2007

TR-2007011: Numerical Computation of Determinants with Additive Preconditioning

V. Y. Pan

B. Murphy

G. Qian

R. E. Rosholt

I. Taj-Eddin

Numerical Computation of Determinants with Additive Preconditioning *

V. Y. Pan[1,3], B. Murphy[1], G. Qian[2],
R. E. Rosholt[1], I. Taj-Eddin[2]
[1] Department of Mathematics and Computer Science,
Lehman College, The City University of New York,
Bronx, NY 10468 USA
victor.pan/brian.murphy/rhys.rosholt@lehman.cuny.edu
[2] Ph.D. Program in Computer Science,
The City University of New York,
New York, NY 10036 USA
gqian/itaj-eddin@gc.cuny.edu
[3] http://comet.lehman.cuny.edu/vpan/

Abstract

Various geometric and algebraic computations (e.g., of the convex hulls, Voronoi diagrams, and scalar, univariate and multivariate resultants) boil down to computing the sign or the value of the determinant of a matrix. For these tasks numerical factorizations of the input matrix is the most attractive approach under the present day computer environment provided the rounding errors are controlled and do not corrupt the output. This is the case where the input matrix is well conditioned. Ill conditioned input matrices, however, frequently arise in geometric and algebraic applications, and this gives upper hand to symbolic algorithms. To overcome the problem, we apply our novel techniques of additive preconditioning and combine them with some nontrivial symbolic-numerical techniques. We analyze our approach analytically and demonstrate its power experimentally.

Key words: Determinants, Symbolic-numerical computations, Additive preconditioning, Aggregation, MSAs.

*Supported by PSC CUNY Awards 66437-0035 and 67297-0036

1 Introduction and the background

1.1 The problem of computing determinants

The classical problem of computing the determinant of a matrix (see, e.g., [M30/50], [M55], [D61]) has important applications to geometric and algebraic computations. Especially, the computation of a convex hull, a Voronoi diagram, the orientation of a polyhedron and an algebraic variety, and the arrangement of lines and line segments can be reduced to computing the sign of the determinant, that is to testing whether $\det A = 0$, $\det A > 0$, or $\det A < 0$ for an associated $n \times n$ matrix A [AF92], [MA93], [FR94], [FvW93], [BKM95], [EC95], [YD95], [Y97], [E98], [BEPP97/99], whereas many multivariate polynomial computations involve the evaluation and expansion of scalar, univariate and multivariate determinants of the associated Newton's and resultant structured matrices (see [EP03/05, Sections 5–7], the bibliography therein, and our Appendix E).

These applications have motivated extensive algorithmic work on computing the value and particularly the sign of a determinant (see [P87], [P88], [C92], [BKM95], [FvW96], [ABD97], [BEPP97/99], [ABM99], [EGV00], [PY99/01], [KV04], [P04], [S05], the bibliography therein, and our Appendix D).

1.2 Arithmetic filtering and symbolic-numerical computations

According to *arithmetic filtering*, in the variety of the known algorithms we first choose faster ones, which compute the certified correct outputs on most of the input instances. In the rare cases of failure, we apply slower algorithms that work for a wider range of inputs.

In particular, in the present day computing environment, numerical algorithms run faster, which makes them most attractive. Due to the rounding errors, however, they fail to produce correct output where the input matrix is ill conditioned, which is frequently the case in geometric and algebraic applications. In contrast symbolic methods are error-free.

Thus one can first apply numerical factorization algorithms that compute a determinant as by-product. If they fail, one can apply symbolic algorithms, which have a number of highly developed implementations, in Maple (LinearAlgebra:Modular package), Magma, Cocoa, LinBox, and NTL. Failure of numerical algorithms may imply a smaller upper bound on $|\det A|$, which would facilitate the application of symbolic methods if the matrix A is filled with integers [ABD97], [BEPP97/99].

Our goal is to enhance the power of numerical approach by preconditioning the input matrix with some novel techniques and by incorporating some symbolic techniques to avoid rounding errors. We refer the reader to [PIMRa], [PIMRTa], [Pa], [Pna], [PMQRa], and [PYa] on various aspects and applications of this approach and to [P91], [P92a], [BP94], [P98], [P01], [EMP04], [WZ07] on many other applications of *symbolic-numerical* methods. In fact, we arrived at our current approach by

looking for preconditioners for matrices arizing in matrix algorithms for polynomial root-finding [PIMRa], [PIMRTa].

At least on first reading, one can assume the more narrow but still highly important task of computing just the sign of the determinant.

1.3 Rounding errors and the certification of the output

Hereafter $\sigma_j(M)$ denotes the jth largest singular value of a $k \times k$ nonsingular matrix M, $j = 1, \dots, k$. M^T denotes its transpose, and M^{-T} denotes $(M^T)^{-1} = (M^{-1})^T$. We write $||M|| = \sigma_1(M)$ for its 2-norm, $\operatorname{cond} M = \sigma_1(M)/\sigma_k(M) = ||M||\ ||M^{-1}||$ for its condition number. A matrix M is *normalized* if $||M|| = 1$ and is *ill* (resp. *well*) *conditioned* if $\operatorname{cond} M$ is large (resp. not large). I_k is the $k \times k$ identity matrix. $\operatorname{diag}(B, C)$ is the 2×2 block diagonal matrix with the diagonal blocks B and C. ϵ is the unit roundoff for a fixed precision of computing.

In numerical computations rounding errors can corrupt the value and even the sign of the determinant. For the output error we have the known upper bounds $e_{\det}(A) \le c_{\det} d_+(A)\epsilon$ where d_+ is the Hadamard's bound on $|\det A|$ and $c_{\det}$ denotes a positive constant. The Hadamard's bound is large and overly pessimistic [ABM99], [EGV00, Corollary 6.2], and so is the above bound on $e_{\det}(A)$.

Trying to yield a better bound, we can rely on PLU or QR factorization of the matrix A or its SVD. (Recall that $\det A = (\det B) \det C$ if $A = BC$ as well as if $A = \operatorname{diag}(B, C)$, whereas the determinants of the permutation, diagonal, triangular, and orthogonal matrices are readily available.)

We can certify the sign of $\det A$ computed based on numerical factorization if $\sigma_n(A)$, the smallest singular value of an $n \times n$ matrix A, exceeds an upper estimate $e_f(A)$ for the norm of its factorization error [PY99/01]. (Indeed to change the sign of $\det A$ by continuously perturbing a nonsingular matrix A, we must pass through a singular matrix, but this cannot occur in the open $\sigma_n(A)$-neighborhood [GL96, Theorem 2.5.3].)

We have the estimate $e_f(A) \le c\sigma_1(A)\epsilon$ for such errors in PLU and QR factorizations and in the SVD where c is a positive constant [GL96, Sections 2.4, 5.2, and 8.6], [S98, Sections 3.3, 3.4, and 4.2], [S01, Section 3.3], [H02]. Therefore, we can certify the sign if $c\sigma_1(A)\epsilon < \sigma_n(A)$, that is if $\operatorname{cond} A = \sigma_1(A)/\sigma_n(A) < 1/(c\epsilon)$. The constant c decreases in a posteriori versus a priori estimates and for the QR factorization versus the SVD. It further decreases for PLU factorization as well as a rational version of the modified Gram–Schmidt algorithm in [C92], which avoids computing square roots.

