Pergamon

Computers Math. Applic. Vol. 34, No. 1, pp. 43–70, 1997
Copyright©1997 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0898-1221/97 $17.00 + 0.00

PII: S0898-1221(97)00097-7

Algebraic and Numerical Techniques for the Computation of Matrix Determinants

V. Y. PAN*
Department of Mathematics and Computer Science
Lehman College, City University of New York
Bronx, NY 10468, U.S.A.
vpan@lcvax.lehman.cuny.edu

Y. YU* AND C. STEWART*
Graduate Center of City University of New York
New York, NY 10038, U.S.A.

(Received November 1996; accepted December 1996)

Abstract—We review, modify, and combine together several numerical and algebraic techniques in order to compute the determinant of a matrix or the sign of such a determinant. The resulting algorithms enable us to obtain the solution by using a lower precision of computations and relatively few arithmetic operations. The problem has important applications to computational geometry.

Keywords—Evaluation of the determinant, Sign of the determinant, Matrix singularity test, Modular (residue) arithmetic, Rounding error analysis.

1. INTRODUCTION

1.1. The Subject and Some Background

We study the classical problems of the computation of the determinant of a matrix or testing if the determinant vanishes, that is, if the matrix is singular. These problems have a long history (see, for instance, [1–11]) and have recently received a new major motivation, due to their important applications to geometric computations, such as computation of convex hulls and Voronoi diagrams, and testing if the line intervals of a given family have a nonempty common intersection. In such applications, one needs *sign* or *singularity* tests, that is, one needs either to test if $\det A > 0$, $\det A = 0$, or $\det A < 0$, for an $n \times n$ matrix A, or just to test whether $\det A = 0$ or not. In one group of these applications, n is *relatively small* [12,13], ranging from 2 to 10, but

*Supported by NSF Grants CCR 9020690 and CCR 9625344, and by PSC CUNY Awards 665301, 666327, and 667340.
The importance of the problems studied in this paper was brought to my attention by K. Clarkson and F. Preparata. Clarkson and J. Hobby also described the customary requirements to the input matrices encountered in the applications to computational geometry, and Clarkson made several useful comments to the first draft of this paper. B. Braams and R. Pollack gave me some useful information about the computational geometry applications in the case of matrices of larger sizes. The discussions with H. Brönnimann, and more briefly, with O. Devillers on implementation and applications of determinant sign algorithms and Brönnimann comments on the works [12,13] were very informative.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

mostly staying below 5, and the entries of A represent high precision approximations to real data. In this class of applications, computations are usually performed with the 64-bit double precision (where either floating point arithmetic with about 50 bits allocated to the mantissas is used or fixed point representation with the double precision of 64 bits is used), though [13] also proposes a single precision approach. In another major group of geometric applications [14–20], one needs to know orientation of a high-dimensional polyhedron or of a high-dimensional algebraic variety (say, for the problems of convex optimization in statistical physics and chemistry). Then the range for n is on a much higher level, say, from 100 to 500, whereas the entries of the input matrix A can be represented with a lower precision of 5 to 10 bits. In both of these groups of applications, the matrix A can be either fixed or updated dynamically, so that every time only one of its columns (or one of its rows) is changing.

In this paper, we study some effective approaches to the sign and singularity testing and to the determinant computation. We view this classical and modern topic as a good occasion for demonstrating the power of combining algebraic and numerical computational techniques.

Let us first recall some previous work. If $n = 2$ or $n = 3$, one may try to compute $\det A$ (so as to output its sign) by using no division and based on its decomposition $\det A = \sum_{j=0}^{n-1} a_{0,j} (-1)^{j-1} A_{0,j}$, where $A_{0,j}$ is the minor of $A = (a_{i,j})$ complementing the entry $a_{0,j}$. In this case, however, one either risks losing the correct answer due to rounding errors or one generally needs to double or triple the input precision, which can be prohibitive for some computer implementations. In this and other cases, one may avoid involving higher precision values by performing elementary transformations of A (that is, by interchanging its rows and/or columns and by replacing a row or a column by its linear combination with other rows or columns of A). Successful work in this direction can be traced back to [6–8], where the computations were performed in rational arithmetic, with no roundoff errors. The recent paper [13] shows how to bound the precision of computations more effectively in the case where $n \leq 3$ (the method remains fairly effective for $n = 4$) and where the objective is not the computation of $\det A$ but only of its sign. The algorithm of Clarkson [12] competes with one of [13] for $n \leq 4$ and supersedes it for larger n. It relies on the application of the modified Gram-Schmidt algorithm for the QR (orthogonal) factorization of A, which is known to lead to quite effective approximate solution and which Clarkson has extended to the provably correct computation of the sign of the determinant. As an example of distinct approaches, we will cite the algorithms that test some classes of geometric degeneracies by relying on small (linear) perturbations of the input parameters (cf. [21]), and we refer the reader to [22], for further information and bibliography.

A simple sketchy argument [23] suggests that, under some restricted model of computing, even the singularity test (verifying whether $\det A = 0$) has the same asymptotic arithmetic complexity as the evaluation of $\det A$. Furthermore, under the straight-line program model (with no branchings), the evaluation of $\det A$ and A^{-1} has the same asymptotic arithmetic complexity. (The proof of the latter property relies on the partial derivative theorem [24,25].) These considerations suggest that decreasing the arithmetic cost of computing the sign of $\det A$ is quite a hard problem.

1.2. Our Study. Numerical and Algebraic Approaches

Our main goal in the present paper is the easier task of decreasing the bit-precision required in algorithms for computing $\det A$ or its sign (without blowing up the arithmetic complexity bounds), and such a decrease is highly important for practical applications to computational geometry.

We do not confine our study to a single algorithm but consider a broad range of various techniques of numerical linear algebra and rational algebraic computations (with no errors).

In particular, numerical algorithms for computing various factorizations of the input matrix A (that is, its orthogonal (QR) and its triangular ($PLUP_1$, PLU, and LU) factorizations) seem to be the most effective basis for computing the sign of $\det A$ provided that $|\det A|$ is large enough, relatively to computer precision. Indeed, the sign of $\det A$ can be immediately read off from the signs of the diagonal entries of the computed triangular matrices, that is, either R or L and U, respectively. To verify if this output is correct, we estimate the output error caused by the accumulation of the rounding errors of our floating point computation with finite precision. This enables us to obtain both the product d of the diagonal entries of R or L and U, and a positive e such that $\det A$ is bracketed in the interval $d - e \leq \det A \leq d + e$. Generally, for numerical computations, such goals can be achieved by means of interval analysis (see, e.g., [26]), but in our case, Wilkinson's techniques of backward error analysis and their extensions give us smaller bounds e; moreover, we analyze various algorithms for factorization of A, so as to be able to decrease e.

If $e < |d|$, then d and $\det A$ have the same sign, and our main problem of the sign computation is solved. It remains to compute the sign of $\det A$ in the case where $e \geq |d|$, that is, where

$$|\det A| < |d| + e \leq 2e. \tag{1.1}$$

According to the known estimates, based on the backward error analysis and recalled in our Sections 4 and 6, e is proportional to the upper bound $\epsilon = 2^{-\beta}$ on the magnitudes of the relative rounding errors, where β denotes the computer precision. Thus, if the inequalities (1.1) hold, $|\det A|$ tends to be substantially smaller than its *a priori* upper bound $\|A\|^n$. (The numerical algorithms for computing $\det A$ or its sign, the analysis of these algorithms, and the error estimates are the subjects of Part I, Sections 3–6.)

Now, if $|\det A|$ is not large, then our problem of obtaining the sign of $\det A$ by using computations with a lower precision is more easily treated by means of algebraic methods, which we study in Part II, Sections 7–15. In principle, the latter methods can be applied based on any upper bound on $|\det A|$ but become more effective in the cases where $|\det A|$ is smaller, thus complementing the numerical approach. (The algebraic methods are applied to integer matrices; a real matrix can be turned into an integer matrix via its scaling after rounding-off its entries.) Indeed, if the computed value d, approximating $\det A$ numerically, within an absolute error bound e, turns out to have a relatively small magnitude $|d|$, then it suffices to compute $(\det A) \bmod M$, for any integer M exceeding $2(e + |d|)$ (see Section 7, Fact 7.1). This only requires computations with a precision of $\lceil \log_2 M \rceil$ or fewer bits.

Furthermore, assuming that an upper bound M on $2|\det A| + 1$ is available, we study two customary algebraic approaches to computing $(\det A) \bmod M$; both of them are performed with a much lower precision than $\lceil \log_2 M \rceil$, except for their final stages, involving fewer arithmetic operations. Actually, our approach based on the Chinese remainder theorem and presented in Sections 7–11, requires lower precision computations even at the final stage, provided that our task is restricted either to testing matrix singularity (see Section 8) or, more generally, to determining whether $|\det A|$ is bounded from above by a fixed moderately large number, and if so, to computing $\det A$ (see Section 9). Moreover, even in the general case of any input matrix A, we decrease the precision of computing at the final stage, at the price of a respective moderate increase of the arithmetic cost (see Section 11, Theorem 11.1). This is achieved by means of replacing arithmetic operations with "long" integers by the same operations with associated polynomials having "short" integer coefficients; such a process reverses the known method of binary segmentation (see, e.g., [11, pp. 277–279; 27]). In Section 11, we show how to make all operations with associated polynomials "linear"; the class of such operations, besides additions and subtractions, only includes multiplications and divisions by "short" integers, rather than by polynomials. By avoiding more complicated codes and subroutines for dealing with nonlinear multiprecision arithmetic, we keep our algorithms for the sign of $\det A$ more readily accessible for practical implementation. In Sections 12–15, we specify algorithms that compute $(\det A) \bmod M$ by using a low precision of computing and by relying on p-adic (Newton-Hensel's) lifting, rather than on the Chinese remainder theorem. For matrices A having larger norms, this approach generally involves multiplications of pairs of "longer" integers or of their associated nonconstant polynomials, besides linear operations. Among the possible alternative algorithms, we briefly examine (in Section 16) the straightforward algorithms ameliorated by means of our approach of Section 11; then again only linear operations with the associated polynomials are involved in this case. The resulting amelioration may make the algorithms practically competitive for small n.

1.3. Further Comments

Our techniques of Sections 11, 13, and 14 for decreasing the precision required in the Chinese remainder and p-adic computations, and also our refinement of the estimates of [28,29] for the probability that $(\det A) \bmod p$ vanishes for a random prime p provided that $\det A \neq 0$ (cf. Appendix A) may be of some independent interest for the designers of algebraic algorithms.

The power of the proposed techniques can be accentuated in their combination with other known ones. In particular, the techniques of [12] can be easily adjusted to the task of numerical factorization, which we face in Sections 3–6, though we prefer to use Gaussian elimination with pivoting, the Householder and/or Givens algorithms to using the modified Gram-Schmidt algorithm (see Remark 4.2 of Section 4, and Remark 6.4 of Section 6); the techniques of [13] naturally complement ours (in the case of smaller n) as an additional means of filtering off the case of absolutely large determinants, and a substantial acceleration of numerical factorization of A and of algebraic computation of $(\det A) \bmod M$ can be achieved by means of parallel processing. (See Remark 10.1 in Section 10 on parallel acceleration of algebraic algorithms for matrix factorizations, see [30,31] on parallelization of numerical factorization algorithms, and observe that at least the first two stages of each of our Algorithms 7.1, 8.1, and 9.1 can be performed concurrently in i, for $i = 1, \dots, k$, by using k processors.)

Presenting the estimates for the complexity of matrix factorization and of performing our other algorithms, we show these estimates either exactly or in the form $cn^3 + O(n^2)$ or $cn^3 + O(n^2 \log n)$, where we specify the constant c. (This is more precise than, for instance, in [12,13], where such bounds have been given in the form $O(n^3)$.) The form $cn^3 + O(n^2)$, with a specified c is customary in the field of matrix computations [30], and we have ignored the possible theoretical improvement of these estimates based on fast matrix multiplication (cf. [11,30]). We have estimated the sequential arithmetic cost of the presented tests and algorithms, but for several of them, effective parallel implementations are well known [11,30,31]. For those applications to computational geometry where n is small, the asymptotic computational complexity bounds are only partially informative, however, and the decisive comparison of the proposed algorithms with each other should rely on experimental tests.

