times n$, we define* 1*-norm*: $\|A\|_1=\max_{1\le j\le n}\sum_{i=1}^{m}|a_{ij}|$, ∞ *-norm*: $\|A\|_\infty=\max_{1\le i\le m}\sum_{j=1}^{n}|a_{ij}|$, *p-norms*: $\|A\|_p=\sup_{x\ne 0}\dfrac{\|Ax\|_p}{\|x\|_p}=\sup_{x\ne 0}\left\|A\left(\dfrac{x}{\|x\|_p}\right)\right\|_p=\max_{\|x\|_p=1}\|Ax\|_p$, *and* 2*-norm*: $\|A\|_2=\max_{\|x\|_2\ne 0}\dfrac{\|Ax\|_2}{\|x\|_2}$.

3. Numerical Computations of Matrix Determinant

We find the following factorizations of an $n \times n$ matrix A whose entries are either integer, real, or rational by Gaussian elimination.

$$A=LU \tag{1}$$

$$A=P_rLU \tag{2}$$

$$A=P_rLUP_c. \tag{3}$$

We reduce A to its upper triangular form U. The matrix L is a unit lower triangular whose diagonal entries are

all 1 and the remaining entries of L can be filled with the negatives of the multiples used in the elementary operations. P_r is a permutation matrix results from row interchange and P_c is a permutation matrix results from column interchange.

Fact 2. *If P is a permutation matrix, then* $\det P = \pm 1$. *Moreover,* $\det I = 1$, *where I is an identity matrix.*

3.1. Triangular Factorizations

The following algorithm computes $\det(A)$ or its sign based on the factorizations (1)-(3).

Algorithm 1. *Triangular factorizations.*

Input: *An $n \times n$ matrix A.*

Output: $\det(A)$.

Computations:

1) *Compute the matrices L, U satisfying Equation* (1), *or P_r, L, U satisfying Equation* (2), *or P_r, L, U, P_c satisfying Equation* (3).

2) *Compute* $\det L$, $\det U$ *for Equation* (1), *or* $\det P_r$, $\det L$, $\det U$ *for Equation* (2), *or* $\det P_r$, $\det L$, $\det U$, $\det P_c$ *for Equation* (3).

3) *Compute and output* $\det A = (\det L)(\det U)$ *for Equation* (1), *or* $\det A = (\det P_r)(\det L)(\det U)$ *for Equation* (2), *or* $\det(A) = (\det P_r)(\det L)(\det U)(\det P_c)$ *for Equation* (3).

One can easily output the sign of $\det A$ from the above algorithm. We only need to compute the sign of $\det P_r$, $\det L$, $\det U$, and $\det P_c$ at stage 2 and multiply these signs. The value of $\det P_r$ and $\det P_c$ can be easily found by tracking the number of row or column interchanges in Gaussian elimination process which will be either 1 or -1 since those are permutation matrices. As L is a unit lower triangular matrix, $\det L = 1$. Finally, the computations of $\det U$ required $n-1$ multiplications since U is an upper triangular matrix. Therefore, the overall arithmetic operations of Algorithm 1 will be dominated by the computational cost at stage 1, that is, $\frac{n(n-1)(2n-1)}{6} = \sum_{i=1}^{n-1} i^2$ multiplications, the same number for additions, subtractions, and comparisons, but $\frac{n(n-1)}{2} = \sum_{i=1}^{n-1} i$ divisions. However, Algorithm 1 uses $\sum_{i=1}^{n-1} i$ comparisons to compute (2) rather than $\sum_{i=1}^{n-1} i^2$.

3.2. Orthogonal Factorization

Definition 4. *A square matrix Q is called an orthogonal matrix if* $Q^{\mathrm{T}}Q = I$. *Equivalently,* $Q^{-1} = Q^{\mathrm{T}}$, *where Q^{T} is the transpose of Q.*

Lemma 1. *If Q is an orthogonal matrix, then* $\det(Q) = \pm 1$.

Proof. Since Q is orthogonal, then $Q^{\mathrm{T}}Q = I$. Now $|Q^{\mathrm{T}}Q| = |I| = 1$ and thus $|Q||Q^{\mathrm{T}}| = 1$, but $|Q^{\mathrm{T}}| = |Q|$ which implies that $1 = |Q||Q| = |Q|^2$, hence $|Q| = \pm 1$.

Definition 5. *If* $A \in \mathbb{R}^{m \times n}$, *then there exists an orthogonal matrix* $Q \in \mathbb{R}^{m \times m}$ *and an upper triangular matrix* $R \in \mathbb{R}^{m \times n}$ *so that*

$$A = QR. \tag{4}$$

Algorithm 2. *QR-factorization.*

Input: *An $n \times n$ matrix A.*

Output: $\det(A)$.