In some cases one can trust just some heuristic correctness verification, e.g., compute the matrix A^{-1} and its determinant $\det(A^{-1})$ and then verify whether $(\det A)\det(A^{-1}) = 1$.

1.4 Our approach and the organization of the paper

By relying on the above observations, we reduce the determinant computation to the case of better conditioned matrices. We achieve this by combining our novel techniques of additive preconditioning with some nontrivial symbolic-numerical techniques of matrix computations, which we describe for a general input matrix. For sparse and/or structured matrices (see [P01], [GS92], [PR93], and the bibliography therein and in [VVGM05]), one can dramatically accelerate the computations (see [PIMRa, Examples 4.1–4.6] and [Pa, Section 11]).

Here is the flowchart of our algorithm for computing $\det A$ for an $n \times n$ matrix A.

Flowchart 1.1. Determinant.

1. *Apply the known effective condition estimators to the matrix A. If it is well conditioned, compute its numerical factorization, output* $\det A$ *and stop.*
2. *Otherwise select a positive integer $r < n$, generate a pair of random $n \times r$ matrices U and V and compute an additive preconditioner UV^T, the well conditioned additive modification $C = A + UV^T$ and* $\det C$.
3. *Compute the $r \times r$ aggregate $G = I_r - V^T C^{-1} U$ with a high precision.*
4. *Compute* $\det G$ *and* $\det A = (\det C) \det G$ *and stop.*

We specify, analyze, and extend this algorithm in the next sections. In particular, we specify the nontrivial choice of the integer r for which the matrices C and G are well conditioned, and we compute the matrix G with the precision that grows to the infinity together with $\operatorname{cond} A$. To solve this task we develop a symbolic variant of the classical numerical iterative refinement. Wherever this computation requires higher precision, we employ advanced multiplication/summation algorithms, hereafter referred to as *MSAs*. In particuar in these algorithms we incorporate our compressing summation algorithm from [PMQRa].

We organize our paper as follows. In the next section we cover additive preconditioning. In Section 3 we relate the determinants of the original and preconditioned matrices via matrices of smaller size (A-aggregates), which typically must be computed with high accuracy. We cover this computation in Sections 4 and 5. In Section 6 we comment on the MSAs. In Section 7 we cover our numerical tests. In the Appendix, we elaborate upon some omitted technical details, outline dual A-preconditioning and aggregation, with which we can simplify the computation of determinants for some "hard-to-handle" inputs, and briefly review symbolic algorithms for computing determinants and extensions to computing resultants.

Our numerical tests were designed by the first author and implemented by all authors, mostly by the second and the third authors. Otherwise the paper is due to the first author (like [PY99/01], [PIMRa], [PIMRTa], [PKRK06], [PMQRa], [PYa]).

2 Additive preconditioning

We rely on *additive preconditioning* $A \leftarrow C = A + UV^T$, i.e., we add a matrix UV^T (which typically has a smaller rank) to an ill conditioned matrix A to decrease its condition number. We use the abbreviations *APPs*, *APCs*, *A-modification*, and *A-preconditioning* for additive preprocessors and preconditioners, additive modification, and additive preconditioning, respectively. (An APC is an APP that decreases the condition number.)

Given a nonsingular $n \times n$ matrix A, a positive integer $r < n$, and the r-tail (resp. r-head) of its SVD, that is the r smallest (resp. largest) singular values of the matrix A together with the associated singular spaces, one can immediately define an APC UV^T of the rank r and the A-modification $C = A + UV^T$ such that $\operatorname{cond} C = \sigma_1(A)/\sigma_{n-r}(A)$ (resp. $\operatorname{cond} C = \sigma_{r+1}(A)/\sigma_n(A)$). If both r-head and r-tail are known and if $2r < n$, one can readily obtain the optimal APCs UV^T of a rank $r < n/2$, such that $\operatorname{cond} C = \sigma_{r+1}(A)/\sigma_{n-r}(A)$ [Wa]. This can help even if we just approximate the r-tail and/or r-head because an APC tends to remain an APC in its small-norm perturbation.

One can obtain the r-head and r-tail of the SVD by applying the Lanczos algorithm [GL96, Chapter 9], [S01, Chapter 5], but we propose a less costly choice of an APP UV^T of a rank r which is

a) random (general, sparse, or structured [PIMRa, Emaples 4.1–4.6]),

b) well conditioned, and

c) properly scaled so that the ratio $||A||/||UV^T||$ is neither very large nor very small.

Then according to the analysis and extensive experiments in [PIMR05], [PIMR06], [PIMRa], we are likely to arrive at an A-modification $C = A + UV^T$ with $\operatorname{cond} C$ of the order of $\sigma_1(A)/\sigma_{n-r}(A)$.

We can apply the effective norm and condition estimators in [GL96, Section 3.5.4] and [S98, Section 5.3] for computing APPs under rules b) and c), as well as at the other stages of A-preconditioning. E.g., we can check if $\operatorname{cond} C$ is as small as desired, and if it is not, we can recompute the A-modification C for new generators U and V chosen either again according to the rules a)–c) or (with more work and more confidence in success) as follows,

$$(U \leftarrow Q(C^{-1}U), \quad V \leftarrow Q(C^{-T}V)). \tag{2.1}$$

Here $Q(M)$ denotes the $k \times l$ Q-factor in the QR factorization of a $k \times l$ matrix M of the full rank. Computing the aggregates $C^{-1}U$ and $C^{-T}V$ is simpler where the matrix C is better conditioned. The efficiency of these policies and their (more than) sufficiency for our tasks in this paper have been confirmed by our extensive tests (see Section 8 and [PIMR06], [PIMRa]). Moreover, these policies can be further advanced (see [Pna], [Wa], or our Appendix B).

3 APC-based factorization of determinants

Our next goal is to compute $\det A$ for an ill conditioned matrix A provided we have computed a well conditioned A-modification $C = A + UV^T$ and $\det C$.

Theorem 3.1. *[H64]. For $n \times r$ matrices U and V and $n \times n$ matrix A, let the matrix $C = A + UV^T$ be nonsingular. Then*

$$\det A = (\det C) \det G \quad \text{for} \quad G = I_r - V^T C^{-1} U. \tag{3.1}$$

Proof. Obtain equations (3.1) by combining the factorizations

$$\begin{aligned} W = \begin{pmatrix} C & U \\ V^T & I_r \end{pmatrix} &= \begin{pmatrix} I_n & U \\ 0 & I_r \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & I_r \end{pmatrix} \begin{pmatrix} I_n & 0 \\ V^T & I_r \end{pmatrix} \\ &= \begin{pmatrix} I_n & 0 \\ V^T C^{-1} & I_r \end{pmatrix} \begin{pmatrix} C & 0 \\ 0 & G \end{pmatrix} \begin{pmatrix} I_n & C^{-1}U \\ 0 & I_r \end{pmatrix}. \end{aligned}$$

□

The $r \times r$ matrix $G = I_r - V^T C^{-1} U$ is known as the Gauss transform of the matrix W and as the Schur complement of its block C [GL96, pages 95, 103]; for $r < n$ we call it an *A-aggregate* of the matrix A. We call *A-aggregation* the transition to computations with A-aggregate. We combine A-aggregation with A-preconditioning, but otherwise it is a natural descendant of the multiplicative *aggregation methods* in [MP80], which in the 1980s evolved into the *Algebraic Multigrid*. One can obtain further insight into aggregation by comparing these two aggregation approaches with the techniques of the Null Aggregation in [Pna] and of *trilinear aggregating* [P84]. The latter technique has been an indispensible ingredient in the design of the currently fastest algorithms for $n \times n$ matrix multiplication, both fastest theoretically for immense dimensions n [CW90] and fastest for moderate dimensions n from 20 to, say, 10^{20} [P84], [LPS92], which have efficient numerical implementations [K99], [K04].