For convenience, we will formally study the evaluation of $\det A$, even though we actually care most about computing its sign, due to the major applications to computational geometry.

1.4. Organization of the Paper

After some preliminaries of Section 2, we present and analyze numerical algorithms for approximating $\det A$ based on numerical factorizations of A (see Sections 3–6). In Sections 7–9, we describe the algebraic approach based on the Chinese remainder theorem, and in Sections 12–15, we rely on p-adic (Newton-Hensel's) lifting. In Section 10, we describe the auxiliary computation of $\det A$ modulo a fixed prime, required in Sections 7–9, 12, and 15. We show our techniques for decreasing the precision of the Chinese remainder computations in Section 11 and of p-adic computations in Sections 13 and 14. In Section 16, we examine the lower precision modifications of the straightforward algorithms (for smaller n). Appendix A contains some estimates for the probability that reduction modulo a prime chosen at random in a fixed interval makes the determinant of a nonsingular integer matrix vanish. Parts I and II and Appendix A are due to the first author and represent his 1996 revision of his unpublished report of 1994 "Combining

Algebraic, Numerical and Randomization Techniques for the Determinant Computation." In Appendix B, the three authors present their numerical experiments for Part I. Improveda;gebraic and numerical solution algorithms can be found in [32] and in the authors' paper with D. Bini (in preparation).

PART I. NUMERICAL COMPUTATION APPROACH

2. BASIC MATRIX FACTORIZATIONS

$\det A$ and its sign for an $n \times n$ integer, rational or real input matrix A can be immediately obtained from the factorizations:

$$A = LU, \tag{2.1}$$

$$A = PLU, \tag{2.2}$$

$$A = PLUP_1, \tag{2.3}$$

$$A = QR, \tag{2.4}$$

where Q is a real $n \times n$ orthogonal (unitary) matrix, such that

$$Q^{\top}Q = I, \tag{2.5}$$

$L^{\top}$, U, and R are $n \times n$ upper triangular matrices, and P and P_1 are permutation matrices, whose application to any vector v amounts to some fixed permutations of its coordinates. Here and hereafter, $W^{\top}$ denotes the transpose of a matrix or a vector W, and I and O denote the identity matrices, $Iv = v$ for any vector v, and the null matrices of appropriate sizes. The computation of $\det A$ based on (2.2)–(2.5) relies on the following well-known facts.

FACT 2.1. $\det(SV) = (\det S)\det V$, *for any pair of $n \times n$ matrices S and V.*

FACT 2.2. $\det T = \prod_{i=0}^{n-1} t_{i,i}$, *for any $n \times n$ triangular matrix $T = (t_{i,j},\ i,j = 0,\dots,n-1)$.*

FACT 2.3. $\det P = 1$ *or* $\det P = -1$, *for any permutation matrix P;* $\det I = 1$.

FACT 2.4. $\det W = \det W^{\top}$, *for any square matrix W.*

COROLLARY 2.1. $\det Q = 1$ *or* $\det Q = -1$, *for any real square matrix Q satisfying (2.5).*

3. SOLUTION BASED ON GAUSSIAN ELIMINATION

Based on the equation (2.3) and Facts 2.1–2.3, we devise the following algorithm for computing $\det A$ and/or its sign.

ALGORITHM 3.1.

```
Input: an $n \times n$ matrix $A$.
Output: $\det A$.
Computations.
  1. Compute the matrices $P$, $L$, $U$, and $P_1$ satisfying (2.3).
  2. Compute $\det P$, $\det L$, $\det U$, and $\det P_1$.
  3. Compute and output
     $\det A = (\det P)(\det L)(\det U)(\det P_1)$.
```

Correctness of the algorithm immediately follows from Facts 2.1–2.3.

If we only need to output the sign of $\det A$, then we only need to compute the signs of $\det P$, $\det L$, $\det U$, and $\det P_1$ at Stage 2 and to multiply these signs at Stage 3.

The computation of the matrices P, L, U, and P_1 at Stage 1 goes by means of routine application of *Gaussian elimination with complete pivoting*, which, as a by-product, records the permutations defined by the matrices P and P_1. The values (1 or -1) of $\det P$ and $\det P_1$ can be immediately recovered from these records. Furthermore, the matrix L is a unit lower triangular matrix, whose diagonal is filled with ones, so that $\det L = 1$. The computation of $\det U$ only requires $n-1$ multiplications, due to Fact 2.2. The overall computational cost of performing Algorithm 3.1 is dominated by the cost of performing its Stage 1: $\sum_{i=1}^{n-1} i^2 = (2n-1)(n-1)n/6$ multiplications, as many additions/subtractions, as many comparisons, and $\sum_{i=1}^{n-1} i = (n-1)n/2$ divisions.

Gaussian elimination with partial pivoting computes the factorization (2.2) (that is, (2.3) for $P_1 = I$) and uses $(n-1)n/2$ comparisons, rather than $(2n-1)(n-1)n/6$ (compare Remark 4.2 in the next section).

4. NUMERICAL IMPLEMENTATION OF THE SOLUTION BASED ON GAUSSIAN ELIMINATION

In its numerical implementation with a finite precision Algorithm 3.1 computes the triangular matrices $\tilde{L} = L + E_L$ and $\tilde{U} = U + E_U$, as well as the permutation matrices $\tilde{P}$ and $\tilde{P}_1$, where E_L and E_U denote the matrices of the perturbations of L and U, due to accumulation of rounding errors. (Since rounding errors are usually small, we usually have $\tilde{P} = P$ and $\tilde{P}_1 = P_1$, but we do not need to assume this.)

DEFINITION 4.1. *Write*

$$A' = \tilde{P}^{-1} A \tilde{P}_1^{-1}, \qquad \tilde{A}' = A' + E_{A'} = \tilde{L}\tilde{U}, \tag{4.1}$$

and let e', a, $\tilde{\ell}$, and $\tilde{u}$ denote the maximum absolute values of the entries of the matrices $E_{A'}$, A, $\tilde{L}$, and $\tilde{U}$, respectively.

Our next goal is to estimate e' from above, assuming floating point binary arithmetic with rounding to β bits. This means the upper bound

$$\epsilon = 2^{-\beta}, \tag{4.2}$$

on the magnitudes of the relative rounding errors (so that ϵ is an upper bound on the *unit roundoff*, also called the *machine epsilon*).

THEOREM 4.1. *(cf., e.g., [33, p. 181], and [34, pp. 191–196], for derivation and minor refinements.) For a matrix $W = (w_{i,j})$, let $|W|$ denote the matrix $(|w_{i,j}|)$ obtained by replacing each entry $w_{i,j}$ with its absolute value $|w_{i,j}|$. Then, under (4.1), we have*

$$|E_{A'}| \le \left(|A'| + \left|\tilde{L}\right| \left|\tilde{U}\right|\right) \epsilon.$$

COROLLARY 4.1.

$$e' \le e^{+} = \left(a + n\tilde{\ell}\tilde{u}\right) \epsilon. \tag{4.3}$$

In this paper, we will also express some estimates in terms of matrix norms $\|W\|_q$, $q = 1, 2, \infty$, satisfying the following basic relations, where $W = (w_{i,j})$:

$$\|W\|_1 = \max_j \sum_i |w_{i,j}|, \qquad \|W\|_\infty = \max_i \sum_j |w_{i,j}|, \tag{4.4}$$

$$\left(\frac{1}{n}\right) \|W\|_q \le \max_{i,j} |w_{i,j}| \le \|W\|_q, \qquad q = 1, 2, \infty, \tag{4.5}$$

$$\|W\|_2^2 \le \|W\|_1 \|W\|_\infty, \quad \|W\|_2 \le \sqrt{n}\|W\|_q, \qquad q = 1, \infty \tag{4.6}$$

(see [29, pp. 56–58]). The norm $\|W\|_2$ is actually needed only in Section 6.

We obtain from (4.5) that $e' \le \|E'_A\|_\infty$ and recall the following result, in whose statement we use an upper bound ϵ of (4.2).

THEOREM 4.2. *(See [35, Chapter 21].)*

$$e' \le e_+ = \|E'_A\|_\infty \le n^2 a^+ \epsilon, \tag{4.7}$$

where a^+ denotes the maximum absolute value of all the entries of the matrices $A^{(i)}$ overwriting A' and computed in the process of Gaussian elimination, which reduces A' to the upper triangular form.

REMARK 4.1. Ignoring the terms proportional to ϵ^k, $k \ge 2$, we may improve (4.7) as follows: $e' \le 3(n-1)\epsilon a^+ + O(\epsilon^2)$ (see [36, p. 178]). The latter bound, as well as both (4.3) and (4.7), are *a posteriori* bounds: besides the input data, they depend on the values $\tilde{\ell}$, $\tilde{u}$ or a^+, which are available as by-products of our computations. In the context of our task, *a priori* bounds, depending only on the input data may only be needed for theoretical analysis. Such bounds can be obtained based on Wilkinson's estimate (cf. [37])

$$a^+ \le a n^{1/2} \prod_{k=1}^{n-1} (k+1)^{1/(2k)} < 1.8 n^{(\ell n\, n)/4} a, \qquad a = \max_{i,j} |a_{i,j}|.$$

Combining this estimate with Theorem 4.2 implies that

$$e' \le e_+ = \epsilon a n^{2.5} \prod_{k=1}^{n-1} (k+1)^{1/(2k)} < 1.8 \epsilon a n^{2+(\ell n\, n)/4}, \qquad a = \max_{i,j} |a_{i,j}|.$$

The above *a priori* bounds, however, are overly pessimistic according to the extensive practical experience with Gaussian elimination, and this has even motivated the conjecture that, actually, for all matrices A, we have

$$a^+ = O(an), \qquad e' = O\left(\epsilon a n^3\right), \tag{4.8}$$

in the case of complete pivoting.

Now, we wish to estimate the magnitude of the perturbation of $\det A$ and $\det A'$ caused by the perturbation of A'. To simplify the notation, we will assume that $P = P_1 = I$, $A' = A$ and will write

$$e = e', \qquad e_d = \det(A + E_A) - \det A \tag{4.9}$$

(cf. Definition 4.1). We will need the following simple and/or well-known estimates.

FACT 4.1. $|\det A| \le \|A\|_q^n$, $q = 1, 2, \infty$.

FACT 4.2. $\|B\|_q \le \|W\|_q$, $q = 1, \infty$, *if* $|B| \le |W|$; $\|B\|_q \le \|W\|_q$, $q = 1, 2, \infty$, *if B is a submatrix of W.*

FACT 4.3. $\|A + E_A\|_q \le \|A\|_q + ne$, *for* $q = 1, 2, \infty$; $\|A + E_A\|_2^2 \le \|A\|_2^2 + ne$.

By using these results, we estimate e_d of (4.9).

PROPOSITION 4.1. $|e_d| \le (\|A\|_q + ne)^{n-1} n^2 e$, *for* $q = 1, 2, \infty$.