Computations:

1) *Compute an orthogonal matrix Q satisfying the Equation* (4).

2) *Compute* $\det Q$.

3) *Compute the matrix* $R = Q^{\mathrm{T}}A = (r_{i,j})$.

4) *Compute* $\det(A) = (\det Q)\prod_i r_{i,i}$, *where* $r_{i,i}$ *are the main diagonal entries of R.*

Here, we consider two well-known effective algorithms for computing the QR factorization; the Householder and the Given algorithms. The Householder transformation (or reflection) is a matrix of the form $H = I - 2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$ where $\|\boldsymbol{v}\|_2 = 1$.

Lemma 2. *H is symmetric and orthogonal. That is,* $H^{\mathrm{T}} = H$ *and* $H^{\mathrm{T}} \cdot H = I$.

Proof. $H = I - 2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$, and since $I^{\mathrm{T}} = I$ and $(\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}})^{\mathrm{T}} = \boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}$, then $H^{\mathrm{T}} = I^{\mathrm{T}} - (2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}})^{\mathrm{T}} = I - 2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}} = H$. Hence,

H is symmetric. On the other hand, $H^{\mathrm{T}}H=\left(I-2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\right)\left(I-2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\right)=I-4\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}+4v\left(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}\right)\boldsymbol{v}^{\mathrm{T}}=I-4\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}+4\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}=I.$ Therefore, H is orthogonal, that is, $H^{-1}=H^{\mathrm{T}}$.

The Householder algorithm computes $Q=\prod_{i}^{n-1}H_i$ as a product of $n-1$ matrices of Householder transformations, and the upper triangular matrix $Q^{\mathrm{T}}A=R$.

Lemma 3. $\det\left(Q\right)=\left(-1\right)^{n-1}$ *for a matrix Q computed by the Householder algorithm.*

Proof. Since $H_i=I-2\boldsymbol{v}_i\boldsymbol{v}_i^{\mathrm{T}}$ for vectors $\boldsymbol{v}_i$, $i=1,\cdots,n-1$, Lemma 2 implies that $H_i^{\mathrm{T}}H_i=I$ and hence $\left(\det H_i\right)^2=1$. Notice that $\det\left(I-2\boldsymbol{e}_1\boldsymbol{e}_1^{\mathrm{T}}\right)=-1$ where $e_1=\left(1,0,\cdots,0\right)^{\mathrm{T}}$. Now for $0\le t\le 1$, we define the transformation $v\left(t\right)=t\left(v_i-e_1\right)+e_1$. Then the function $D\left(t\right)=\det\left(I-v\left(t\right)v\left(t\right)^{\mathrm{T}}\right)$ is continuous and $D^2=1\ \ \forall t$. Hence $D\left(1\right)=\det\left(I-2v_1v_1^{\mathrm{T}}\right)=-1$ since $\left\|\boldsymbol{v}\right\|_2=1$ and $D\left(0\right)=\det\left(I-2e_1e_1^{\mathrm{T}}\right)=-1$. This proves that $\det H_i=-1\ \ \forall i$. But $Q=\prod_{i=1}^{n-1}H_i$, we have $\det Q=\left(-1\right)^{n-1}$.

The Givens rotations can also be used to compute the *QR* factorization, where $Q=G_1\cdots G_t$, t is the total number of rotations, G_j is the the j-th rotation matrix, and $Q^{\mathrm{T}}A=R$ is an upper triangular matrix. Since the Givens algorithm computes Q as a product of the matrices of Givens rotations, then $\det\left(Q\right)=1$. We define $G=W_{i,i}\left(c,s\right)=W_{k,k}\left(c,s\right)=c$, $G=W_{i,k}=-W_{k,i}=s$ for two fixed integers i and k, and for a pair of nonzero real numbers c and s that satisfies $c^2+s^2=1$ such that $G^{\mathrm{T}}G=I$. If the Householder is used to find the *QR*, Algorithm 2 uses $n-1$ multiplications at stage 3. It involves $\frac{4}{3}n^3+O\left(n^2\right)$ multiplications at stage 1, and $O\left(n\right)$ evaluations of the square roots of positive numbers. If the Givens algorithm is used to find the *QR*, then it involves $2n^3+O\left(n^2\right)$ multiplications at stage 3, $n^3+O\left(n^2\right)$ multiplications at stage 1, and $O\left(n^2\right)$ evaluations. This shows some advantage of the Householder over the Givens algorithm. However, the Givens rotations algorithm allows us to update the *QR* of A successively by using only $O\left(kn^2\right)$ arithmetic operations and square root evaluations if $O\left(k\right)$ rows or columns of A are successively replaced by new ones. Therefore, in this case, the Givens algorithm is more effective.