4 Cancellation of the most significant bits in the A-aggregates

If the matrix C is well conditioned, we readily yield $\det C$, and it remains to compute the A-aggregate G and then its determinant. The difficulty of this computation largely depends on the norm and condition number of the output matrix G. They can be estimated based on the following result.

Theorem 4.1. *[Pa, Theorem 7.3]. For positive integers n and r, a normalized $n \times n$ matrix A, and a pair of matrices U and V of size $n \times r$, write $C = A + UV^T$ and $G = I_r - V^T C^{-1} U$. Suppose the matrices A and $C = A + UV^T$ have full rank $\rho \geq r$. Then the matrix G is nonsingular, and we have*

$$\sigma_j(A^{-1})\sigma_-^2(C) - \sigma_-(C) \leq \sigma_j(G^{-1}) \leq \sigma_j(A^{-1})\sigma_+^2(C) + \sigma_+(C)$$

for $\sigma_-(C) = \sigma_\rho(C)$, $\sigma_+(C) = \sigma_1(C) \leq 2$, $\sigma_j(A^{-1}) = 1/\sigma_{\rho-j+1}(A)$, $j = 1, \ldots, r$.

Proof. See Appendix B. □

Corollary 4.1. *Under the assumption of Theorem 4.1 we have*

$$\operatorname{cond} G = \operatorname{cond}(G^{-1}) \le (\operatorname{cond} C)(\sigma_1(A^{-1})\sigma_+(C)+1)/(\sigma_r(A^{-1})\sigma_-(C)-1),$$

$$||G|| = \sigma_1(G) = 1/\sigma_j(G^{-1}) \le 1/(\sigma_r(A^{-1})\sigma_-^2(C)-\sigma_-(C)).$$

We assume that A is a nonsingular ill conditioned matrix, $\rho = n$, and C is a well conditioned matrix. Then, in virtue of the theorem, the matrix G is well conditioned provided the ratio $\sigma_{n-r+1}(A)/\sigma_n(A)$ is not large (observe that $\operatorname{cond} G = 1$ if $r = 1$). In this case, given the matrix G, we can readily compute $\det G$.

The theorem also implies, however, that all singular values of the matrix G and therefore its norm are small in our case. Thus we must compute the matrix G within a small absolute error norm. Such a task leads to numerical problems because the diagonal entries of the matrix $V^TC^{-1}U$ are close to one and are nearly cancelled when we subtract it from the matrix I_r.

5 Symbolic-numerical iterative refinement

We obtain the A-aggregate $G = I_r - V^TC^{-1}U$ by computing

- the matrix $W = C^{-1}U$ (by solving the matrix equation $CW = U$) and
- the matrix $I_r - V^TW$.

We compute the matrix W with high precision by extending the Wilkinson's iterative refinement in [GL96, Section 3.5.3], [S98, Sections 3.3.4 and 3.4.5], and [H02, Chapter 11]. In its classical form this algorithm approximates the matrix $W = C^{-1}U$ with at most double precision. This is generally insufficient in our case. Thus we continue the steps of iterative refinement in the fashion of Hensel's lifting in [MC79], [D82] to improve the approximation further. As in the latter symbolic algorithm, we represent the output values as the sums of fixed-precision numbers.

Let us specify and analyze the extended iterative refinement of the matrices $W = \sum_{i=0}^{k} W_i$ and $G = I_r - V^TW = I_r + \sum_{i=1}^{k} F_i$. Fix a sufficiently large integer k, write $U_0 = U$ and $G_0 = I_r$, and successively compute the matrices $W_i \leftarrow C^{-1}U_i$, $U_{i+1} \leftarrow U_i - CW_i$, $F_i \leftarrow -V^TW_i$, and $G_{i+1} \leftarrow G_i + F_i$ for $i = 0, 1, \ldots, k$. (For comparison, the classical algorithm begins with a crude approximation $W_0 \approx W = C^{-1}U$ and recursively computes the matrices $U_i \leftarrow U - CW_{i-1}$, $E_i \leftarrow C^{-1}U_i$, and $W_i \leftarrow W_{i-1} + E_i$ for $i = 0, 1, \ldots, k$, so that the norm $||W_i - W||$ recursively decreases until it reaches the limit posed by rounding errors.)

Theorem 4.1 defines a small upper bound on the norm $||G||$ if A is an ill conditioned matrix, whereas the matrix C is well conditioned. Therefore, we can have $G_i \approx 0$ for $i = 0, 1, \ldots, k$ and some positive integer k. We do not need to store such matrices G_i. Furthermore, at the ith step of iterative refinement for $i \le k$ we can overwrite the matrices W_{i-1}, U_i, and F_{i-1} with their updates W_i, U_{i+1}, and F_i, to decrease the memory space.

At the stages of computing the matrices $C \leftarrow A + UV^T$, $U_{i+1} \leftarrow U_i - CW_i$, $F_i \leftarrow -V^T W_i$, and $G_{i+1} \leftarrow G_i + F_i$ for $i = 0, 1, \ldots, k$ we seek error-free output because even small relative errors can completely corrupt the matrix G. To meet the challenge, we have two tools, that is a) policy of truncation of the entries of the matrices U, V, C, and W_i for all i and b) MSAs.

We can choose any pair of matrices U and V up to a perturbation within a fixed small norm as long as this perturbation keeps the A-modification $C = A + UV^H$ well conditioned. Likewise, in our computation of the matrices C^{-1} and $W_i = C^{-1}U_i$ we can allow any errors within a fixed small norm bound as long as this ensures that the residual norm $u_i = ||U_i||_2$ decreases by at least a fixed factor $1/\theta > 1$ in each iteration (cf. Corollary 5.2). At this point we can apply any direct or iterative algorithm (e.g., Gaussian elimination or Newton's iteration in [PS91], [P01, Chapter 6], [PKRK06]), and if θ is too close to one, we can seek support from the classical numerical iterative refinement.

We vary the matrices U, V, C^{-1}, and W_i for all i to decrease the number of bits in the binary representation of their entries. We first set the entries to zero wherever this is compatible with the above requirements to the matrices. Then we truncate the remaining (nonzero) entries to decrease the number of bits in their representation as much as possible under the same requirements to the matrices.

Let us estimate the errors, the parameter θ, and the precision in computing the residual matrices U_i.

Theorem 5.1. *Assume the subiteration*

$$\begin{aligned} W_i &\leftarrow \mathrm{fl}(C^{-1}U_i) = C^{-1}U_i - E_i \\ U_{i+1} &\leftarrow U_i - CW_i \end{aligned}$$

for $i = 0, 1, \ldots, k$ and $U = U_0$. Then

$$C(W_0 + \cdots + W_k) = U - CE_k.$$

Proof. Due to the assumed equations, we have $CW_i = U_i - U_{i+1}$, $i = 0, 1, \ldots, k-1$. Sum the latter equations to obtain that $C(W_0 + \cdots + W_{k-1}) = U_0 - U_k$. Substitute the equations $U_0 = U$ and $U_k = CW_k + CE_k$ and obtain the theorem. □

The theorem implies that the sum $W_0 + \cdots + W_k$ approximates the matrix $W = C^{-1}U$ with the error matrix $-E_k$.