PROOF. Represent $\det(A + E_A)$ as the sum of monomials $\pm \tilde{a}_{i_0,0} \tilde{a}_{i_1,1} \cdots \tilde{a}_{i_{n-1},n-1}$, where $\tilde{a}_{i_g,g} = a_{i_g,g} + e_{A,i_g,g}$, $g = 0, 1, \dots, n-1$. Rewrite such a monomial as a telescopic sum

$$\pm \left[\left(\tilde{a}_{i_0,0} - a_{i_0,0}\right) \tilde{a}_{i_1,1} \cdots \tilde{a}_{i_{n-1},n-1} + a_{i_0,0} \left(\tilde{a}_{i_1,1} - a_{i_1,1}\right) \tilde{a}_{i_2,2} \cdots \tilde{a}_{i_{n-1},n-1} \right.$$
$$\left. + \cdots + a_{i_0,0} a_{i_1,1} \cdots a_{i_{n-2},n-2} \left(\tilde{a}_{i_{n-1},n-1} - a_{i_{n-1},n-1}\right) + a_{i_0,0} a_{i_1,1} \cdots a_{i_{n-1},n-1} \right].$$

Summation of the respective terms of this sum over all the subscripts of the monomials of $\det(A + E_A)$) gives us the following equation:

$$\det(A + E_A) = \sum_{i,j} \left(\tilde{a}_{i,j} - a_{i,j}\right) \tilde{D}_{i,j} + \det A,$$

where $\tilde{D}_{i,j}$ denotes the determinant of the $(n-1) \times (n-1)$ matrix obtained by replacing the first $j-1$ columns of A with the first $j-1$ columns of $\tilde{A} = A + E_A$, and by deleting the i^{th} row and the j^{th} column of the resulting matrix. Therefore,

$$\begin{aligned} |e_d| = |\det(A+E_A) - \det A| &= \left|\sum_{i,j} (\tilde{a}_{i,j} - a_{i,j})\, \tilde{D}_{i,j}\right| \le \sum_{i,j} |e_{A,i,j}| \left|\tilde{D}_{i,j}\right| \\ &\le e \sum_{i,j} \left|\tilde{D}_{i,j}\right| \le n^2 e \max_{i,j} \left|\tilde{D}_{i,j}\right|. \end{aligned}$$

By combining Facts 4.1–4.3, we obtain that $|\tilde{D}_{i,j}| \le (\|A\|_q + ne)^{n-1}$ for $q = 1, 2, \infty$. Substitute the latter estimates into the above bound on $|e_d|$ and yield Proposition 4.1. ▮

Combining Proposition 4.1 with the bound (4.7) enables us to extend Algorithm 3.1 as follows (cf. Remark 4.2 of this section).

ALGORITHM 4.1.