4. Algebraic Computations of Matrix Determinant

4.1. Determinant Computation Based on the Chinese Remainder Theorem

Theorem 1. (*Chinese remainder theorem.*) *Let* $m_1,m_2,\cdots,m_k$ *be distinct pairwise relatively prime integers such that*

$$m_1>m_2>\cdots>m_k>1. \tag{5}$$

Let $M=\prod_{i=1}^{k}m_i$, and let D denote an integer satisfying the inequality

$$0\le D<M. \tag{6}$$

Let

$$r_i=D\left(\mathrm{mod}\ m_i\right). \tag{7}$$

$$M_i=\frac{M}{m_i},\ \ s_i\equiv M_i\left(\mathrm{mod}\ m_i\right),\ \ y_is_i\equiv 1\left(\mathrm{mod}\ m_i\right),\ \ \forall\, i=1,\cdots,k. \tag{8}$$

Then D is a unique integer satisfying (6) and (7). In addition,

$$D=\sum_{i=1}^{k}\left(M_ir_iy_i\right)\left(\mathrm{mod}\ M\right). \tag{9}$$

Algorithm 3. (*Computation of* $\det A\left(\mathrm{mod}\ M\right)$ *based on the Chinese remainder theorem.*)

Input: *An integer matrix A, an algorithm that computes* $\det A\left(\mathrm{mod}\ m\right)$ *for any fixed integer* $m>1$, k *integers* $m_1,\cdots,m_k$ *satisfying* (5) *and are pairwise relatively prime, and* $M=m_1m_2\cdots m_k$.

Output: $\det A\left(\mathrm{mod}\ M\right)$.

Computations:

1) *Compute* $r_i=\det A\left(\mathrm{mod}\ m_i\right)$, $i=1,\cdots,k$.

2) *Compute the integers* M_i, s_i, *and* y_i *as in* (8) $\forall\, i=1,\cdots,k$.

3) *Compute and output D as in* (9).

The values of y_i in (8) are computed by the Euclidean algorithm when applied to s_i and m_i for $i=1,\cdots,k$. Algorithm 3 uses $O(kn^3)$ arithmetic operations (ops) at stage 1, $O((k\log m_1)\log\log m_1)$ ops at stage 2, and $O(k\log^2 k)$ ops at stage 3. The computations of the algorithm are performed with precision of at most $\lceil \log_2 m_1 \rceil$ bits at stage 1, and at most $\lceil \log_2 M \rceil$ bits at stages 2 and 3. The latest precision can be decreased at the cost of slightly increasing the arithmetic operations [8].

Lemma 4. *For any pair of integers M and d such that* $M > 2|d|$, *we have*

$$d = (d \bmod M) \text{ if } (d \bmod M) < \frac{M}{2}, \text{ and } d = (d \bmod M) - M \text{ otherwise.} \tag{10}$$

Suppose that the input matrix A is filled with integers and an upper bound $d_u \geq |\det A|$ is known. Then we may choose an integer M such that $M > 2|d_u|$ and compute $\det A \pmod M$. Hence, $\det A$ can be recovered by using (10).

4.2. Modular Determinant

Let $A \in \mathbb{Z}^{n\times n}$ and $d = \det(A)$. In order to calculate $\det A \pmod p$, we choose a prime p that is bigger than $2|d|$ and perform Gaussian elimination (2) on $A \pmod p \in \mathbb{Z}_p^{n\times n}$. This is the same as Gaussian elimination over $\mathbb{Q}$, except that when dividing by a pivot element a we have to calculate its inverse modulo p by the extended Euclidean algorithm. This requires $\frac{2}{3}n^3 + O(n^2)$ arithmetic operations modulo p. On the other hand, if we need to compute $\det A \pmod p$ for the updated matrix A, we rely on the QR factorization such that $A = (QR) \pmod p$ and $Q^{\mathrm{T}}Q = D \pmod p$, where D is a diagonal matrix and R is a unit upper triangular matrix. The latest factorization can be computed by the Givens algorithm [9]. If the entries of the matrix A are much smaller than p, then we do not need to reduce modulo p the results at the initial steps of Gaussian elimination. That is, the computation can be performed in exact rational arithmetics using lower precision. In this case, one may apply the algorithm of [10] and [11] to keep the precision low. The computations modulo a fixed integer $M > 1$ can be performed with precision $\lceil \log_2 M \rceil$ bits. In such a computation, we do not need to worry about the growth of the intermediate values anymore. However, to reduce the cost of the computations, one can work with small primes modular instead of a large prime, that is, these primes can be chosen very small with logarithmic length. Then the resulting algorithm will have lower cost of $O(n^3 k)$ and can be performed in parallel. Now, one can find a bound on the $\det A$ without actually calculating the determinant. This bound is given by Hadamard's inequality which says that

$$|\det A| \leq n^{\frac{n}{2}} a^n = \left(a\sqrt{n}\right)^n, \tag{11}$$

where $a = \max_{1\leq i,j\leq n} |a_{ij}| \in \mathbb{Z}^+$ (nonnegative integers) is the maximal absolute value of an entry of A.