It remains to show that the error term E_i converges to zero as $i \to \infty$.

Theorem 5.2. *Let $Z_i = 0$ for all i and $u_0 = ||U||$. Assume that $W_i = (C - \tilde{E}_i)^{-1}U_i = C^{-1}U_i - E_i$. Write $e_i = ||E_i||$ and $u_i = ||U_i||$ for all i, $\delta_i = \delta(C, \tilde{E}_i) = 2||\tilde{E}_i||_F \max\{||C^{-1}||^2, ||(C - \tilde{E}_i)^{-1}||^2\}$, $\theta_i = \delta_i||C||$ where $||M||_F = \sqrt{\sum_j \sigma_j^2(C)} = \sqrt{\mathrm{trace}(C^T C)}$ denotes the Frobenius norm of a matrix M, $||M||_F \geq ||M||$. Then we have $e_i \leq \delta_i u_i$ for all i, $e_{i+1} \leq \theta_i e_i$, $u_{i+1} \leq \theta_i u_i$ for $i = 0, 1, \ldots, k-1$.*

Proof. We follow [Pa, Section 8] and begin with some auxiliary results.

Theorem 5.3. *We have $U_{i+1} = CE_i$ and consequently $u_{i+1} \le e_i||C||$ for all i.*

Proof. Pre-multiply the matrix equation $C^{-1}U_i - W_i = E_i$ by C and add the resulting equation to the equation $U_{i+1} - U_i + CW_i = 0$. □

Lemma 5.1. *Let C and $C+E$ be two nonsingular matrices. Then*

$$||(C+E)^- - C^-|| \le ||(C+E)^- - C^-||_F \le 2||E||_F \max\{||C^{-1}||^2,\ ||(C+E)^{-1}||^2\}.$$

Proof. See [GL96, Section 5.5.5]. □

Corollary 5.1. *Assume that $W_i = (C - \tilde{E}_i)^{-1}U_i = C^{-1}U_i - E_i$. Then $e_i \le \delta_i u_i$ where $\delta_i = \delta(C, \tilde{E}_i) = 2||\tilde{E}_i||_F \max\{||C^{-1}||^2, ||(C-\tilde{E}_i)^{-1}||^2\}$.*

Combine Theorem 5.3 and Corollary 5.1 and obtain that $u_{i+1} \le \theta u_i$ and $e_{i+1} \le \theta e_i$ for $\theta_i = \delta_i||C||$ and for all i. Summarize our estimates, and obtain Theorem 5.2. □

The theorem shows linear convergence of the error norms e_i to zero as $i \to \infty$ provided $\theta = \max_i \theta_i < 1$. This implies linear convergence of the matrices $W_0 + \cdots + W_i$ to W, $U_0 + \cdots + U_i$ to U, $F_0 + \cdots + F_i$ to F, and G_{i+1} to G.

Next we estimate θ. We assume dealing with a well conditioned matrix C, and so the ratios $r_i = ||\tilde{E}_i||_F/||C||_F$ are small and $\mathrm{cond}(C - \tilde{E}_i) \approx \mathrm{cond}\, C$ (cf. [GL96, Section 3.3], [S98, Theorem 3.4.9], [H02]). In this case the values

$$\theta_i = \delta_i||C|| = 2r_i \max\{(\mathrm{cond}\, C)^2, (\mathrm{cond}(C - E_i))^2\}||C||_F/||C|| \approx$$

$$2(\mathrm{cond}\, C)^2 r_i||C||_F/||C|| \le 2(\mathrm{cond}\, C)^2 r_i n$$

tend to be significantly less than one.

Finally we estimate precision in our error-free computation of the residual matrices U_i. Hereafter for a finite precision binary number $b = \sigma \sum_{k=t}^{s} b_k 2^k$, where $\sigma = 1$ or $\sigma = -1$ and each b_i is zero or one, we write $t(b) = t$, $s(b) = s = \lfloor \log_2 |b| \rfloor$, $p(b) = t - s + 1$, so that $p(b)$ is the precision in the binary representation of the number b. For a matrix $M = (m_{i,j})_{i,j}$ we write

$$s(M) = \max_{i,j} s(m_{i,j}) = \lfloor \log_2 \max_{i,j} |m_{i,j}| \rfloor \le \lfloor \log_2 ||M||_2 \rfloor,$$

$t(M) = \min_{i,j} t(m_{i,j})$, $p(M) = s(M) - t(M) + 1$, so that each entry of the matrix M is the sum of some powers 2^k for integers k selected in the range $[t(M), s(M)]$. Furthermore, we write

$$p_i = \lfloor \log_2(||W_i||_2/e_i) \rfloor \tag{5.1}$$

to represent the relative precision of approximating the entries of the output matrix W achieved at the ith step of iterative refinement, so that $p_i \ge \lfloor -\log_2 \theta_i \rfloor$.

We readily deduce from Theorem 5.2 and equation (5.1) that

$$s(U_{i+1}) \le s(W_i) + \lceil \log_2 ||C||_2 \rceil - p_i \tag{5.2}$$

for all i. We extend this inequality to the following bound.

Theorem 5.4. *Let the (scaled) matrix C be filled with integers. Then we have $p(U_{i+1}) \le \min\{p(U_i), p(W_i) + \lceil \log_2 ||C||_2 \rceil - p_i\}$ for all i.*

Proof. The theorem follows from the bounds (5.2), $t(U_{i+1}) \ge \min\{t(U_i), t(CW_i)\}$ (implied by the equation $U_{i+1} = U_i - CW_i$), and $t(CW_i) \ge t(W_i)$ (which holds because the matrix C is filled with integers). □

Corollary 5.2. *Let Theorems 5.2 and 5.4 hold and let $p(W_i) - p_i \le p$ and $\theta_i < \theta$ for all i and for some $p < \infty$ and $\theta < 1$, so that $e_{i+1} \le \theta e_i$ and $u_{i+1} \le \theta u_i$. Then $p(U_{i+1}) \le \max\{p(U), p + \lceil \log_2 ||C||_2 \rceil\}$.*

The corollary shows that the precision required for the entries of all residual matrices U_i is uniformly bounded if the entries of the matrices C and U are given with bounded precision and if the progress in the iterative refinement is ensured where $p(W_i) - p_i \le p < \infty$ for all i, e.g., if the precision range for the entries of the matrices W_i is uniformly bounded for all i.

Realistically we do not know a priori for which minimum precision bound p the progress in iterative refinement is ensured, but we can find this dynamically, by beginning with the IEEE standard double precision and then increasing it recursively until convergence is observed. MSAs can readily handle any reasonable growth of the precision, but in our tests the growth was quite limited.

If the ratio $\sigma_{n-r}(A)/\sigma_n(A)$ is large, then the A-aggregate G is ill conditioned. To compute $\det G$, we can reapply A-preconditioning to the matrix G and continue this process recursively until we arrive at a well conditioned A-aggregate. All ill conditioned A-aggregates G, however, must be computed with a high precision (at a higher cost) to enable subsequent correct computation of their determinants. For a large class of input matrices we can counter this deficiency by applying the dual A-aggregation (see Appendix A).