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 3.1 in floating point arithmetic with unit roundoff bounded by $\epsilon$. Let
   $\tilde{U} = U + E_U$ denote the computed upper triangular matrix, approximating the factor $U$
   of $A$ (from (2.2) or (2.3)).
2. Compute the upper bound $e_+$ of (4.7) on $e = e'$ of Definition 4.1.
3. Substitute $e_+$ for $e$ and $\min(\|A\|_1, \|A\|_\infty, (\|A\|_1\|A\|_\infty)^{1/2})$ for $\|A\|_q$ in Proposition 4.1,
   and obtain an upper bound $e_d^+$ on $|e_d|$.
4. Output the values $\det\tilde{U} - e_d^+$ and $\det\tilde{U} + e_d^+$.
```

Correctness of the algorithm follows from Theorem 4.2 and Proposition 4.1.

Due to (4.3), Algorithm 4.1 can be modified with e^+ replacing e_+ at its Stages 2 and 3. In both versions, the computations at Stages 2–4 involve $O(n^2)$ arithmetic operations (hereafter, referred to as 'ops') and comparisons (cf. (4.3), (4.4), (4.7)), which is dominated by the arithmetic cost of performing Stage 1.

We may obtain the sign of $\det A$ as the sign of $\det\tilde{U}$ provided that

$$e_d^+ < \left|\det\tilde{U}\right|. \tag{4.10}$$

Otherwise, some additional computations are needed. They can be simplified (see Section 7) by using the bound

$$|\det A| \le e_d^+ + \left|\det\tilde{U}\right|,$$

which turns into

$$|\det A| \le 2e_d^+, \tag{4.11}$$

unless (4.10) holds (compare (1.1)). For smaller ϵ, the bound (4.11) is substantially stronger than the bounds of Fact 4.1 (compare Proposition 4.1).

REMARK 4.2. Algorithm 4.1 performed with partial (rather than complete) pivoting, numerically computes the factorization (2.2). The advantage is in saving $\sum_{i=1}^{n-1}(i^2 - i) = (n-2)(n-1)n/3$ comparisons. The disadvantage is a plausible substantial growth of the upper bounds on $e = e'$ and e_d. Namely, Wilkinson's *a priori* upper bound on a^+ of Remark 4.1 does not hold anymore, and is replaced by a much weaker *a priori* upper bound $a^+ \le 2^{n-1}a$, $a = \max_{i,j}|a_{i,j}|$. The latter bound has been reached for some specially concocted matrices A, but extensive experience of performing Gaussian elimination in worldwide computational practice suggests that, as a rule, the value a^+ remains bounded from above by the value cna, for some moderate constant c, even in the variant with partial pivoting [30]. This suggests that Algorithm 4.1 should be actually applied with using partial pivoting. In the unlikely case of excessively large a^+, we still may repeat the computations with complete pivoting or we may shift to Algorithms 5.1 and/or 6.1 of the next sections, which compute QR factorization of A. For the latter algorithms, the output error bounds that are very close to (4.8) have been formally proven and not only conjectured (see (6.4) in Section 6). Furthermore, a modification of these algorithms (based on using Givens rotations) is highly effective for computing (the sign of) $\det A$ for a dynamically updated matrix A, which is unmatched if we rely on Algorithms 3.1 and 4.1 (see the end of the next section). Those readers for who these advantages, however, are not important may skip the next two sections and go directly to Section 7.

5. SOLUTIONS BASED ON QR FACTORIZATION

Next, based on the equations (2.4) and (2.5), Facts 2.1, 2.2, and Corollary 2.1, we devise the following algorithm, as an alternative to Algorithm 3.1.

ALGORITHM 5.1.

Input: an $n \times n$ matrix A.
Output: $\det A$.
Computations.

1. Compute a matrix Q satisfying (2.4) and (2.5), and represented either by its entries or by a sequence of its orthogonal factors.
2. Compute $\det Q$.
3. Compute the matrix $R = Q^\top A = (r_{i,j})$.
4. Compute $\det A = (\det Q) \prod_i r_{i,i}$.

Correctness of the algorithm immediately follows from Facts 2.1 and 2.2.

We will consider two effective customary algorithms for computing the QR factorization at Stage 1, that is, the Householder and the Givens algorithms (see [30,37] and compare Remark 6.4 in the next section). The Givens algorithm applied at Stage 1, always produces a matrix Q with $\det Q = 1$ since it computes Q as the product of the *matrices of Givens rotations*, each having determinant 1 (see the matrices G in the beginning of the next section or see [30, p. 201]). Therefore, Stage 2 is a by-product of Stage 1 in this case. Let us deduce a similar property in the case where we perform Stage 1 by means of the Householder algorithm, which computes Q as a product of $n-1$ matrices of Householder transformations.

FACT 5.1. *For a matrix Q of (2.4),(2.5) computed by means of the Householder algorithm, we have*

$$\det Q = (-1)^{n-1}. \tag{5.1}$$

PROOF. Recall that the latter algorithm defines Q as the product of $n-1$ Householder matrices, $Q = \prod_{i=1}^{n-1} H_i$, $H_i = I - 2v_i v_i^\top / v_i^\top v_i$ for some vectors v_i, $i = 1, \dots, n-1$. Now, (5.1) follows since

$$\det H_i = -1, \qquad \text{for all } i. \tag{5.2}$$

To prove (5.2), recall that $H_i^\top H_i = I$, so that $(\det H_i)^2 = 1$. On the other hand, $D(0) = \det(I - 2e_0 e_0^\top) = -1$, where $e_0^\top = (1, 0, \dots, 0)$, $e_0^\top e_0 = 1$. Consider the homotopic transformation $v(t) = e_0 + t(v_i - e_0)$, for t ranging from 0 to 1 and observe that $D(t) = \det(I - v(t)v^\top(t)/(v^\top(t)v(t)))$ is a function continuous in t, $D^2(t) = 1$ for all t, and $D(0) = -1$. Therefore, $D(1) = \det I - v_i v_i^\top / v_i^\top v_i = D(0) = -1$, and (5.2) follows. ∎

Stage 3 of Algorithm 5.1 uses only $n-1$ multiplications. Stage 1 is by far, the hardest. It involves $(4/3)n^3 + O(n^2)$ multiplications, about as many additions/subtractions, and $O(n)$

evaluations of the square roots of positive numbers if the Householder algorithm is applied. With the Givens algorithm, these bounds turn into $2n^3 + O(n^2)$, $n^3 + O(n^2)$, and $O(n^2)$, respectively. This shows some minor advantage of the Householder algorithm (see however, [30, p. 216]), and similarly does the comparison of the associated bounds on the output perturbation caused by rounding errors (compare (6.4) and (6.5) in the next section).

Using Givens rotations, however, is highly effective in the cases where the matrix A is dynamically updated, that is, where some of its rows and/or columns are dynamically replaced by new ones. Givens rotations enable us to update the QR factorization of A successively, by using $O(kn^2)$ ops and square root evaluations, if $O(k)$ rows and/or columns of A are successively replaced by new ones (see [30, Sections 12.6.1–12.6.3, pp. 593–597]).

This is a special feature of Givens rotations, not shared by other QR factorization algorithms. Furthermore, dynamic updatings of the PLU factorization of (2.2) and of the $PLUP_1$ factorization of (2.3) are more complicated and more involved, due to the problem of pivoting [38].

On the other hand, the efficacy of the fast updating of the QR factorization by means of Givens rotations is limited because, after s successive fast updatings, the implicitly computed matrix Q is represented as a product of $r = s + n(n-1)/2$ Givens matrices, which means recursive accumulation of many rounding errors (see the bound (6.3) in the next section for larger r). Therefore, one needs to recompute the QR factorization of the current input matrix from the scratch, after sufficiently many updatings of A, when s and r grow too large.

6. NUMERICAL COMPUTATION OF THE ORTHOGONAL FACTOR Q (IN A FACTORIZED FORM)

It is convenient to keep Q in the factorized form as a product of the matrices of Householder transformations or Givens rotations [30,37]. Keeping also the Householder matrices in the form $H = I - 2vv^\top$, for the associated vectors v, we preserve the equation $\det Q = (-1)^{n-1}$, even in the case of numerical implementation of Stage 1, with a finite precision and rounding errors.

Let us examine the perturbation of Q caused by rounding errors in the case of finite precision implementation of the Givens algorithm. The matrices of Givens rotations are of the form $G = I + R_{i,k}(c,s)$, where the matrix $R_{i,k}(c,s)$ has exactly four nonzeros entries, that is, $R_{i,i}(c,s) = R_{k,k}(c,s) = c - 1$, $R_{i,k} = -R_{k,i} = s$, for two fixed integers i and k, and for a pair of nonzero real c and s, satisfying $c^2 + s^2 = 1$, so that $G^\top G = I$. In a finite precision implementation, c and s appear as $c + e_c$ and $s + e_s$, respectively, where e_c and e_s denote the rounding errors

$$|e_c| < |c|\epsilon, \qquad |e_s| < |s|\epsilon, \tag{6.1}$$

and $\epsilon = 2^{-\beta}$ bounds the magnitudes of relative rounding errors (cf. (4.2)). Then, $\det(G^\top G) = (\det G)^2 = (c + e_c)^2 + (s + e_s)^2 = (c^2 + s^2)(1 + \Delta) = 1 + \Delta$, where, due to the bounds (6.1), we have

$$|\Delta| < (1 + \epsilon)^2 - 1 = 2\epsilon + \epsilon^2. \tag{6.2}$$

Thus, for small ϵ, $|\Delta|$ is small, and $\det G$ is close to 1. We will assume that, at Stage 1 of Algorithm 3.1, we represent Q as the product of r Givens matrices for $r \le (n-1)n/2$, and use a finite precision with the bound ϵ on the relative rounding errors. Then, the matrix Q is actually replaced by the matrix $Q + E_Q$, E_Q denoting the perturbation caused by the rounding errors, and due to (6.2), we have

$$\left(1 - 2\epsilon - \epsilon^2\right)^r \le \det(Q + E_Q) \le \left(1 + 2\epsilon + \epsilon^2\right)^r, \qquad r \le (n-1)\frac{n}{2}. \tag{6.3}$$

Let us next estimate the perturbation of the matrix R caused by rounding errors. According to Wilkinson's *backward error analysis*, the rounding errors of floating point computation cause the same perturbation E_R of the matrix R, as would have followed if the input matrix A had been perturbed by a certain matrix $E = E_A$, and if the subsequent computation of Q and R had been performed exactly, with no errors [37]. Furthermore, Wilkinson has proved the following upper bounds on $\|E\|_2$, assuming an upper bound ϵ on the unit roundoff:

$$\|E\|_2 \le h\|A\|_2, \qquad h < h^+ = 3.35(n-1)(1+9.01\epsilon)^{n-2}\epsilon, \tag{6.4}$$

$$\|E\|_2 \le g\|A\|_2, \qquad g < g^+ = 3n^{5/2}\left(\frac{1+6\epsilon}{1-4.3\epsilon}\right)^{2n-4}\epsilon; \tag{6.5}$$

the bounds (6.4) or (6.5) hold, where the Householder or, respectively, Givens algorithms have been applied at Stage 1 of Algorithm 5.1 (cf. [37, pp. 236, 240], respectively).

REMARK 6.1. The reader is referred to [39] on further refinements of the error bounds for finite precision computation of the QR factorization of a matrix.

REMARK 6.2. The bound (6.5) incorporates the effect of the perturbation of the matrix Q caused by rounding and estimated in (6.3).

By combining the bounds (4.5), (6.4), and (6.5) with Proposition 4.1, we obtain the following result.

PROPOSITION 6.1. *Let Algorithm 5.1 be performed by using the floating point arithmetic with relative rounding errors (unit roundoff) within ϵ. Let at Stage 1 of this algorithm, the Householder or Givens algorithms compute numerically the Householder or Givens factors of Q, respectively. Then Algorithm 5.1 outputs the values $d = (-1)^{n-1}\det\tilde{R}$ or $d = \det\tilde{R}$, respectively, satisfying the following relations:*

$$d = \det A + e_d,$$
$$|e_d| \le n^2 s\|A\|_2\left(\|A\|_q + ns\|A\|_2\right)^{n-1}, \qquad q = 1, 2, \infty, \tag{6.6}$$

where s stands either for h of (6.4) or for g of (6.5), respectively.

Now, we may specify a floating point implementation of Algorithm 5.1 as follows.

ALGORITHM 6.1.

Input and Output: as in Algorithm 4.1.

Computations.

1. Apply Algorithm 5.1 by using the Householder algorithm at its Stage 1, and by performing all the computations in floating point arithmetic with the unit roundoff bounded by ϵ. Let $\tilde{R} = R + E_R$ denote the computed upper triangular matrix approximating the factor R of A in (2.4).
2. Set $s = h$ and specify the two upper bounds of (6.6), on $|e_d|$, for $q = 1$ and $q = \infty$, by substituting for $\|A\|_2$ its upper bound $\min\{\|A\|_1\sqrt{n}, \|A\|_\infty\sqrt{n}, (\|A\|_1\|A\|_\infty)^{1/2}\}$, (compare (4.6)). Of the two computed upper bounds on $|e_d|$, choose the smaller one and denote it e_d^+.
3. Compute and output the values $\det\tilde{R} - e_d^+$ and $\det\tilde{R} + e_d^+$.

Correctness of the algorithm follows from (4.6) and Propositions 4.1 and 6.1. By replacing the Householder algorithm by the Givens algorithm and by substituting $s = g$ into equation (6.6), we arrive at an alternative extension of Algorithm 5.1. In both cases, the computation at Stages 2 and 3 of these extensions only requires $O(n^2)$ ops and comparisons (see (4.4)).