5. Alternative Approach for Computing Matrix Inverses and Determinant

5.1. *p*-Adic Lifting of Matrix Inverses

In this section, we present an alternative approach for computing matrix determinant to one we discussed in earlier sections. The main goal is to decrease the precision of algebraic computations with no rounding errors. The technique relies on Newton's-Hensel's lifting (p-adic lifting) and uses $O(n^4 \log n)$ arithmetic operations. However, we will also show how to modify this technique to use order of n^3 arithmetic operations by employing the faster p-adic lifting. Given an initial approximation to the inverse, say S_0, a well-known formula for Newton's iteration rapidly improves the initial approximation to the inverse of a nonsingular $n \times n$ matrix A:

$$S_{i+1} = S_i(2I - AS_i) = 2S_i - S_iAS_i = (2I - S_iA)S_i, \quad i \geq 0. \tag{12}$$

Algorithm 4. (*p-adic lifting of matrix inverses.*)
Input: *An n × n matrix A, an integer* $p > 1$, *the matrix* $S_0 = A^{-1} \pmod p$, *and a natural number h.*
Output: *The matrix* $A^{-1} \pmod{p^H}$, $H = 2^h$.
Computations:
1) *Recursively compute the matrix* S_j *by the equation,*

$$S_j = S_{j-1}\left(2I - AS_{j-1}\right)\left(\bmod p^J\right), \text{ where } J = 2^j,\ j = 1,\cdots,h. \tag{13}$$

2) *Finally, outputs* S_h.

Note that, $S_j = A^{-1}\left(\bmod p^J\right)$, $J = 2^j$, $\forall j$. This follows from the equation $I - AS_j = \left(I - AS_{j-1}\right)^2\left(\bmod p^J\right)$. The above algorithm uses $O\left(n^3\right)$ arithmetic operations performed modulo p^J at the j-th stage, that is, a total of $O\left(hn^3\right)$ arithmetic operations with precision of at most $\lceil J\log_2 p\rceil$ bits.

5.2. *p*-Adic Lifting of Matrix Determinant

We can further extend *p*-adic lifting of matrix inverses to *p*-adic lifting of matrix determinants, that is $\det A\left(\bmod p\right)$, using the following formula [12]:

$$\det A = \frac{1}{\prod_{k=1}^{n}\left(A_k^{-1}\right)_{k,k}}. \tag{14}$$

Here, A_k denotes the $k \times k$ leading principal (north-western) submatrix of A so that $A_n = A$ for $k = 1,\cdots,n$. Moreover, $\left(B\right)_{k,k}$ denotes the (k,k)-th entry of a matrix B. In order to use Formula (14), we must have the inverses of A_k available modulo a fixed prime integer p for all k. A nonsingular matrix A modulo p is called strongly nonsingular if the inverses of all submatrices A_k exist modulo p. In general, a matrix A is called strongly nonsingular if A is nonsingular and all $k \times k$ leading principle submatrices are nonsingular. Here, we assume that A is strongly nonsingular for a choice of p. Finally, Algorithm 4 can be extended to lifting $\det A$ (see Algorithm 5).

Algorithm 5. (*p-adic lifting of matrix determinant.*)

Input: *An integer* $p > 1$, *an* $n \times n$ *matrix A, the matrices* $S_{0,k} = A_k^{-1}\left(\bmod p\right)$, *and a natural number h.*

Output: $\det A\left(\bmod p^{2H}\right)$, $H = 2^h$.

Computations:

1) *Apply Algorithm* 4 *to all pairs of matrices* A_k *and* $S_{0,k}$, (*replacing the matrices A and* S_0 *in the algorithm*), *so as to compute the matrices* $S_{h,k} = A_k^{-1}\left(\bmod p^H\right)$, *for* $k = 1,\cdots,n$.

2) *Compute the value*

$$\left(\frac{1}{\det A}\right)\bmod p^{2H} = \prod_{k=1}^{n}\left[S_{h,k}\left(2I - A_kS_{h,k}\right)\right]_{k,k}\left(\bmod p^{2H}\right). \tag{15}$$

3) *Compute and output the value* $\left(\det A\right)\left(\bmod p^{2H}\right)$, *as the reciprocal of* $\left(\frac{1}{\det A}\right)\left(\bmod p^{2H}\right)$.