6 Multiplication/summation algorithms

Effective MSAs in [H02], [LDB02], [DH03], [ORO05], [ORO05a], and the bibliography therein compute the sum and products with double or k-fold precision for any k, but the computation slows down for $k > 2$. Our computation of A-aggregates, however, has lead us to computing the sums $s = t_1 + \cdots + t_h$ that nearly vanish compared to $\max_j |t_j|$. Moreover, we need some of these sums error-free but wish to avoid the slow down of computing with multi-precision.

Let us comments on how our MSAs achieve this. In our comments "addition" usually stands for "addition or subtraction", "dpn" and "dpn-1" are our abbreviations for "number represented with the IEEE standard double precision", and "dpn-ν" is the set of ν such dpns, which together implicitly represent their sum. We can represent a $((p+1)\nu)$-bit floating point number as the dpn-ν where $p+1$ is the double precision.

The MSAs incorporate the Dekker's and Veltkamp's algorithms in [D71] to compute the product of a dpn-μ and a dpn-ν error-free as a dpn-γ for $\gamma \le 2\mu\nu$. To add a dpn-μ and a dpn-ν we just combine them into a dpn-$(\mu + \nu)$.

Periodically we perform *compressing summations*. Such a summation is an error-free transforms of a *swelling sum*, that is a dpn-μ with an excessively large integer μ, into a dpn-ν for the nearly minimum ν.

Contrary to the most sacred rule of numerical computing, our compressing summation removes many leading most significant bits of the summands, but does this without affecting the output sum. Technically, we achieve this by combining Dekker's splitting algorithm in [D71] with the techniques of real modular reduction from [P92] (see also [EPY98]).

We adopt compressing summation from [PMQRa], where we perform some sequences of usual floating-point additions interrupted with the computation of the exponent of the current floating-point approximation of the output sum. We compute this exponent every time when we update the sum, and we always add at least $\theta p - \log_2 h - O(1)$ new correct bits to the sum in every updating. Here $\theta = 1$ or $\theta = 2$ depending on our choice of the basic subroutine for floating-point summation that we employ in our MSAs.

Remark 6.1. *Accessing exponents of floating point numbers can be inexpensive. The IEEE floating point standard defines the function* logb(x) *to extract the significand and exponent of a floating point number. Floating point units (FPUs) in Intel's Pentium processor family provide hardware implementations of an instruction, FXTRACT, offering a superset of the* logb(x) *functionality [I01]. For double precision floating point numbers, the FPU of the Pentium 4 processor can execute the FXTRACT instruction in 12 cycles [F04] (almost three times as fast as the same FPU handles division). Because FXTRACT is a floating point instruction, the FPU can overlap the early cycles of FXTRACT with late cycles of various other floating point instructions when they immediately precede FXTRACT, thereby allowing further speed up [F04].*

Remark 6.2. *We apply MSAs essentially to computing sums and dot products, but in principle one can apply them to the evaluation of any polynomial and, in combination with approximating the reciprocals and with error analysis, to approximate evaluation of rational functions.*

7 Numerical tests for computing determinants

We perform most of our determinant algorithm numerically, which enables decisive acceleration versus the known symbolic algorithms for determinants as soon as we match the level of their highly developed implementations. This is our ultimate goal.

So far we tested the accuracy of our fast numerical algorithm versus the Matlab's Subroutine **det**. We computed determinants with Matlab in two ways, by applying

1. the Matlab's Subroutine **det** and
2. our algorithm based on factorization (3.1).

As in [PY99/01], we used nonsingular matrices $A = PML$ where P were permutation matrices, each swapping k random pairs of the rows of the matrix A, whereas

L and M^T denoted random $n \times n$ lower triangular matrices with unit diagonal entries and with integer subdiagonal entries randomly sampled from the line intervals $[-\gamma, \gamma]$ for a fixed positive γ. It followed that $\det A = (-1)^k$. We generated such matrices for $k = 2n$ and $k = 2n - 1$ and for $\gamma \geq 5,000$ and for $n = 4, 8, 16, 32, 64$.

We generated random APCs UV^T of recursively increasing ranks $r = 1, 2, \ldots$ until we arrived at a well conditioned A-modification $C = A + UV^T$. More precisely, we generated two random $n \times r$ unitary matrices U and V, then truncated their entries to represent them with the precision of 20 bits, denoted the resulting matrices $\tilde{U}$ and $\tilde{V}$, and computed the APP $\widehat{U}\widehat{V}^T = 2^q \tilde{U}\tilde{V}^T$ and the A-modification $\tilde{C} = A + \widehat{U}\widehat{V}^T$ for an integer q such that $1/2 < ||UV^T||/||A|| \leq 2$. If $\operatorname{cond} \tilde{C}$ was small enough, we accepted the APP $\widehat{U}\widehat{V}^T$ as the desired APC UV^T. Otherwise we regenerated APP in the same way. If this did not produce a desired APP, we recomputed an APC according to the recipe (2.1). If this did not help either, we incremented r by one and repeated the computations. We encountered overflows and underflows for larger n but overcame the problems by simultaneously scaling the matrix U by factor 2^k and the matrix V by factor 2^{-k} for an appropriate integer k and/or by temporarily scaling the matrices I_r and U by the same factor 2^h for an appropriate integer h.

The selected matrices A were ill conditioned for all integers n in our range (with $\operatorname{cond} A$ quite steadily in the range from 10^{17} to 10^{25} for all n). The numerical subroutines in Matlab performed poorly for the matrices of the selected class. They have lost competition in accuracy not only to the slower symbolic subroutines in MAPLE but also to our numerical tests. Already for $n = 4$ and $\gamma = 5,000$, the Matlab's numerical outputs had wrong sign in over 45% out of 100,000 runs and were off from the true value of $\det A$ by the factor of two or more in over 90% of the runs. As we expected, our algorithms have outperformed the Subroutine **det**. Although we still relied on the standard double precision computations, our algorithms always output the correct sign and approximated the value of the determinant with relative errors within 0.001 in all our runs for $n = 4$, 8, 16, 32, 64 and for the same value of γ.

8 Discussion

We have described our approach and demonstrated its power for computing determinants. This is just the tip of an iceberg. The approach has been extended to facilitating the solution of linear systems of equations in [Pa] (by combining our present techniques with the Sherman–Morrison-Woodbury formula [GL96, page 50]), to computing vectors in and bases for the null spaces of a matrix in [Pna] (with further applications to computing and refining APCs), and to approximation of eigenvectors in [PYa]. Various further extensions and applications (e.g., to root-finding for polynomials and systems of polynomials) have been pointed out in these papers. Elaboration upon these directions is among our further subjects.

Regarding determinants, our next goals include

- advanced numerical implementation of our algorithms to the level at which they can compete or be combined with the symbolic implementations in Maple, Magma, Cocoa, LinBox, and NTL

- extension of our approach to cover the cases of hard-to-handle input matrices, one of the directions being the dual A-preconditioning and dual A-aggregation, which can be combined with our present approach (see Appendix A), and
- elaboration upon our techniques in the case of sparse and structured input matrices.

Appendix

A Dual A-preconditioning and A-aggregation

Dual A-preconditioning and dual A-aggregation is an example of a natural extension of our approach, which enables us to use fewer and simpler matrix inversions.