REMARK 6.3. Generally, computing the entries of the matrix Q explicitly gives us no advantage in the context of our topic of computing $\det A$. Indeed, such a computation of Q roughly doubles the computational cost of performing Algorithms 5.1 and 6.1, but does not affect the computed matrix $\tilde{R}$.

REMARK 6.4. At Stage 1 of Algorithm 5.1, we could have alternatively applied the modified Gram-Schmidt (MGS) algorithm, which is one of the customary tools for computing the QR factorization. The MGS algorithm requires $4n^3 + O(n^2)$ ops in order to compute the factor R, and also needs some additional computation in order to find out if $\det Q$ equals 1 or -1. In the case of implementation of the MGS algorithm in floating point arithmetic, the extra factor of $\text{cond}_2 A$ (versus the Householder and Givens algorithms) appears in the upper estimate for the 2-norm of the perturbation of the matrix Q caused by rounding to a fixed finite precision (see [30, p. 219, or 40], and note that $\text{cond}_2 A$ is very large for nearly singular matrices A), which is the main deficiency of the MGS algorithm.

PART II. ALGEBRAIC COMPUTATION APPROACH

7. RECOVERY OF $\det A$ FROM $\det A$ MODULO AN INTEGER M BASED ON THE CHINESE REMAINDER THEOREM

Hereafter, $a \bmod M$ (for two integers a and $M > 1$) denotes an integer b satisfying the relations $0 \le b \le M$, M divides $b - a$. In this and the next sections, we will rely on the following observation.

FACT 7.1. *For any pair of integers d and $M > 2|d|$, we have*

$$d = (d \bmod M), \quad \text{if } d \bmod M < \frac{M}{2}, \quad d = -M + (d \bmod M), \text{ otherwise.} \tag{7.1}$$

Suppose that the input matrix A is filled with integers. If we know an upper bound $d^+ \ge |\det A|$, we may choose any integer M exceeding $2d^+$ and then compute $(\det A) \bmod M$ and recover $\det A$ by applying (7.1).

In Section 10, we will review the computation of $(\det A) \bmod M$ for a fixed integer $M > 1$. Computations modulo M, by these algorithms can be performed with the precision of $\lceil \log_2 M \rceil$ bits, but we wish to perform most of them with a lower precision. In this and the next two sections, we will rely on the following celebrated theorem [41,42].

THEOREM 7.1. *(Chinese remainder theorem.) Let $m_1, \dots, m_k$ denote k pairwise relatively prime integers (say, k distinct primes),*

$$m_1 > m_2 > \cdots > m_k > 1. \tag{7.2}$$

Let D denote an integer satisfying

$$0 \le D < M = m_1 m_2 \dots m_k. \tag{7.3}$$

Let

$$r_i = D \bmod m_i, \tag{7.4}$$

$$M_i = \frac{M}{m_i}, \quad v_i = M_i \bmod m_i, \quad w_i v_i = 1 \bmod m_i, \quad i = 1, \dots, k. \tag{7.5}$$

Then D is a unique integer satisfying (7.3) and (7.4); furthermore,

$$D = \left(\sum_{i=1}^{k} M_i r_i w_i \right) \bmod M. \tag{7.6}$$

ALGORITHM 7.1. Computation of $(\det A) \bmod M$ based on the Chinese remainder theorem.

Input: an integer matrix A, a black box algorithm that computes $(\det A) \bmod m$ for any fixed integer $m > 1$, k integers $m_1, \dots, m_k$ satisfying (7.2) and pairwise relatively prime, and $M = m_1 m_2 \dots m_k$.

Output: $(\det A) \bmod M$.
Computations.

1. Compute $r_i = (\det A) \bmod m_i$, $i = 1, \dots, k$.
2. Compute the integers M_i, v_i, and w_k of (7.5), for $i = 1, \dots, k$.
3. Compute and output D of (7.6).

Correctness of the algorithm immediately follows from Theorem 7.1.

We compute the values M_i, v_i by using (7.5). The values w_i are obtained by means of applying the Euclidean algorithm to the pair of v_i and m_i, $i = 1, \dots, k$ (cf. [41,42]).

We need $O(kn^3)$ 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 are performed with the precision of at most $\lceil \log_2 m_1 \rceil$ bits at Stage 1, and of at most $\lceil \log_2 M \rceil$ bits at Stages 2 and 3. Furthermore, in Section 11, we will show how to decrease the latter precision (at Stages 2 and 3) by using a little more ops.

8. MATRIX SINGULARITY TEST

Next, we will show a modification of Algorithm 7.1, for testing matrix singularity, where Stages 2 and 3 are replaced by much simpler computations.

ALGORITHM 8.1. Matrix singularity test.

Input: as in Algorithm 7.1, where

$$M > |\det A|. \tag{8.1}$$

Output: SINGULAR if $\det A = 0 \bmod M$, NONSINGULAR otherwise.
Computations.

1. As in Algorithm 7.1.
2. Output SINGULAR if $r_i = 0$ for $i = 1, \dots, k$; output NONSINGULAR otherwise.

If $r_i = (\det A) \bmod m_i \neq 0$ for some i, then, clearly, $\det A \neq 0$, and the output of the algorithm is correct. If $r_i = 0 \bmod m_i$ for all i, then $D = 0$ due to (7.6), where $D = (\det A) \bmod M$, by the virtue of Theorem 7.1. Due to (8.1), the latter equations imply that $\det A = 0$, and this completes correctness proof for Algorithm 8.1. ▮

REMARK 8.1. Suppose that we have only computed $r_1 = (\det A) \bmod m_1$. If $r_1 \neq 0$, then $\det A \neq 0$, so that we arrive at a one-sided singularity test. On the other hand, our estimates of Appendix A effectively bounded from above the probability that $\det A \neq 0$, provided that $\det A = 0 \bmod m_1$ for a random prime m_1 in a fixed interval. These observations turn the computation of r_1 into a randomized singularity test for the matrix A.

9. COMPUTING ABSOLUTELY SMALLER DETERMINANTS

The next algorithm extends Algorithm 8.1. Under the assumption that

$$|\det A| < M - m_k, \tag{9.1}$$

it tests if

$$-m_k \leq \det A < m_k, \tag{9.2}$$

and if so, computes $\det A$. The new algorithm shares its Stage 1 with Algorithms 7.1 and 8.1; in addition, it only performs subtractions of k pairs of nonnegative integers bounded from above by m_i, for $i = 1, \dots, k$, and $2k - 2$ integer distinctness tests (each of which tests whether two given integers are distinct or not) for pairs of integers ranging from $-m_1$ to $m_1 - 1$ (cf. (7.2)).

ALGORITHM 9.1. Testing a range for $|\det A|$, and computation (modulo an integer) of absolutely smaller determinants.

Input: as for Algorithm 7.1, assuming (9.1).
Output: $\det A$ if (9.2) holds, the word OVERSIZED otherwise.
Computations.

1. As in Algorithm 7.1.
2. If
$$r_i = r, \qquad i = 1, \dots, k, \tag{9.3}$$
for some integer r, then output r.
3. Otherwise, compute $r_i - m_i$, $i = 1, \dots, k$.
4. If
$$r_i - m_i = r_-, \qquad i = 1, \dots, k, \tag{9.4}$$
for some integer r_-, then output r_-. Otherwise, output OVERSIZED.

Let us show correctness of the algorithm.

If $0 \le \det A < m_k$, then we have

$$\det A = (\det A) \bmod m_i = r_i = r, \qquad i = 1, \dots, k$$

(cf. (7.2)), so that the value $\det A$ is correctly computed at Stage 2.

If $-m_k \le \det A < 0$, then

$$\det A = ((\det A) \bmod m_i) - m_i = r_i - m_i = r_-, \qquad i = 1, \dots, k,$$

so that the value $\det A$ is correctly computed at Stage 4.

Let us next show that neither (9.3) nor (9.4) hold under (9.1) if

$$\det A \ge m_k. \tag{9.5}$$

Let us write
$$\det A = r_i + s_i m_i, \qquad i = 1, \dots, k, \tag{9.6}$$
where $0 \le r_i = (\det A) \bmod m_i < m_i$, $0 \le s_i < M_i$, $i = 1, \dots, k$,

$$0 < s_k < M_k - 1, \tag{9.7}$$

and all s_i are integers (cf. (7.5), (9.1), and (9.5)). Then we have

$$r_i + s_i m_i = r_k + s_k m_k, \qquad i = 1, \dots, k-1. \tag{9.8}$$

If (9.3) holds, then $r_i = r_k = r$, $i = 1, \dots, k-1$, and (9.8) implies that

$$s_i m_i = s_k m_k, \qquad \text{for all } i. \tag{9.9}$$

Since m_i and m_k are relatively prime, (9.9) implies that m_i divides s_k, for all $i < k$. Therefore, the integer $M_k = m_1 \dots m_{k-1} = \operatorname{lcm}(m_1, \dots, m_k)$ divides s_k, since all m_i are pairwise relatively prime. This contradicts (9.7). Therefore, (9.3) cannot hold under (9.5).

If (9.4) holds, we similarly deduce from (9.8) that

$$(s_i + 1)m_i = (s_k + 1)m_k, \qquad i = 1, \dots, k-1, \tag{9.10}$$

which implies that m_i divides $s_k + 1$ for $i = 1, \dots, k-1$. Therefore, M_k divides $s_k + 1$, since all m_i are pairwise relatively prime. Then again, we arrive at a contradiction to (9.7). Therefore, (9.4) cannot hold under (9.5), which implies correctness of Algorithm 9.1 under (9.5).

To complete the correctness proof, it remains to show that neither (9.3) nor (9.4) hold under (9.1) if

$$\det A < -m_k. \tag{9.11}$$

In this case, the integers s_i of (9.6), (9.8)–(9.10) satisfy the bounds $-M_i \le s_i < 0$, $i = 1, \dots, k-1$,

$$-M_k < s_k < -1. \tag{9.12}$$

(cf. (7.5), (9.1), and (9.5)). Now, if (9.3) holds, then we deduce from (9.9) that m_i divides $-s_k$ for $i = 1, \dots, k-1$, and consequently, M_k divides $-s_k$. This contradicts (9.12). Therefore, (9.3) cannot hold under (9.11).

If (9.4) holds, then we deduce from (9.10) that M_k divides $-1 - s_k$. Then again, we arrive at a contradiction to (9.12). Therefore, (9.4) cannot hold under (9.11). This completes the correctness proof. ∎

10. COMPUTATION MODULO A FIXED PRIME OF THE DETERMINANT OF A MATRIX

Let p denote a fixed prime. Then $(\det A) \bmod p$ can be immediately computed as soon as we compute modulo p the PLU factorization of A (see (2.2)). This can be achieved by means of Gaussian elimination with partial pivoting, which involves at most $(2/3)n^3 + O(n^2)$ arithmetic operations (ops) modulo p and at most $(n-1)n/2$ comparisons of integers modulo p with 0. (The latter comparisons are needed in order to ensure that only nonzero entries are used as pivots, so as to avoid divisions by 0.) If, at some elimination step, all the candidate entries of the pivot column equal $0 \bmod p$, then we immediately output $\det A = 0 \bmod p$.

If we need to compute $(\det A)$ mod p for a recursively updated matrix A, we may rely on a modified (*unnormalized*) QR-factorization of A, such that

$$A = (QR) \bmod p, \qquad Q^{\mathsf T} Q = D \bmod p, \tag{10.1}$$

D is a diagonal matrix, R is a unit upper triangular matrix (compare [12]). Such a factorization can be computed (modulo a prime p) by means of the Givens algorithm modified [30, p. 216].

REMARK 10.1. Under the models of parallel computing, the cited algorithms are relatively slow: they use order of at least n parallel steps. Faster parallel algorithms (using polylogarithmic parallel time and order of n^3 ops performed modulo p) can be found in [11,27–29,43–46] (cf. also [47,48]).

REMARK 10.2. If the entries of the matrix A are much smaller than p, then we do not need to reduce modulo p the results of the computations at the initial steps of Gaussian elimination, so that they can be performed in exact rational arithmetic with using lower precision. To keep the precision lower, one may apply the algorithms of [7,8] at these steps. If we compute $\det A$ modulo several primes, this will be a common stage for all the primes.

11. DECREASING THE PRECISION OF THE CHINESE REMAINDER COMPUTATIONS

Let us recall Algorithm 7.1 of Section 7, where we will assume that m_1 is relatively small compared to M but large compared to k. The computation of r_i at Stage 1, and v_i and w_i at Stage 2, can be performed modulo m_i with the precision of $\lceil \log_2 m_i \rceil$ bits, $i = 1, \dots, k$, which is small relative to the precision required in order to represent $M, M_1, \dots, M_k$, and D. Next, we are going to examine the computation of $M_1, \dots, M_k$ at Stage 2 and D at Stage 3, which requires us to perform relatively fewer arithmetic operations (ops) but with a higher precision (cf. (7.6)). We will modify the computations so as to perform them by using a little more ops but a lower precision. Towards this goal, let us fix an *integer base* b such that $\log_2 b$ exceeds the computer precision. Then the integers lying in the range from 0 to $b-1$ will fit the computer precision and will be called *"short" integers* and *b-integers*. The integers that exceed $b-1$ and their negations $-q$ will be called "long" and will be associated with the *b-polynomials*

$$q(x) = \sum_i q_i x^i, \qquad 0 \le q_i < b, \text{ for all } i, \tag{11.1}$$

and with $(-b)$*-polynomials* $-q(x)$, respectively, such that

$$q(b) = q, \qquad -q(b) = -q. \tag{11.2}$$

The polynomials $q(x)$ and $-q(x)$ satisfying (11.1),(11.2) will be called the *b-associates of the integers* q and $-q$, respectively, as well as of any pair of polynomials $Q(x)$ and $-Q(x)$ with integer coefficients satisfying

$$Q(b) = q. \tag{11.3}$$