The overall computational cost of Algorithm 5 at stage 1 is $O\left(hn^4\right)$ arithmetic operations performed with precision at most $\lceil H\log_2 p\rceil$ bits. However, at stage 2, the algorithm uses $O\left(n^3\right)$ operations with precision $\lceil 2H\log_2 p\rceil$ bits. At stage 3, only one single multiplication is needed. All the above operations are calculated modulo p^{2H}. Now, we will estimate the value of H and p that must satisfy the bound $p^{2H} > 2\left|\det A\right|$. But, due to Hadamard's bound (11), we have $2\left|\det\left(A\right)\right| < 2\left(a\sqrt{n}\right)^n < p^{2H}$ which implies that $2H > n\log_p a + n\log_p\sqrt{n} + \log_p 2$. Therefore, it is suffices to choose p and H satisfying the inequalities $2\left(a\sqrt{n}\right)^n < p^{2H}$ and $2H > \log_p\left(2\left(a\sqrt{a}\right)^n\right)$. In the next section, we will present an alternative faster technique to computing matrix determinant that uses only $O\left(n^3\right)$ based on the divide-and-conquer algorithm.

Example 1. *Let* $A = \begin{pmatrix}1 & 17 & 18\\ 1 & 18 & 19\\ 5 & 16 & 20\end{pmatrix}$. *Then,* $A_1 = \left[1\right]$, $A_2 = \begin{pmatrix}1 & 17\\ 1 & 18\end{pmatrix}$, *and* $A_3 = \begin{pmatrix}1 & 17 & 18\\ 1 & 18 & 19\\ 5 & 16 & 20\end{pmatrix}$. *By Algorithm* 4, *compute the matrices*: $A_1^{-1} = \left[1\right]$, $A_2^{-1} = \begin{pmatrix}18 & -17\\ -1 & 1\end{pmatrix}$, *and* $A_3^{-1} = \begin{pmatrix}-56 & 52 & 1\\ -75 & 70 & 1\\ 74 & -69 & -1\end{pmatrix}$. *Now, we compute*

$S_{0,1} = A_1^{-1} \bmod 3 = [1]$, $S_{0,2} = A_2^{-1} \bmod 3 = \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix}$, *and* $S_{0,3} = A_3^{-1} \bmod 3 = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 2 & 0 & 2 \end{pmatrix}$. *We apply Algorithm* 4 (*that is*, $S_k = S_{0,k}\left(2I - A_k S_{0,k}\right) \bmod p^J$, $J = 2^j$, $j = 1, \cdots, h$) *to all pairs of matrices*:

$$S_{1,1} = S_1 = S_{0,1}\left(2I - A_1 S_{0,1}\right) \bmod 3^2 = [1],$$

$$S_{1,2} = S_2 = S_{0,2}\left(2I - A_2 S_{0,2}\right) \bmod 3^2 = \begin{pmatrix} 0 & 1 \\ 8 & 1 \end{pmatrix},$$

and

$$S_{1,3} = S_3 = S_{0,3}\left(2I - A_3 S_{0,3}\right) \bmod 3^2 = \begin{pmatrix} 7 & 7 & 1 \\ 6 & 7 & 1 \\ 2 & 3 & 8 \end{pmatrix}.$$

We then compute the value

$$\frac{1}{\det A} \bmod p^{2H} = \prod_{k=1}^{3} \left[S_{1,1}\left(2I - A_1 S_{1,1}\right)\right]_{1,1} \cdot \left[S_{1,2}\left(2I - A_2 S_{1,2}\right)\right]_{2,2} \cdot \left[S_{1,3}\left(2I - A_3 S_{1,3}\right)\right]_{3,3} \bmod 3^{2\times 2\times 1}$$

$$= [1]_{1,1} \cdot \begin{pmatrix} -144 & -17 \\ -1216 & -161 \end{pmatrix}_{2,2} \cdot \begin{pmatrix} -2243 & -2783 & -2510 \\ -2100 & -2603 & -2348 \\ -2113 & -2580 & -2269 \end{pmatrix}_{3,3}$$

$$= (1)(-161)(-2269) = 365309 \pmod{81} = 80.$$

Therefore, we output the value of $(\det A) \bmod 3^{2\times 2\times 1} = -1$, *since* $-1 \equiv 80 \bmod 81$.

5.3. Faster *p*-Adic Lifting of the Determinant

Assuming *A* is strongly nonsingular modulo *p*, block Gauss-Jordan elimination can be applied to the block 2 × 2 matrix $A = \begin{pmatrix} B & C \\ E & G \end{pmatrix}$ to obtain the well-known block factorization of *A*:

$$A = \begin{pmatrix} I & O \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & O \\ O & S \end{pmatrix} \begin{pmatrix} I & B^{-1}C \\ O & I \end{pmatrix},$$

where $S = G - EB^{-1}C$ is the Schur complement of *B* in *A*. The following is the divide-and-conquer algorithm for computing $\det(A)$.