By applying A-preconditioning to the inverse matrix A^{-1} without explicitly computing it, we obtain the factorization

$$\det A = (\det H)\ \det((C_-)^{-1}) \tag{A.1}$$

where $H = I_q + U^T AV$ is the *dual A-aggregate*, $C_- = A^{-1} + VU^T$ is the *dual A-modification*, and $(C_-)^{-1} = A - AVH^{-1}U^T A$.

Our analysis of A-preconditioning can be extended to the inverse matrix A^{-1}, and so the matrix C_- is likely to be well conditioned if a random well conditioned dual APP VU^T is scaled so that the ratio $||A^{-1}||/||VU^T||$ is neither large nor small and if the ratio $\sigma_{q+1}(A)/\sigma_n(A)$ is not large for $q = \mathrm{rank}(VU^T)$. Then our original determinant problem for the matrix A is reduced to the same problem for the dual $q \times q$ A-aggregate H.

Now, by extending Theorem 4.1, we deduce that if the matrix C_- is well conditioned, then $\mathrm{cond}\, H$ has the order of the ratio $\sigma_1(A)/\sigma_q(A)$, that is unless this ratio is large, the dual Schur aggregate H is well conditioned. Then again, its computation leads to numerical problems, which we overcome by applying MSAs. In the dual case we compute the matrix H by using no matrix inversions, and so we need no iterative refinements.

If the ratio $\sigma_1(A)/\sigma_q(A)$ is large, then the dual A-aggregate H is ill conditioned. To compute $\det H$, we can reapply dual A-preconditioning and dual A-aggregation to the matrix H.

To our benefit, we can compute the matrix H error-free by applying MSAs. A delicate point is its inversion for computing the matrix $(C_-)^{-1}$. Since the latter matrix is well conditioned, it is sufficient to output its entries just with double precision. This means a milder requirement on the accuracy of inverting the matrix H. Having its approximate inverse, however, we can rapidly refine it with Newton's iteration.

Computing an approximate inverse of a matrix H is equivalent to solving q linear systems with this matrix. For this task, we can apply effective classical and modern algorithms in [GL96], [S98], [DDS98], [DER86], [PKRK06]. For $q < n$, the problem size decreases versus the primal aggregation. For smaller ranks q, the GMRES

and Conjugate Gradient algorithms are effective even for ill conditioned matrices H [GL96], [DDS98].

Instead of primal A-preconditioning and A-aggregation for the matrix A we can equivalently apply the dual ones to its inverse. This can simplify the determinant task if its solution cost dominates that of the inversion.

B Computing APCs via inflation, aggregation, and compression

Computation of the aggregates $C^{-1}U$ and $C^{-T}V$ is simpler where the matrix C is better conditioned. We can more readily obtain a well conditioned A-modification $C = A + UV^T$ by choosing an APP UV^T of a larger rank. Then we can modify the transform in (2.1) to obtain an APC of a smaller rank for which we still have $\mathrm{cond}_2 C$ nicely bounded (cf. [PIMRa], [Pna], [Wa]). Specifically, assuming that the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ is not large, whereas $\sigma_{n-r}(A) \gg \sigma_{n-r+1}(A)$ for an $n \times n$ ill-conditioned input matrix A, we can proceed as follows.

1. **(Generation of an inflated APC.)** Generate an APC UV^T of a larger rank, say, of a rank h exceeding $2r$.

2. **(Aggregation.)** Compute two properly scaled and well conditioned matrix bases $T(U)$ and $T(V)$ for the singular spaces of the matrices $AC^{-}U$ and $A^TC^{-T}V$, respectively, associated with the r smallest singular values of these matrices.

3. **(Compression.)** Update the generators

$$U \leftarrow Q(C^{-1}UT(U)) \quad \text{and} \quad V \leftarrow Q(C^{-T}VT(V)).$$

   Output them and the new APC UV^T.

These recipes in [Pna] redirect an approach in [Wa] motivated by [PIMRa]. The efficiency of the recipes has been confirmed by extensive tests in [PIMRa] and [Wa].

C Proof of Theorem 4.1

For the reader's convenience we reproduce the proof of [Pa, Theorem 7.3]. We simplify it to our case of square real input matrices A.

We begin with some auxiliary results. The following two equations are immediately verified.

$$A = C(I_n - C^{-1}UV^T),\ A^{-1} = (I_n - C^{-1}UV^T)^{-1}C^{-1}. \tag{C.1}$$

Next recall the SMW formula for matrix inversion by Sherman–Morrison–Woodbury (cf. [GL96, page 50]) $(C - UV^T)^{-1} = C^{-1} + C^{-1}US^{-1}V^TC^{-1}$. Substitute $C \leftarrow I_n$ and $U \leftarrow C^{-1}U$ into this formula and obtain that

$$(I_n - C^{-1}UV^T)^{-1} = I_n + C^{-1}U(I_r - V^TC^{-1}U)^{-1}V^T. \tag{C.2}$$

Theorem C.1. *Let W denote an $m \times n$ matrix of full rank $\rho = \min\{m, n\}$. Write $\sigma_+(W) = \sigma_1(W)$, $\sigma_-(W) = \sigma_\rho(W)$. Then we have $\sigma_j(M)\sigma_-(W) \le \sigma_j(MW) \le \sigma_j(M)\sigma_+(W)$ and $\sigma_j(N)\sigma_-(W) \le \sigma_j(WN) \le \sigma_j(N)\sigma_+(W)$ for $j = 1, \ldots, \rho$ and $\rho \times \rho$ matrices M and N.*

Proof. Since singular values are invariant in multiplication by a unitary matrix, it is sufficient to consider the case of a positive diagonal matrix W. In this case the claimed bounds readily follow from the Courant–Fischer Minimax Characterization [GL96, Theorem 8.1.2], [S01, Theorem 3.3.2]. □

Our next theorem is a special case of [S01, Theorem 3.3.3] where $E = I_n$.

Theorem C.2. *We have $\sigma_j(W) - 1 \le \sigma_j(W + I_n) \le \sigma_j(W) + 1$ for an $n \times n$ matrix W and for $j = 1, 2, \ldots, n$.*

Proof of Theorem 4.1. Deduce from equation (C.1) that the matrix $G_n = I_n - C^{-1}UV^T$ is nonsingular. So is the matrix G as well because $\det G = \det G_n$ [H64, Exercise 1.14].