The transition from a polynomial $Q(x)$ with nonnegative coefficients to its b-associate, which we call the *b-reduction* of $Q(x)$, can be performed by means of first substituting the b-associate for every coefficient of $Q(x)$, then replacing each term of $Q(x)$ by its b-associate, and finally, summing all the resulting b-polynomials together, by means of the known algorithms for multiprecision summation of nonnegative integers (in the base b arithmetic) (see [49, pp. 251,252], and [50,51]). Furthermore, any polynomial $Q(x)$ with integer coefficients can be immediately represented as the difference $Q^+(x) - Q^-(x)$ of two polynomials $Q^+(x)$ and $Q^-(x)$ with nonnegative integer coefficients. We may *b-reduce* $Q(x)$ by b-reducing these two polynomials $Q^+(x)$ and $Q^-(x)$ and subtracting their b-associates from each other by means of a known algorithm for multiprecision integers (in the base b arithmetic) (see [49, pp. 252,253]). We will define the class of *linear operations*, consisting of additions, subtractions, multiplications by ("short") b-integers, and divisions by ("short") b-integers (the latter division operation will only be used in our Algorithms 11.2 and 11.3, which can both be replaced by Algorithm 11.1 at the expense of using by factor of $O(\log k)$ more ops with b-integers).

We will need the following simple fact.

FACT 11.1. *For an integer $b > 1$ and a pair of b-reduced polynomials $U(x)$ and $V(x)$, the sign of $U(b) - V(b)$ coincides with the sign of the leading coefficient of the polynomial $U(x) - V(x)$.*

Let us assume that

$$m_1^2 < b, \tag{11.4}$$

which holds, say, for

$$b = 2^{\lceil \log_2(m_1^2+1) \rceil}.$$

Then, clearly, m_i, r_i, and w_i of (7.2)–(7.6) are b-integers for all i, and the following algorithm computes M, $M_1, \dots, M_k$ by using only linear operations of multiplication by b-integers as follows.

ALGORITHM 11.1.

Input: integers $m_1, \dots, m_k$ satisfying (7.2).

Output: the integers M of (7.3), $M_1, \dots, M_k$ of (7.5).

Initialization. Define the binary tree with the root 1 (at level 0), the leaves $M_1, \dots, M_k$ and other nodes filled with the products of some consecutive moduli $m_1, \dots, m_k$ such that m_i is missing from the parent node if and only if it is missing from at least one of its children nodes.

Computations. Recursively in l, compute the products of the moduli m_i, associated with all the nodes of Level l of the tree, for $l = d, d-1, \dots, 1$. Output the products $M_1, \dots, M_k$ associated with Level 1. Then compute and output $M = M_1 m_1$.

The transition from Level l to Level $l-1$ requires k multiplications by the moduli m_i, for $l = d-1, \ldots, 2$, and $k-2$ multiplications for $l = d$. In the entire algorithm, we use a total of $dk - 1 = k\lceil \log_2 k \rceil - 1$ multiplications, each by a b-integer m_i. All other operands are in the range from 0 to M_k, that is, their b-associates are polynomials of degrees at most $\lfloor (k-1)/2 \rfloor$. The entire computation by Algorithm 11.1 costs $O(k^2 \log k)$ multiplications of pairs of b-integers.

The next algorithm uses only $k-1$ similar multiplications by b-integers m_i for $i = 2, \ldots, k$, and $k-1$ divisions of M by m_i, for $i = 1, 2, \ldots, k-1$, which have a total cost of $O(k^2)$ ring operations and divisions with pairs of b^2-integers as operands.

ALGORITHM 11.2.

Input and **output:** as in Algorithm 11.1.

Computations.

1. Set $m^{(1)} = m_i$, and recursively compute $m^{(i)} = m^{(i-1)} m_i$, $i = 2, \ldots, k$. Output $M = m^{(k)}$, $M_k = m^{(k-1)}$.
2. For $i = 2, \ldots, k$, compute and output $M_i = M/m_i$, $i = 1, \ldots, k-1$.

At Stage 2, we apply the following simple algorithm.

ALGORITHM 11.3.

Input: a b-polynomial $P(x) = \sum_{i=0}^{h} P_i x^i$ and a b-integer $d > 0$.

Output: an integer $0 \le \rho < d$ and a b-polynomial $Q(x)$ of a degree at most h such that $P(b) - dQ(b) = \rho$.

Computations.

1. Compute the quotient q_0 and the remainder ρ_0 of the division of P_h by d.
2. Recursively compute the quotients q_i and the remainders ρ_i of the division of $\rho_{i-1} + bP_{h-1}$ by d, for $i = 1, \ldots, h$.
3. Output the integer $\rho = \rho_h$ and the b-polynomial $Q(x) = \sum_{i=0}^{h} q_i x^{h-i}$.

Correctness of the algorithm and its cost bound of $O(k)$ arithmetic operations with b^2-integers are immediately verified.

Now, we recall that r_i and w_i are b-integers under (11.4), whereas the b-associates of M_i are b-polynomials of degrees less than $(k-1)/2$ for all i. It follows that, based on Algorithms 11.2 and 11.3, we may evaluate the b-associates of $M_i r_i w_i$ for all i, and $D^* = \sum_{i=1}^{k} M_i r_i w_i$, which only involves $O(k^2)$ operations with b-integers.

Let us consider the remaining stage of the reduction modulo M of the integer

$$D^* = \sum_{i=1}^{k} M_i r_i w_i, \qquad 0 \le D^* < M \sum_{i=1}^{k} (m_i - 1) = \kappa M, \tag{11.5}$$

where $\kappa = \sum_{i=1}^{k} (m_i - 1) \le k m_1 - k - \sum_{i=1}^{k-1} i = (m_1 - (k+1)/2)k$.

The latter bound, (7.2) and (11.4) together immediately imply that κ is a b-integer. Now, we may restate our remaining task as follows: given a pair of b-polynomials $M(x)$ and $D^*(x)$, which are b-associates of the two integers M of (7.3) and D^* of (11.5), compute a b-associate $D(x)$ of the integer $D^* - M\bar{Q} = D^* \bmod M$ such that

$$0 \le D^* - \bar{Q}M < M, \tag{11.6}$$

for an integer $\bar{Q}$. Equations (11.5) and (11.6) together imply that

$$0 \le \bar{Q} < \kappa < b, \tag{11.7}$$

so that $\bar{Q}$ is a b-integer, which can be found by means of binary search in the interval from 0 to $\kappa - 1$. Having $\bar{Q}$ of (11.6) computed, we immediately compute the polynomial $D^*(x) - M(x)\bar{Q}$ and then compute and output its b-associate $D(x)$. The binary search will be simplified due to the following observation.

FACT 11.2. *For a pair of b-polynomials $U(x)$ and $V(x)$, the sign of the value $U(b) - V(b)$ of the polynomial $U(x) - V(x)$ at $x = b$ coincides with the sign of the leading coefficient of the latter polynomial.*

ALGORITHM 11.4. Reduction modulo an integer by means of binary search.

Input: two integers κ and b, $b > \kappa > 1$, and a pair of b-polynomials $D^*(x)$ and $M(x)$ such that $D^* = D^*(b)$ and $M = M(b)$ satisfy (11.5) and (7.3).

Output: the integer $D = D^* \bmod M$ (cf. (7.6)) represented by the associated b-polynomial.

Initialization: write $Q^- = 0$ and compute $Q^+ = 2^{\lceil \log_2(\kappa-1)\rceil}$.

Computations.

1. Compute the integer $Q = \lfloor (Q^+ + Q^-)/2 \rfloor$.
2. Compute and b-reduce the polynomial $M(x)Q$, so as to arrive at the b-polynomial $D_Q(x)$, such that $D_Q(b) = MQ$.
3. (a) If the leading coefficient of the polynomial $D^*(x) - D_Q(x)$ is negative, then note that
$$D^* = D^*(b) < D_Q(b) = MQ, \tag{11.8}$$
write $Q^+ = Q$, and repeat the computations starting with Stage 1.
   (b) Otherwise, check if the leading coefficient of the polynomial $D^*(x) - D_Q(x) - M(x)$ is nonnegative, and if so, then note that
$$D^* = D^*(b) \geq D_Q(b) + M(b) = (Q+1)M, \tag{11.9}$$
write $Q^- = Q$, and repeat the computations starting with Stage 1.
   (c) Otherwise, deduce from Fact 11.1 that neither (11.8) nor (11.9) hold, note that, consequently, (11.6) holds, write $\bar{Q} = Q$, and go to Stage 4.
4. Compute the polynomial $\tilde{D}(x) = D^*(x) - M(x)\bar{Q}$. (Since (11.6) holds, $\tilde{D}(b) = D^* \bmod M = D$.) Then compute and output its b-associate polynomial $D(x)$.

Correctness of Algorithm 11.4 immediately follows from (11.5) and Fact 11.1.

The algorithm requires at most $\lceil \log_2(\kappa - 1)\rceil \leq \lceil \log_2((m_1 - (k+1)/2)k - 1)\rceil$ steps of the binary search and a single pass through Stage 4. Each step of the binary search involves $O(k)$ ring operations with b-integers, and so is Stage 4.

REMARK 11.1. The number of ops involved in each step of binary search, in Stage 4 and, consequently, in the entire algorithm, can be substantially decreased if, by means of a special choice of b and $M > 2|\det A| + 1$, we arrive at a sparse polynomial $M(x)$, say, at $M(x) = x^u + 1$, for an appropriate integer u.

Having computed the b-polynomial $D(x)$, we may easily compute the b-associate $D_{b,2}(x)$ of $2D(x)$ and then determine the sign of the leading coefficient of the polynomial $D_{b,2}(x) - M(x)$. Due to Fact 11.1, this will give us the sign of $2D(b) - M(b) = 2D - M$, which will enable us to define the sign of $\det A$, based on (7.1).

REMARK 11.2. For many input instances, we may decrease the arithmetic cost of performing Stages 2 and 3 of Algorithm 11.4 by skipping or not completing some of the b-reductions, as long as our computations still enable us to test if the inequalities (11.7) and (11.8) hold. Similar comments apply to Stage 4 and to the computation of $D_{b,2}(x)$.

Based on combining Algorithms 7.1, 11.2–11.4, we obtain the following result.

THEOREM 11.1. *Suppose that $2k + 1$ given integers $b, m_1, \dots, m_k, r_1, \dots, r_k$, satisfy (7.2) and the following relations: $b > \sum_{i=1}^{k}(m_i - 1)$ and $r_i = D \bmod m_k$, $i = 1, \dots, k$, for some unknown integer D satisfying*
$$|D| < \prod_{i=1}^{k} m_i - m_k.$$

Then it suffices to use $O(k \log b)$ arithmetic operations and comparisons with b^2-integers as operands in order to compute $M = m_1 \dots m_k$ and the b-associate $D(x)$ of $D \bmod M$.

12. p-ADIC (NEWTON-HENSEL'S) LIFTING FOR THE COMPUTATION OF MATRIX INVERSES AND DETERMINANTS

In this section, we will recall another customary approach (alternative to one of Sections 7–9) to decreasing the precision of algebraic computations with no rounding errors. The approach relies on p-adic (Newton-Hensel's) lifting. We will first show how to apply this approach to computing $\det A$ by using order of n^4 log ops, so that the algorithm is apparently inferior to one of Section 7 for larger n but may be competitive for small n. In Section 15, we will show how to modify this approach so as to use order of n^3 ops. The approach will still have a disadvantage of generally involving multiplications of pairs of nonconstant b-polynomials rather than only multiplications by b-integers, as in Section 11.

ALGORITHM 12.1. p-adic lifting of matrix inverses.

Input: an $n \times n$ integer matrix A, an integer $p > 1$, the matrix $S_0 = A^{-1} \bmod p$, and a natural h.

Output: the matrix $A^{-1} \bmod p^H$, $H = 2^h$.

Computations. Recursively compute

$$S_j = S_{j-1}(2I - AS_{j-1}) \bmod p^J, \qquad J = 2^j, \quad j = 1, \dots, h. \tag{12.1}$$

Output S_h.

Correctness of the algorithm immediately follows from the next simple fact.

FACT 12.1. *(See [52].) $S_j = A^{-1} \bmod p^J$, $J = 2^j$ for all j.*

PROOF. Fact 12.1 follows from the matrix equation $I - AS_j = (I - AS_{j-1})^2 \bmod p^J$, implied by (12.1). ∎

At the jth stage, the algorithm uses $O(n^3)$ ops performed modulo p^J (with a precision of at most $\lceil J \log_2 p \rceil$ bits), that is, a total of $O(hn^3)$ ops.

To extend p-adic lifting of matrix inverses to p-adic lifting of $(\det A) \bmod p$, we next recall the equation

$$\det A = \frac{1}{\prod_{k=1}^{n} \left(A_k^{-1}\right)_{k,k}} \tag{12.2}$$

(cf., e.g., [11, p. 327], and Section 15). Here and hereafter, A_k denotes the $k \times k$ leading principal submatrix of A, $k = 1, \dots, n$, so that $A_n = A$, and $(W)_{k,k}$ denotes the (k,k)th entry of a matrix W. To apply (12.2), we need to have the inverses modulo p of A_k available for all k, and for a fixed (prime) integer p. The existence of such inverses for all k is called *strong nonsingularity* of A modulo p. We will assume that, for a given matrix A and for our choice of p, the matrix A is strongly nonsingular modulo p. (With a random choice of a prime p in a fixed interval, we have a convenient estimate for the probability that A is strongly nonsingular modulo p (see Appendix A)). Then we extend Algorithm 12.1 to lifting $\det A$ as follows.

ALGORITHM 12.2. p-adic lifting to matrix determinants.

Input: an integer $p > 1$, an $n \times n$ matrix A, the matrices $S_{0,k} = A_k^{-1} \bmod p$ (so that the matrix A is strongly nonsingular modulo p), and a natural h.

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

Computations.

1. Apply Algorithm 12.1 to all pairs of matrices A_k and $S_{0,k}$ (replacing A and S_0 in the input), so as to compute the matrices $S_{h,k} = A_k^{-1} \bmod p^H$ for $k = 1, \dots, n$.

2. Compute the value

$$\left(\frac{1}{\det A}\right) \bmod p^{2H} = \prod_{k=1}^{n} \left[S_{h,k}\left(2I - A_k S_{h,k}\right)\right]_{k,k} \bmod p^{2H}. \tag{12.3}$$

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