Algorithm 6. (*Computing the determinant of a strongly nonsingular matrix.*)

Input: *An n × n strongly nonsingular matrix A.*

Output: $\det(A)$.

Computations:

1) *Partition A according to the block factorization above, where B is an* $\left\lfloor \frac{n}{2} \right\rfloor \times \left\lfloor \frac{n}{2} \right\rfloor$ *matrix.* ($\lfloor \cdot \rfloor$ *refers to the floor of* $\frac{n}{2}$.) *Compute* B^{-1} *and S using the matrix equation* $S = G - EB^{-1}C$.

2) *Compute* $\det(B)$ *and* $\det(S)$.

3) *Compute and output* $\det(A) = (\det B) \det S$.

Since *A* is strongly nonsingular modulo *p*, we can compute the inverse of *k* × *k* matrix by $O\left(k^3 \log H\right) = I(k)$ arithmetic operations using Algorithm 4. From the above factorization of *A*, we conclude that $D(n) \le D(\lfloor n/2 \rfloor) + D(\lceil n/2 \rceil) + I(\lfloor n/2 \rfloor)$. The computational cost of computing the determinant is $D(n) = O\left(n^3 \log H\right)$. This can be derived from the above factorization of *A* using recursive factorization applied to *B* and *S* and the inverses modulo p^{2H}. Here, $I(k)$ is the cost of computing the inverse and $D(k)$ is

the cost of computing the determinant.

6. Improving the Precision of the *p*-Adic Lifting Algorithm

Definition 6. *Let e be a fixed integer such that* $\log_2 e > \beta$ *where* β *is the computer precision. Then any integer z, such that* $0 \le z \le e-1$, *will fit the computer precision and will be called short integer or e-integer. The integers that exceed* $e-1=\alpha$ *will be called long integers and we will associate them with the e-polynomials* $p(x)=\sum_i p_i x^i$, $0 \le p_i < e$ *for all i.*

Recall that Algorithm 4 uses $\lceil j \log_2 p \rceil$ bits at stage *j*. For large *j*, this value is going to exceed β. In this section we decrease the precision of the *p*-adic algorithm and keep it below β. In order to do this, we choose the base $e = p^K$ where *K* is a power of 2, $2K \le H$, *K* divides $H/2$, and

$$K \le \log_2\left(\frac{\alpha^2 nH}{K}\right) < \beta. \tag{16}$$

Now, let us associate the entries of the matrices $A_k \bmod p^J$ and $S_{j-1,k} \bmod p^J$ with the *e*-polynomials in *x* for $J = 2^j$ and for all *j* and *k*. These polynomials have degrees $(J/K)-1$ and take values of the entries for $x = e$. Define the polynomial matrices $A_{j,k}(e) = A_k \bmod p^J$ and $S_{j,k}(e) = S_{j,k} \bmod p^J$. Then for $J \ge K$, we associate the *p*-adic lifting step of (13) with the matrix polynomial

$$S_{j,k}(x) = 2S_{j-1,n}(x) - S_{j-1,n}(x)A_{j,n}(x)S_{j-1,n}(x) \left(\bmod x^{J/K}\right). \tag{17}$$

The input polynomial matrices are $S_{j-1,n}(x)$ and $A_{j,n}(x)$ for $j = 1, 2, \cdots, h$. We perform the computations in (17) modulo $x^{J/K}$. The idea here is to apply *e*-reduction to all the entries of the output matrix polynomial followed by a new reduction modulo $x^{J/K}$. The resulting entries are just polynomials with integer coefficients ranging between $1-e$ and $e-1$. This is due to the recursive *e*-reductions and then taking modular reductions again. Note that the output entries are polynomials with coefficients in the range $-\gamma$ to γ for $\gamma \le 2^\beta$ even before applying the *e*-reductions. This shows that the β-bit precision is sufficient in all of the computations due to (16). The same argument can be applied to the matrices $S_{j,k}$ for all *j* and $k < n$.

7. Numerical Implementation of Matrix Determinant

In this section we show numerical implementation of the determinant of $n \times n$ matrices based on the triangular factorization *LU*, $P_r LU$, and $P_r LUP_c$. Algorithm 1 computes the triangular matrices $\tilde{L} = L + E_L$ and $\tilde{U} = U + E_U$, the permutation matrices $\tilde{P}_r$ and $\tilde{P}_c$, where E_L and E_U are the perturbations of *L* and *U*. In general, we can assume that $\tilde{P}_r = P_r$ and $\tilde{P}_c = P_c$ since the rounding error is small.