Next combine equation (C.1) with Theorem C.1 for $M = G_n^{-1}$, $W = C^{-1}$, and $A^{-1} = MW$, to obtain that

$$\sigma_j(G_n^{-1})\sigma_-(C^{-1}) \le \sigma_j(A^{-1}) \le \sigma_j(G_n^{-1})\sigma_+(C^{-1}) \text{ for } j = 1, \ldots, \rho.$$

Substitute $\sigma_-(C^{-1}) = 1/\sigma_+(C)$ and $\sigma_+(C^{-1}) = 1/\sigma_-(C)$ and obtain that

$$\sigma_j(A^{-1})\sigma_-(C) \le \sigma_j(G_n^{-1}) \le \sigma_j(A^{-1})\sigma_+(C) \text{ for } j = 1, \ldots, \rho. \tag{C.3}$$

Combine Theorem C.1 for $W = C^{-1}U$ and $N = G^{-1}$ with the equations and inequalities $\sigma_j(C^{-1}UG^{-1}V^T) = \sigma_j(C^{-1}UG^{-1})$ for $j = 1, \ldots, r$, $\sigma_-(C^{-1}U) \ge \sigma_-(C^{-1}) = 1/\sigma_+(C)$, and $\sigma_+(C^{-1}U) \le \sigma_-(C^+) = 1/\sigma_-(C)$ to deduce that

$$\sigma_j(G^{-1})/\sigma_+(C) \le \sigma_j(C^{-1}UG^{-1}V^T) \le \sigma_j(G^{-1})/\sigma_-(C) \text{ for } j = 1, \ldots, r.$$

Combine the latter bounds with Theorem C.2 for $W = C^{-1}UG^{-1}V^T$ and equation (C.2) to deduce that

$$\sigma_j(G^{-1})/\sigma_+(C) - 1 \le \sigma_j(G_n^{-1}) \le \sigma_j(G^{-1})/\sigma_-(C) + 1$$

and therefore

$$(\sigma_j(G_n^{-1}) - 1)\sigma_-(C) \le \sigma_j(G^{-1}) \le (\sigma_j(G_n^{-1}) + 1)\sigma_+(C) \text{ for } j = 1, \ldots, r.$$

Combine this equation with equation (C.3) and obtain the claimed bounds of Theorem 4.1.

D Symbolic computation of determinants

We have already cited advanced implementation of symbolic algorithms for determinants in Maple, Magma, Cocoa, LinBox and NTL. In this appendix we briefly review the main approaches to symbolic computation of determinants.

One can begin with computing the determinant of an integer matrix A modulo sufficiently many reasonably small primes $p_1, \dots, p_k$ whose product exceeds $2|\det A|$. The most common way is to employ the PLU triangular factorization of the input matrix with pivoting (cf. [BEPP97/99], [PY99/01], and the bibliography therein). The Wiedemann and block Wiedemann alternative symbolic algorithms in [W86], [C94] compute $\det M$ modulo such primes as the x-free term of the characteristic polynomial $\det(M - xI)$ obtained from the Krylov sequence of the matrix M. At this stage the computations are performed with a lower precision of $\lceil \log p_i \rceil$ for $i = 1, \dots, k$. Then the integer $\det A$ can be recovered readily by means of the Chinese remainder algorithm.

In [P87, Appendix] and [P88] it was proposed to compute the determinant $\det M$ of an $n \times n$ general integer matrix M by solving the linear system $M\mathbf{y} = \mathbf{v}$ for a random integer vector $\mathbf{v}$ and to employ Hensel's lifting for the solution. This approach was ressurected and advanced in [ABM99] and [EGV00] to support the randomized Monte Carlo computation of the determinant by using $(n^d \log |M|)^{1+o(1)}$ bit operations for $d = 3$ for the average input matrix M and $d = 7/2$ for the worst case input matrix.

The worst case Monte Carlo exponent $d = 7/2$ of [EGV00] was successively decreased to the Las Vegas exponents $d = 10/3$ in [KV01, Theorem 2], based on an adapted block Wiedemann algorithm, $d = 16/5$ in [P02, Theorem 5.1(I)], based on the acceleration of the stage of block Hankel computations by exploiting the displacement structure of the block Hankel matrices, and $d = 3$ in Storjohann 2005 [S05], based on high order Newton's lifting.

One can decrease all of the above exponents d by incorporating the asymptotically fast algorithms in [CW90] for $n \times n$ matrix multiplication, but we ignore this option as practically invalid due to the huge overhead constants.

Likewise one can decrease the exponent $10/3$ in [KV01] to $16/5$ by incorporating the LKS block half-gcd algorithm (due to Lehmer, Knuth, and Schönhage [B03]), but again this would produce a practically invalid algorithm, whereas with exploiting displacement structure in [P02] and (even more so) with application of Hensel's lifting in [P04], the block Hankel bottleneck stage is accelerated and remains practically valid, and the same applies to the entire block Wiedemann algorithm.

The customary numerical and symbolic algorithms based on the LUP or QR factorization of the input matrix only support the exponent $d = 4$, but they are deterministic and for matrices of a moderate size perform faster and can be the methods of choice.

For computing the determinants of sparse and/or structured matrices A one can obtain dramatic and practically valid acceleration based on either the MBA algorithm due to Morf 1974 and 1980 and Bitmeed and Anderson 1980 (cf. [P01, Chapter 5]) or the Wiedemann algorithm. Both algorithms output $\det A$ and the vector $A^{-1}\mathbf{b}$.

Wiedemann's algorithm supports the exponent $d = \nu + 1$ where the input matrix

can be multiplied by a vector in $n^{\nu+o(1)}$ arithmetic operations for $\nu \geq 1$; in particular $\nu = 1$ for Toeplitz input matrices. The MBA algorithm supports the same cost bound where the latter property holds for the inverses of the input matrix and its leading principal submatrices as well. In other words, the MBA algorithm loses its efficiency where the structure and/or sparseness of the input matrix is not readily extended to its inverse, which is the case for the multivariate polynomial resultants. In this case faster symbolic solution is computed by means of an adapted (block) Wiedemann algorithm (see [EP03/05]).

E Extension to the resultants and polynomial systems of equations

The roots of a system of multivariate polynomial equations can be expressed as the roots of the determinant of the associated resultant or Newton's structured matrices [EP03/05, Sections 5–7]. With due caution to numerical stability problems, we can combine the known iterative root-finders (e.g., Müller's, Laguerre's discrete, or Newton's) with the algorithms in the previous section for computing determinants and possibly with the partial derivative theorem. To support the Newton's method we can apply the equation $\frac{d(A^{-1})}{dx} = -A^{-1}\frac{dA}{dx}A^{-1}$ to the resultant matrix $A = A(x)$.

The known rounding error estimates for computing determinants (based on the Hadamard's bound), however, are overly pessimistic, particularly where the determinants vanish. Numerically it can be more effective to direct the iteration, e.g. to the annihilation of the smallest singular value of the resultant or Newton's matrix.

References

[ABD97] F. Avnaim, J.-D. Boissonnat, 0. Devillers, F. P. Preparata, M. Yvinec, Evaluating Signs of Determinants Using Single-Precision Arithmetic, *Algorithmica*, **17**, 111–132, 1997.

[ABM99] J. Abbott, M. Bronstein, T. Mulders. Fast Deterministic Computation of the Determinants of Dense Matrices, *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, 197–204, ACM Press, New York, 1999.

[AF92] D. Avis, K. Fukuda, A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra, *Discrete Computational Geometry*, **8**, 295–313, 1992.

[B69] I. Babuška, Numerical Stability in Mathematical Analysis, *Information Processing*, **68**, (*Proceedings of 1968 IFIP Congress*, Vol. I), 11–23, North-Holland, Amsterdam, 1969.

[B03] D. J. Bernstein, Fast Multiplication and Its Applications, to be printed in *Algorithmic Number Theory* (edited by Joe Buhler, Peter Stevenhagen), **44**, Mathematical Sciences Research Institute Publications, Cambridge University Press. Availabe from http://cr.yp.to/papers.html

[BBC93] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1993.

[BEPP97/99] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science*, **210**, **1**, 173–197, 1999. Proceedings version in *Proceedings of 13th Annual ACM Symposium on Computational Geometry*, 174–182, ACM Press, New York, 1997.