Correctness of Algorithm 12.2 immediately follows from (12.2) and from correctness of Algorithm 12.1. The overall computational cost is dominated by order of hn^4 ops performed at Stage 1 with the precision of at most $\lceil H \log_2 p \rceil$ bits. At Stage 2, we only need $O(n^3)$ ops (in order to premultiply A_k by the last row of $S_{h,k}$, to postmultiply the resulting vector by the last column of $S_{h,k}$, to subtract the resulting value from $2(S_{h,k})_{k,k}$, for $k = 1, \dots, n$, and to multiply together the results for all k). These operations, as well as the single operation of Stage 3, are performed modulo p^{2H}, that is, with the precision of $2H \log_2 p$ bits. In the next section, we will show how to perform the latter computations with a lower precision.

To complete the computation of $(\det A) \bmod p^{2H}$, we will complement Algorithm 12.2 by Gaussian elimination modulo p applied in order to compute $S_{0,k} = A_k^{-1} \bmod p$ for all k. In the special case, where the computations require no pivoting and no row interchange, we compute and output the $L_k U_k$ factorizations modulo p of A_k, simultaneously for all k, where L_k and U_k are the $k \times k$ leading principal submatrices of the computed factors of A, that is, L and U, respectively, (cf. (2.1)). Then $A_k^{-1} = U_k^{-1} L_k^{-1}$, and Gaussian elimination outputs $A_k^{-1} \bmod p$ for all k, at the cost of performing $O(n^3)$ ops modulo p.

In the general case, Gaussian elimination with partial pivoting either determines that $\det A = 0 \bmod p$ or outputs a PLU factorization of A modulo p, where L and U^{T} denote two lower triangular matrices and P is a permutation matrix, whose determinant (1 or -1) is available. Then we have LU factorizations of the matrix $B = B_n = P^{-1}A$ and of all its leading principal submatrices, $B_1, \dots, B_{n-1}$. By applying Algorithm 12.2 to the matrix B replacing A in the input, we compute $(\det B) \bmod p^{2H}$ and then immediately recover $(\det A) \bmod p^{2H} = ((\det B) \bmod p^{2H} \det P) \bmod p^{2H}$. To conclude, we will estimate H and p sufficient in order to satisfy the bound $p^{2H} > 2|\det A|$. Due to Hadamard's bound,

$$|\det A| \le \left(a\sqrt{n}\right)^n, \qquad a = \max_{i,j} |a_{i,j}|, \tag{12.4}$$

which holds for any matrix $A = (a_{i,j})$, it suffices to choose p and H satisfying

$$2\left(a\sqrt{n}\right)^n < p^{2H}, \qquad 2H > n \log_p\left(a\sqrt{n}\right) + \log_p 2. \tag{12.5}$$

13. DECREASING THE PRECISION OF THE COMPUTATION OF p-ADIC LIFTING

At the jth stage of Algorithm 12.1, we generally compute with the precision of $\lceil j \log_2 p \rceil$ bits. For larger j, this value may exceed the computer precision β. Let us next extend the techniques of Section 11 in order to decrease the precision of computing so as to keep it below the fixed level β, denoting the computer precision. We will choose the base $b = p^G$, where G is a power of 2, $G \le H/2$, so that G divides $H/2$, and

$$G \le \log_2\left(\frac{(b-1)^2 nH}{G}\right) < \beta, \tag{13.1}$$

β denoting the computer precision. We associate the entries of the matrices $A_k \bmod p^J$ and $S_{j-1,k} \bmod p^J$ for $J = 2^j$, and for all j and k, with the b-polynomials in x (see the definition of b-polynomials in Section 11); such polynomials take on the values of these entries for $x = b$

and have degrees at most $(J/G) - 1$. Let $A_{j,k}(x)$ and $S_{j-1,k}(x)$ denote the resulting matrix polynomials such that

$$A_{j,k}(b) = A_k \bmod p^J, \qquad S_{j,k}(b) = S_{j,k} \bmod p^J.$$

Then, for $J \geq G$, we associate the p-adic lifting step (12.1) with the computation of the matrix polynomial

$$S_{j,n}(x) = S_{j-1,n}(x)\,(2I - A_{j,n}(x)S_{j-1,n}(x)) \bmod x^{J/G}, \tag{13.2}$$

where the matrix polynomials $A_{j,n}(x)$ and $S_{j-1,n}(x)$ are given as input and where $j = 1, 2, \ldots, h$. We will perform arithmetic operations modulo $x^{J/G}$ with the matrix polynomials of (13.2). Having completed such an operation, we will apply b-reduction to all the entries of the output matrix polynomial, followed by a new reduction modulo $x^{J/G}$. Due to the recursive application of b-reductions and modular reductions, the entries of all the operands are polynomials with integer coefficients in the range from $1 - b$ to $b - 1$. Therefore, the output entries, even before their b-reductions, are the polynomials with coefficients in the range from $-B$ to B with $B \leq 2^\beta$, so that the β-bit precision suffices in all these computations, due to (3.1). Similar analysis applies to the computation of the matrix polynomials associated with the matrices $S_{j,k}$ for all j and $k < n$.

14. DECREASING THE PRECISION OF COMPUTING $\det A$ WHEN THE INVERSES OF A_k ARE AVAILABLE

Having computed the matrix polynomials $A_{h+1,k}(x)$ and $S_{h,k}(x)$ associated with the matrices $A_k \bmod p^{2H}$ and $S_{h,k}$, we next compute the polynomial

$$D^-(x) = \prod_{k=1}^{n} \left[S_{h,k}(x)\,(2I - A_{h+1,k}(x)S_{h,k}(x))\right]_{k,k} \bmod x^{2H/G},$$

associated with $(1/\det A) \bmod p^{2H}$. Then again, we assume that reduction modulo $x^{2H/G}$ and the extension of the b-reduction described at Stages 5–7 of Algorithm 11.1, have been repeatedly applied after each arithmetic operation with polynomials in the process of computing $D^-(x)$. Finally, we apply a known algorithm for computing the reciprocal of a polynomial (see, e.g., [11, p. 24]) in order to compute the polynomial $\hat{D}(x) = (1/D^-(x)) \bmod x^{2H/G}$. Then $\hat{D}(b) = (\det A) \bmod p^{2H}$, due to (12.3).

15. FASTER p-ADIC LIFTING OF THE DETERMINANTS

Some deficiency of Algorithm 12.2 is due to its excessive work for inverting all the $k \times k$ leading principal submatrices A_k of A, which requires order of n^4 ops. Let us show how to avoid this stage, so as to use $O(n^3)$ ops. Due to our argument of the end of Section 12, we may assume that the matrix A is strongly nonsingular modulo p. Then block Gauss-Jourdan elimination applied to the 2×2 block matrix

$$A = \begin{pmatrix} B & C \\ E & G \end{pmatrix}, \tag{15.1}$$

defines the following well-known block factorization of A:

$$A = \begin{pmatrix} I & O \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & O \\ O & S \end{pmatrix} \begin{pmatrix} I & B^{-1}C \\ O & I \end{pmatrix}, \tag{15.2}$$

where

$$S = G - EB^{-1}C. \tag{15.3}$$

Moreover, strong nonsingularity of A modulo p implies strong nonsingularity modulo p of the matrices B and S. Now, we arrive at the following divide-and-conquer algorithm.

ALGORITHM 15.1. Computing the determinant of a strongly nonsingular matrix.

Input: an $n \times n$ strongly nonsingular matrix A.
Output: $\det A$.
Computation.

1. Partition the input matrix A according to (15.1), where B is an $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ matrix. Compute B^{-1} and S of (15.3).
2. Compute $\det B$ and $\det S$.
3. Compute and output

$$\det A = (\det B) \det S. \tag{15.4}$$

Correctness of the algorithm immediately follows from (15.2). Letting $C_D(k)$ and $C_I(k)$ denote the arithmetic cost of computing the determinant and the inverse of any $k \times k$ strongly nonsingular matrix, respectively, we obtain from (15.3) and (15.4) that

$$C_D(n) \le C_D\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + C_D\left(\left\lceil \frac{n}{2} \right\rceil\right) + C_I\left(\left\lfloor \frac{n}{2} \right\rfloor\right). \tag{15.5}$$

The latter bounds also hold for solving the same computational problems modulo p^{2H} for a fixed prime p and a positive integer H, assuming strong nonsingularity of A modulo p. We may compute modulo p^H the inverse of a $k \times k$ matrix by using $C_I(k) = O(k^3 \log H)$ ops, by means of Algorithm 12.1. Substitute this bound into (15.5), and obtain recursively that $C_D(n) = O(n^3 \log H)$, that is, we deduce the latter bound based on recursive application of the factorization (15.2) to the matrices B and S and on computing the inverses modulo p^{2H} by means of Algorithm 12.1 (see the bound (12.5) on H). Our techniques of Section 13 for decreasing the precision of the computation of $(\det A) \bmod p^{2H}$ are immediately extended to the case of Algorithm 15.1.

16. DISCUSSION. DECREASING THE PRECISION OF THE STRAIGHTFORWARD ALGORITHMS

Further theoretical and experimental study may reveal other competitive approaches to determining the sign of $\det A$. In particular, we may extend the approach of Sections 11, 12, and 14 to any algorithm that outputs $\det A$ or its sign in the result of a sequence of ring operations $+$, $-$, and $*$. For instance, we may do this for the algorithms of [7,8] (though their b-associates generally involve multiplications of nonconstant b-polynomials) as well as for the straightforward algorithms based on the decompositions

$$\det A = \sum_{i=1}^{n} (-1)^{i-1} a_{i,1} \det A_{i,1}, \tag{16.1}$$

or

$$\det A = \sum_{i=1}^{n} (-1)^{i-1} a_{1,j} \det A_{1,j}, \tag{16.2}$$

and on similar recursive decompositions of $\det A_{1,j}$ and/or $\det A_{i,1}$. Here, $A_{i,j}$ denotes an $(n-1) \times (n-1)$ submatrix of A obtained by deleting the i^{th} row and the j^{th} columns of A. Every multiplication in the latter algorithms has an operand $a_{i,j}$ for some pair (i,j). By choosing $b > \max_{i,j} |a_{i,j}|$, we may ensure that the b-associates of these operands are b-integers, so that the b-associates of the algorithms only involve linear operations. Hadamard's bound (12.4), which holds for any $n \times n$ matrix A, enables us to bound similarly the absolute values of all operands of these straightforward algorithms, so that the b-associates of these operands are polynomials of degrees less than $(n+1)\log_b(a\sqrt{n})$. These observations suggest that the algorithms are competitive for small n.

For large n, however, the algorithms use prohibitively many ops. Indeed, denoting the number of ops by C_n, we have $C_n = (2n+1) + nC_{n-1}$. By extending this relation recursively, to $C_{n-1}, C_{n-2}, \dots$, we obtain the expression $1+C_n = n!\sum_{i=1}^{n} 1/i!$, which converges to $1+C_n = en!$, $e = 2.7182\dots$, as $n \longrightarrow \infty$.

APPENDIX A
ESTIMATING THE PROBABILITY OF SINGULARITY

We will estimate the probability that $\det A \neq 0$ provided that $(\det A) \bmod p = 0$ for a random prime p and for a fixed integer matrix A.

FACT A.1. *(See [53].) Let $f(n) > 0$ for all natural n, $\lim_{n\to\infty} f(n) = \infty$. Then there exist two positive constants C and n_0 such that the interval*

$$\left\{p : \frac{f(n)}{n} < p < f(n)\right\} \tag{A.1}$$

contains at least $f(n)/(C\log_2 f(n))$ distinct primes provided that $n > n_0$.

COROLLARY A.1. *(See [28,29].) For a fixed nonsingular $n \times n$ matrix A filled with integers, for $f(n) \geq n^{d+1}$, $d > 0$, and for a random prime p chosen in the interval (A.1), under the uniform probability distribution, we have*

$$q = \textit{Probability}\{\det A = 0 \bmod p\} \leq (1+d)\frac{(C\log_2|\det A|)}{(f(n)d)}. \tag{A.2}$$

PROOF. Let k distinct primes from the interval (A.1) divide $|\det A|$. Then their product P also divides $|\det A|$. Therefore, this product P satisfies the bounds $(f(n)/n)^k < P \leq |\det A|$.

It follows that $(\det A) \bmod p = 0$ for at most $k \leq (\log_2|\det A|)/\log_2(f(n)/n)$ primes in (A.1), among at least $f(n)/(C\log_2 f(n))$. Hence,

$$q \leq C\frac{(\log_2|\det A|)\log_2 f(n)}{f(n)\log_2(f(n)/n)},$$

and the bound (A.2) follows for $f(n) \geq n^{1+d}$. ∎

In particular, we may ensure that

$$q = \text{Probability}\{\det A = 0 \bmod p\} \leq \Delta,$$

for a fixed Δ, by choosing p in the interval (A.1) where, say,

$$f(n) \geq \max\left\{n^2, \frac{(2C\log_2|\det A|)}{\Delta}\right\}.$$

APPENDIX B
NUMERICAL EXPERIMENTS FOR PART I

A set of experiments has been performed for numerical computation of the determinants of $n\times n$ matrices for $2 \leq n \leq 10$, based on computing the LU, PLU, $PLUP'$, and QR (Householder) factorizations of A.

Table 1. Average estimated relative error on 10,000 random matrices.

Matrix Size	LU	PLU	$PLUP'$	QR
2	1.112e−011	1.836e−010	2.312e−014	6.877e−014
3	3.215e−005	9.644e−010	4.510e−013	7.702e−013
4	1.138e−003	3.007e−004	2.812e−010	1.827e−009
5	9.501e−003	2.677e−004	6.277e−012	3.729e−012
6	4.001e−002	8.723e−004	4.683e−011	1.049e−011
7	1.220e−001	3.056e−003	1.087e−010	1.256e−011
8	2.624e−001	5.316e−003	3.001e−010	1.438e−011
9	4.369e−001	8.361e−003	5.955e−010	1.733e−011
10	6.158e−001	1.210e−002	1.716e−009	1.904e−011

Table 2. "Estimated failure" count on 10,000 random matrices.

Matrix Size	LU	PLU	$PLUP'$	QR
2	0	0	0	0
3	0	0	0	0
4	8	3	0	0
5	73	2	0	0
6	332	7	0	0
7	1046	26	0	0
8	2353	45	0	0
9	3999	71	0	0
10	5819	105	0	0

Table 3. Average estimated relative error on random matrices having ±1 determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
2	1.841e−010	1.154e−012	1.154e−012	3.138e−010