Definition 7. *Let*

$$A' = \tilde{P}_r^{-1} A \tilde{P}_c^{-1}, \ \tilde{L}\tilde{U} - A' = E', \tag{18}$$

and let e', a, $\tilde{l}$, *and* $\tilde{u}$ *denote the maximum absolute value of the entries of the matrices* E', A, $\tilde{L}$, *and* $\tilde{U}$. *Also,* E' *is the error matrix of the order of the roundoff in the entries of A.*

Assuming floating point arithmetic with double precision (64-bits) and round to β-bit. Then the upper bound on the magnitude of the relative rounding errors is $\epsilon = 2^{-\beta}$, where ϵ is the machine epsilon. Our goal is to estimate e'.

Theorem 2. ([13]) *For a matrix* $A = (a_{i,j})$, *let* $|A|$ *denote the matrix* $(|a_{i,j}|)$. *Then under* (18), $|E'| \le \left(|A'| + |\tilde{L}||\tilde{U}|\right)\epsilon$, *and*

$$e' \le e^+ = \left(a + n\tilde{l}\tilde{u}\right)\epsilon. \tag{19}$$

From the following matrix norm property $\left(\frac{1}{n}\right)\|A\|_q \le \max_{i,j}|a_{i,j}| \le \|A\|_q$, for $q = 1, 2, \infty$, we obtain the bound $e' \le \|E'\|_\infty$.

Theorem 3. *Let* a^+ *denotes the maximum absolute value of all entries of the matrices* $A^{(i)}$ *computed by Gaussian elimination, which reduces* A' *to the upper triangular form and let* ϵ *as defined above. Then*

$$e' \le e_+ = \|E'\|_\infty \le n^2 a^+ \epsilon < 1.8\epsilon a n^{2+(\ln^2)/4}. \tag{20}$$

By using the following results, we can estimate the magnitude of the perturbation of $\det A$ and $\det A'$ caused by the perturbation of A'. Here we assume that $P_r = P_c = I, A' = A$ and write,

$$e = e',\ e_d = \det(A+E) - \det A. \tag{21}$$

We use the following facts to estimate e_d of (21) in Lemma 5 below.

Fact 3. $|\det A| \le \|A\|_q^n$, $q = 1, 2, \infty$. $\|B\|_q \le \|A\|_q$, $q = 1, \infty$, *if* $|B| \le |A|$; $\|B\|_q \le \|A\|_q$, $q = 1, 2, \infty$, *if* B *is a submatrix of* A. $\|A+E\|_q \le \|A\|_q + ne$, *for* $q = 1, 2, \infty$; $\|A+E\|_2^2 \le \|A\|_2^2 + ne$.

Lemma 5. $|e_d| \le \left(\|A\|_q + ne\right)^{n-1} n^2 e$, for $q = 1, 2, \infty$.

Combining Lemma 5 with the bound (19) enables us to extend Algorithm 1 as follows:

Algorithm 7. (*Bounding determinant.*)

Input: *An* $n \times n$ *real matrix* A *and a positive* ϵ.

Output: *A pair of real numbers* d_- *and* d_+ *such that* $d_- \le \det A \le d_+$.

Computations:

1) *Apply Algorithm 1 in floating point arithmetic with unit roundoff bounded by* ϵ. *Let* $\tilde{U} = U + E_U$ *denote the computed upper triangular matrix, approximating the factor* U *of* A *from* (2) *or* (3).

2) *Compute the upper bound* e_+ *of* (20) *where* $e = e'$.

3) *Substitute* e_+ *for* e *and* $\min\left(\|A\|_1, \|A\|_\infty, \left(\|A\|_1 \|A\|_\infty\right)^{\frac{1}{2}}\right)$ *for* $\|A\|_q$ *in Lemma* 5 *and obtain an upper bound* e_d^+ *on* $|e_d|$.

4) *Output the values* $d_- = \det\tilde{U} - e_d^+$ *and* $d_+ = \det\tilde{U} + e_d^+$.

Example 2. *Let* $A = \begin{pmatrix} 0 & \frac{5}{3} & -\frac{31}{6} \\ \frac{19}{4} & \frac{3}{2} & \frac{16}{5} \\ \frac{17}{5} & \frac{21}{4} & \frac{12}{9} \end{pmatrix} \approx \begin{pmatrix} 0 & 1.6667 & -5.1667 \\ 4.7500 & 1.5000 & 3.2000 \\ 3.4000 & 5.2500 & 1.3333 \end{pmatrix}$. *Using Matlab, Gaussian elimination with complete pivoting gives the matrix* U *rounded to* 4 *bits*: $U = \begin{pmatrix} 5.2500 & 1.3333 & 3.4000 \\ 0 & -5.5899 & -1.0794 \\ 0 & 0 & 3.2342 \end{pmatrix}$,

$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.3175 & 1 & 0 \\ 0.2857 & -0.5043 & 1 \end{pmatrix}$, $\tilde{U} = U + E_U$, $\tilde{L} = L + E_L$ *where* E_U *and* E_L *are the matrices obtained from accumulation of rounding errors. Also* $P_r = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, *and* $P_c = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. *Then* $E_{A'} = \begin{pmatrix} 0 & 6\times10^{-5} & 6\times10^{-5} \\ 0 & 0 & 0 \\ 0 & 0 & 3\times10^{-5} \end{pmatrix}$ *is a perturbation in* A, *and* $\|E_{A'}\|_\infty = 1.2\times10^{-4}$. *We now compute the upper bound* e_+ *of* (20). *Therefore,* $e' \le e_+ \le 62.9618$, *where* $a = 5.25$ *is the maximum of* $|a_{i,j}|$ *of* A, $l = 1$ *is the maximum of* $|l_{i,j}|$ *of* L, $\epsilon = 2^{-\beta}$, $\beta = 4$, *and* $n = 3$ *is the size of the input matrix* A. *Hence we obtain the following upper bound* e_d^+ *on* $|e_d|$ *by Lemma* 5. *That is,* $|e_d| \le \left(\min\left(\|A\|_1, \|A\|_\infty, \left(\|A\|_1 \|A\|_\infty\right)^{\frac{1}{2}}\right) + ne_+\right)^{n-1} n^2 e_+ = 2.2347\times10^7$, *where* $\|A\|_1 = 9.7$, $\|A\|_\infty = 9.9833$, *and* $\sqrt{\|A\|_1 \|A\|_\infty} = 9.8406$. *Hence,* $e_d^+ = 2.2347\times10^7$ *is an upper bound of* $|e_d|$.

$d_- = \det\tilde{U} - e_d^+ = -2.2347\times10^7$ *and* $d_+ = \det\tilde{U} + e_d^+ = 2.2347\times10^7$. *Since* $\det(A) = -94.91597$, *we have*

$\det\tilde{U} - e_d^+ \le \det(A) \le \tilde{U} + e_d^+$. *On the other hand* $e_d = \det(A+EA) - \det(A) = 0.00123$ *which is less than the upper bound* 2.2347×10^7 *we have computed.*

Acknowledgements

We thank the editor and the referees for their valuable comments.

References

[1] Muir, T. (1960) The Theory of Determinants in Historical Order of Development. Vol. I-IV, Dover, New York.

[2] Fortune, S. (1992) Voronoi Diagrams and Delaunay Triangulations. In: Du, D.A. and Hwang, F.K., Eds., *Computing in Euclidean Geometry*, World Scientific Publishing Co., Singapore City, 193-233.

[3] Brönnimann, H., Emiris, I.Z., Pan, V.Y. and Pion, S. (1997) Computing Exact Geometric Predicates Using Modular Arithmetic. *Proceedings of the* 13*th Annual ACM Symposium on Computational Geometry*, 174-182.

[4] Brönnimann, H. and Yvinec, M. (2000) Efficient Exact Evaluation of Signs of Determinant. *Algorithmica*, **27**, 21-56. http://dx.doi.org/10.1007/s004530010003

[5] Pan, V.Y. and Yu, Y. (2001) Certification of Numerical Computation of the Sign of the Determinant of a Matrix. *Algorithmica*, **30**, 708-724. http://dx.doi.org/10.1007/s00453-001-0032-8

[6] Clarkson, K.L. (1992) Safe and Effective Determinant Evaluation. *Proceedings of the* 33*rd Annual IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 387-395.

[7] Avnaim, F., Boissonnat, J., Devillers, O., Preparata, F.P. and Yvinec, M. (1997) Evaluating Signs of Determinants Using Single-Precision Arithmetic. *Algorithmica*, **17**, 111-132. http://dx.doi.org/10.1007/BF02522822

[8] Pan, V., Yu, Y. and Stewart, C. (1997) Algebraic and Numerical Techniques for Computation of Matrix Determinants. *Computers & Mathematics with Applications*, **34**, 43-70. http://dx.doi.org/10.1016/S0898-1221(97)00097-7

[9] Golub, G.H. and Van Loan, C.F. (1996) Matrix Computations. 3rd Edition, John Hopkins University Press, Baltimore.

[10] Edmonds, J. (1967) Systems of Distinct Representatives and Linear Algebra. *Journal of Research of the National Bureau of Standards*, **71**, 241-245. http://dx.doi.org/10.6028/jres.071B.033

[11] Bareiss, E.H. (1969) Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. *Mathematics of Computation*, **22**, 565-578.

[12] Bini, D. and Pan, V.Y. (1994) Polynomial and Matrix Computations, Fundamental Algorithms. Vol. 1, Birkhäuser, Boston. http://dx.doi.org/10.1007/978-1-4612-0265-3

[13] Conte, S.D. and de Boor, C. (1980) Elementary Numerical Analysis: An Algorithm Approach. McGraw-Hill, New York.