3	3.563e−004	1.440e−006	2.318e−007	1.129e−006
4	6.155e−002	3.287e−003	3.194e−004	1.528e−003
5	6.633e−001	3.270e−001	2.213e−001	6.315e−001
6–10	> 9.5e−001	> 9.5e−001	> 9.5e−001	> 9.5e−001

Table 4. "Estimated failure" count on 10,000 random matrices having ±1 determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
2	0	0	0	0
3	1	0	0	0
4	431	16	0	0
5	5572	1840	816	3986
6–10	> 9500	> 9500	> 9500	> 9500

The algorithms have been implemented with C++ and built as a console application with Microsoft Visual C++ 4.0 compiler and linker. All numerical operations have been performed with double precision floating point arithmetic. The double precision representation of a number uses 64 bits: 1 for the sign, 11 for the exponent, and 52 for the mantissa. Its range is $\pm 1.7 \times 10^{308}$ with at least 15 decimal digits of precision. The test results have been collected on a Pentium-100 MHz PC, running under Windows 95's DOS session. The system pseudo-random number generator functions **srand()** and **rand ()** have been used to generate input matrices.

All the output values representing the determinants have been compared with the error bounds e_d^+ and e_d estimated according to (4.10) and (6.6), respectively. We considered three classes of input matrices A. In the cases where $|e_d^+| \geq |\det A|$ or $|e_d| \geq |\det A|$ for the computed values

Figure 5. Actual failure count in 10,000 tests on random matrices having ± 1 determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
≤ 4	0	0	0	0
5	10	0	0	0
6	25	0	0	0
7	46	0	0	0
8	78	0	0	0
9	76	1	5	5
10	141	56	114	144

Table 6. Average estimated relative error on random matrices having small determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
2	7.207e−012	1.389e−012	1.645e−012	7.705e−012
3	6.776e−007	2.132e−007	2.261e−009	9.055e−009
4	5.100e−003	5.500e−004	9.923e−007	3.878e−006
5	1.005e−001	3.859e−003	3.315e−004	1.444e−003
6	4.015e−001	1.229e−001	5.013e−002	1.668e−001
7	6.864e−001	5.414e−001	4.761e−001	5.965e−001
8	8.367e−001	6.601e−001	6.566e−001	6.593e−001
9	9.381e−001	7.607e−001	7.510e−001	7.582e−001
10	9.844e−001	8.243e−001	8.010e−001	8.203e−001

Table 7. "Estimated failure" count on 10,000 random matrices having small determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
2	0	0	0	0
3	0	0	0	0
4	43	6	0	0
5	918	24	0	0
6	3839	937	235	1252
7	6709	5360	4640	5974
8	8218	6584	6561	6577
9	9295	7553	7491	7530
10	9802	8111	7935	8076

Table 8. Actual failure count in 10,000 tests on random matrices having small determinants.

Matrix Size	LU	PLU	$PLUP'$	QR
≤ 5	0	0	0	0
6	18	0	0	0
7	49	0	0	0
8	53	0	0	0
9	55	0	0	0
10	71	0	0	0

of $\det A$, e_d^+, and/or e_d, "failure" has been recorded. Otherwise, "correct solution" has been recorded. For each of the three classes of $n \times n$ input matrices and for each n the average of the estimated relative error ($\min(1, |e_d^+|)$ if $\det A = 0$, $\min(1, |(e_d^+)/\det A|)$ otherwise) is presented. Table 5 and Table 8 also include actual failure count in addition to "estimated failure" count since the input matrices are generated with their determinants known beforehand. It can be seen that the "actual failures" occur substantially less frequently than the "estimated failures."

Test 1: Random Matrices

Determinants of 10000 random matrices have been evaluated. The entries of the input matrices are random integers in the interval $(-32768, 32768)$. The estimated relative errors are presented in Table 1, "estimated failure" count is presented in Table 2.

Test 2: Matrix Having ± 1 Determinants

The input matrices are composed by using the following steps.

1. Compute a pair of lower and upper triangular matrices M and N, respectively, where the nondiagonal entries of M and N are random integers in the interval $(-p, p)$, for $p = \sqrt{2048/n}$.
2. Set $M_{i,i}$ and $N_{i,i}$ to 1, where $i = 1, \dots, n$.
3. Compute $A = MN$.
4. Swap a random pair of rows in matrix A.
5. Repeat Step 4 (the previous step) m times, where m is a random integer in $[0, n)$.

The average estimated relative errors are presented in Table 3. The "estimated failure" count recorded in 10,000 tests is presented in Table 4. The exact values of the determinants are known to equal $(-1)^m$ in this case.

If an algorithm outputs det A satisfying $|\det A - (-1)^m| \geq 1$, an "actual failure" is recorded for the test and the count is presented in Table 5.

Test 3: Matrix Having Small Determinants

The input matrices are composed by using the following steps.

1. Compute a pair of lower and upper triangular matrices M and N, respectively, where the nonzero entries of M and N are random integers in the interval $(-p, p)$, for $p = \sqrt{512/n}$.
2. Compute $A = MN$ and $D = \prod_{i=1}^{n} M_{i,i} N_{i,i}$.
3. Swap a random pair of rows in matrix A.
4. Repeat Step 3 (the previous step) m times, where m is a random integer in $[0, n)$.

The average estimated relative errors are presented in Table 6. The "estimated failure" count recorded in 10,000 tests is presented in Table 7. The exact values of the determinants are known to equal $(-1)^m D$ in this case. If an algorithm outputs det A satisfying $|(\det A)/D - (-1)^m| \geq 1$, an "actual failure" is recorded for the test and the count is presented in Table 8.

REFERENCES

1. T. Muir, *The Theory of Determinants in Historical Order of Development*, (four volumes bound as two: Vol. I, 1693–1841; Vol. II, 1841–1860; Vol. III, 1861–1880; Vol. IV, 1881–1900), Dover, New York, (1960).
2. T. Muir, *Contributions to the History of Determinants*, Blackie and Son, London, (1930 and 1950).
3. R.H. Macmillan, A new method for the numerical evaluation of determinants, *J. Roy. Aeronaut. Soc.* **59** (772ff), (1955).
4. E. Durand, *Solutions Numériques des Équations Algébriques*, Volume II: Systèmes de Plusieurs Équations, Valeurs Propres des Matrices, Masson et Cie, Paris, (1961).
5. L. Fox, *An Introduction to Numerical Linear Algebra*, Clarendon Press, Oxford, (1964).
6. J.B. Rosser, A method of computing exact inverses of matrices with integer coefficients, *J. Res. Na. Bur. Standards*, Sect. B **49**, 349–358, (1952).
7. J. Edmonds, Systems of distinct representatives and linear algebra, *J. Res. Nat. Bur. Standards*, Sect. B **71** (4), 241–245, (1967).
8. E.H. Bareiss, Sylvester's identity and multistep integer-preserving Gaussian elimination, *Math. of Comp.* **22**, 565–578, (1968).
9. V.Y. Pan, *How to Multiply Matrices Faster*, Lecture Notes in Computer Science, Volume 179, Springer, (1984).
10. V.Y. Pan, Computing the determinant and the characteristic polynomial of a matrix via solving linear systems of equations, *Information Processing Letters* **28** (2), 71–75, (1988).
11. D. Bini and V.Y. Pan, *Polynomial and Matrix Computations*, Vol. 1, Fundamental Algorithms, Birkhaeuser, Boston, (1994).
12. K.L. Clarkson, Safe and effective determinant evaluation, In *Proc. 33rd Annual IEEE Symp. on Foundations of Computer Science*, pp. 387–395, IEEE Computer Society Press, (1992).

13. F. Avnaim, J.-D. Boissonnat, O. Devillers, F.P. Preparata and M. Yvinec, Evaluating signs of determinants using single-precision arithmetic, *Algorithmica* (to appear).
14. M. Deza and M. Laurent, Applications of cut polyhedra, Report LIENS-92-18, Laboratoire d'Informatique, Ecole Normale Superieure, Paris, France, (1992).
15. M. Deza and M. Laurent, Applications of cut polyhedra, *J. of Computational and Applied Math.* **55** (1), 191–216, (1994).
16. M. Deza and M. Laurent, Applications of cut polyhedra, *J. of Computational and Applied Math.* **55** (2), 217–247, (1994).
17. R.M. Erdahl and V.H. Smith, Editors, *Density matrices and density functionals*, *Proc. of the A. John Coleman Symp.*, Reidel, Dordrecht, (1987).
18. D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra, *Discrete Comput. Geometry* **8**, 295–313, (1992).
19. K. Fukuda and V. Rosta, Combinatorial face enumeration in convex polytopes, *Computational Geometry, Theory and Applications* **4**, 191–198, (1994).
20. R.E.M. Moore and I.O. Angell, Voronoi polygons and polyhedra, *J. of Computational Physics* **105**, 301–305, (1993).
21. I.Z. Emiris, J.F. Canny and R. Seidel, Efficient perturbations for handling geometric degeneracies, *Algorithmica* (to appear).
22. C. Yap, Towards exact geometric computation, *Computational Geometry, Theory and Applications* (to appear).
23. J. Demmel, *From Topology to Computation: Proc. of the Smalefest*, (Edited by M.W. Hirsch, J.E. Marsden and M. Shub), pp. 305–316, Springer, New York, (1993).
24. S. Linnainmaa, Taylor expansion of the accumulated rounding errors, *BIT* **16**, 146–160, (1976).
25. W. Baur and V. Strassen, On the complexity of partial derivatives, *Theoretical Computer Science* **22**, 317–330, (1983).
26. U. Kulish and W.L. Miranker, *Computer Arithmetic in Theory and Practice*, Academic Press, New York, (1980).
27. V.Y. Pan, Complexity of computations with matrices and polynomials, *SIAM Review* **34** (2), 225–262, (1992).
28. V.Y. Pan, Fast and efficient parallel algorithms for the exact inversion of the integer matrices, In *Proc. 5th Conference on Foundation of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 206, pp. 504–521, Springer, Berlin, (1985).
29. V.Y. Pan, Complexity of parallel matrix computations, *Theoretical Computer Science* **54**, 65–85, (1987).
30. G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, MD, (1989).
31. K.A. Gallivan, R.J. Plemmons and A.H. Sameh, Parallel algorithms for dense linear algebra computations, *SIAM Review* **32** (1), 54–135, (1990).
32. H. Brönnimann, I.Z. Emiris, V.Y. Pan and S. Pion, Computing exact geometric predicates using molular arithmetic with single precision, In *Proc. 13th Ann. ACM Symp. on Computational Geometry*, ACM Press, New York, (1997).
33. S.D. Conte and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York, (1980).
34. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, (1992).
35. G. Forsythe and C. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice Hall, Englewood Cliffs, NJ, (1967).
36. G. Dahlquist and Å. Björk, *Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ, (1967).
37. J.H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendion Press, Oxford, (1965).
38. P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, Methods for modifying matrix factorizations, *Math. of Computation* **28**, 505–535, (1974).
39. G.W. Stewart, Perturbation bounds for the QR factorization of a matrix, *SIAM J. on Numerical Analysis* **14**, 509–518, (1977).
40. Å. Björck, Solving linear least squares problems by Gram-Schmidt orthogonalization, *BIT* **7**, 1–21, (1967).
41. A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, (1974).
42. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, (1975).
43. E. Kaltofen and V.Y. Pan, Processor efficient parallel solution of linear systems over an abstract field, In *Proc. 3rd Ann. ACM Symp. on Parallel Algorithms and Architectures*, pp. 180–191, ACM Press, New York, (1991).
44. E. Kaltofen and V.Y. Pan, Processor efficient parallel solution of linear systems—II. The positive characteristic and singular cases, In *Proc. 33rd Ann. IEEE Symp. on Foundation of Computer Science*, pp. 714–723, IEEE Computer Society Press, (1992).
45. V.Y. Pan, Parametrization of Newton's iteration for computations with structured matrices and applications, *Computers Math. Applic.* **24** (3), 61–75, (1992).
46. V. Y. Pan, Parallel computation of polynomial GCD and some related computations over abstract fields, *Theoretical Computer Science* **162** (2), 173–223, (1996).
47. D.W. Wiedemann, Solving sparse linear equations over finite fields, *IEEE Trans. on Information Theory* **IT-32** (1), 54–62, (1986).

48. E. Kaltofen and B.D. Saunders, On Wiedemann's method for solving sparce linear systems, In *Proc. AAECC-5*, *Lecture Notes in Computer Science*, Vol. 536, pp. 29–38, Springer, Berlin, (1991).
49. D.E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, Additon-Wesley, Massachusetts, (1981).
50. Y.P. Ofman, On the algorithmic complexity of discrete functions, (English transl.: *Soviet Physics-Doklady*, **7**, (7), pp. 589–591, (1963)), *Dokl Acad. Nauk SSSR* **145** (1), 48–51, (1962).
51. Y.P. Ofman, Univeral automaton, (in Russian), *Moscow Math. Society (Trudy)* **14**, 186–199, (1965).
52. R.T. Moenck and J.H. Carter, Approximate algorithms to derive exact solutions to systems of linear equations, In *Proc. EUROSAM*, Lecture Notes in Computer Science, Vol. 72, pp. 65–73, Springer, Berlin, (1979).
53. K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Springer, New York, (1982).