[BKM95] C. Burnikel, J. Könnemann, K. Mehlhorn, S. Naher, S. Schirra, C. Uhrig, Exact Geometric Computation in LEDA, *Proceedings of 11th Annual ACM Symposium on Computational Geometry*, C18–C19, 1995. Package available at http://www.mpi-sb.mpg.de/LELA/leda.html.

[BP94] D. Bini, V. Y. Pan, *Polynomial and Matrix Computations, Volume 1, Fundamental Algorithms,* Birkhäuser, Boston, 1994.

[C92] K. L. Clarkson, Safe and Effective Determinant Evaluation, *Proceedings of 33rd Annual IEEE Symposium on Foundations of Computer Science*, 387–395, IEEE Computer Society Press, Los Alamitos, California, 1992.

[C94] D. Coppersmith, Solving Homogeneous Linear Equations over *GF(2)* via Block Wiedemann Algorithm, *Math. of Computation*, **62**, **205**, 333–350, 1994.

[CW90] D. Coppersmith, S. Winograd, Matrix Multiplicaton via Arithmetic Progressions, *J. of Symbolic Computation*, **9**, **3**, 251–280, 1990.

[D61] E. Durand, *Solutions Numêriques des Êquations Algêbriques*, Vol. II, Masson, Paris, 1961.

[D71] T. J. Dekker, A Floating-point Technique for Extending the Available Precision, *Numerische Mathematik*, **18**, 224–242, 1971.

[D82] J. D. Dixon, Exact Solution of Linear Equations Using p-adic Expansions, *Numerische Math.*, **40**, 137–141, 1982.

[DDS98] J. J. Dongarra, I. S. Duff, D. C. Sorensen, H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, SIAM, Philadelphia, 1998.

[DER86] I. S. Duff, A. M. Erisman, J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, England, 1986.

[DH03] J. Demmel, Y. Hida, Accurate and Efficient Floating Point Summation, *SIAM J. on Scientific Computing*, **25**, 1214–1248, 2003.

[E98] I. Z. Emiris, A Complete Implementation for Computing General Dimensional Convex Hulls, *Internat. J. Comput. Geom. Appl.*, **8**, **2**, 223–253, 1998.

[EC95] I. Z. Emiris, J. F. Canny, A General Approach to Removing Degeneracies, *SIAM J. on Computing*, **24**, **3**, 650–664, 1995.

[EGV00] W. Eberly, M. Giesbrecht, G. Villard. On Computing the Determinant and Smith Form of an Integer Matrix. *Proceedings of 41st Annual Symposium on Foundations of Computer Science (FOCS'2000)*, 675–685, IEEE Computer Society Press, Los Alamitos, California, 2000.

[EMP04] I. Z. Emiris, B. Mourrain, V. Y. Pan, editors. Special Issue on Algebraic and Numerical Algorithms, *Theor. Comp. Science*, **315**, **2–3**, 2004.

[EP03/05] I. Z. Emiris, V. Y. Pan, Improved Algorithms for Computing Determinants and Resultants, *J. of Complexity*, **21**, **1**, 43–71, 2005. Proceedings version in *Proceedings of 6th International Workshop on Computer Algebra in Scientific Computing (CASC '03)*, E. W. Mayr, V. G. Ganzha, E. V. Vorozhtzov (editors), 81–94, Technische Univ. München, Germany, 2003.

[EPY98] I. Z. Emiris, V. Y. Pan, Y. Yu, Modular Arithmetic for Linear Algebra Computations in the Real Field, *J. of Symbolic Computation*, **21**, 1–17, 1998.

[F04] Agner Fog, *How to optimize for the Pentium family of microprocessors*, www.agner.org, 1996–2004, last updated 2004-04-16.

[FR94] K. Fukuda, V. Rosta, Combinatorial Face Enumeration in Convex Polytopes, *Comput. Geom. Theory Appl.*, **4**, 191–198, 1994.

[FvW93] S. Fortune, C. J. van Wyk, Efficient Exact Arithmetic for Computational Geometry, *Proceedings of 9th Annual ACM Symposium on Computational Geometry*, 163–172, 1993.

[FvW96] S. Fortune, C. J. van Wyk, Static Analysis Yields Efficient Exact Integer Arithmetic for Computational Geometry, *ACM Transactions on Graphics*, **15**, **3**, 223–248, 1996.

[GL96] G. H. Golub, C. F. Van Loan, *Matrix Computations*, 3rd edition, The Johns Hopkins University Press, Baltimore, Maryland, 1996.

[GS92] J. R. Gilbert, R. Schreiber, Highly Parallel Sparse Cholesky Factorization, *SIAM J. on Scientific Computing*, **13**, 1151–1172, 1992.

[H64] A. S. Householder, *The Theory of Matrices in Numerical Analysis*, Dover, New York, 1964.

[H02] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd edition, SIAM Publications, Philadelphia, 2002.

[I01] *IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture*, (Order Number 245470) Intel Corporation, Mt. Prospect, Illinois, 2001.

[K69/81/98] D. E. Knuth, *The Art of Computer Programming: Volume 2, Seminumerical Algorithms*, Addison-Wesley, Reading, Massachusetts, 1969 (first edition), 1981 (second edition), 1998 (third edition).

[K99] I. Kaporin, A Practical Algorithm for Faster Matrix Multiplication, *Numerical Linear Algebra with Applications*, **6, 8**, 687–700, 1999.

[K04] I. Kaporin, The Aggregation and Cancellation Techniques As a Practical Tool for Faster Matrix Multiplication, *Theoretical Computer Science*, **315**, **2–3**, 469–510, 2004.

[KV01] E. Kaltofen, G. Villard, On the Complexity of Computing Determinants, *Proc. Fifth Asian Symposium on Computer Mathematics (ASCM 2001)*, (Shirayanagi, Kiyoshi and Yokoyama, Kazuhiro, editors), *Lecture Notes Series on Computing*, **9**, 13–27, World Scientific, Singapore, 2001.

[KV04] E. Kaltofen, G. Villard. Computing the Sign or the Value of the Determinant of an Integer Matrix, a Complexity Survey, *J. Computational Applied Math.*, **162**, **1**, 133–146, 2004.

[LDB02] X. Li, J. Demmel, D. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, M. Martin, B. Thompson, T. Tung, D. Yoo, Design, Implementation and Testing of Extended and Mixed Precision BLAS, *ACM Transactions on Mathematical Software*, **28**, 152–205, 2002. http //crd.lbl.gov/x̃iaoye/XBLAS/.

[LPS92] J. Laderman, V. Y. Pan, H. X. Sha, On Practical Algorithms for Accelerated Matrix Multiplication, *Linear Algebra and Its Applications*, **162–164**, 557–588, 1992.

[M30/50] T. Muir, *The Theory of Determinants in Historical Order of Development*, four volumes bound as two (I, 1693–1841; II, 1841–1860; III, 1861–1880; IV, 1881–1900), Dover, New York, 1960; *Contributions to the History of Determinants*, 1900–1920, Blackie and Son, London, 1930 and 1950.

[M55] R. H. Macmillan, A New Method for the Numerical Evaluation of Determinants, *J. Roy. Aeronaut. Soc.*, **59**, 772ff, 1955.

[MA93] R. E. M. Moore, I. 0. Angell, Voronoi Polygons and Polyhedra, *J. Comput. Phys.*, **105**, 301–305, 1993.

[Matlab04] The MathWorks Inc., *Matlab User's Guide*, Version 7, 2004.

[MC79] R. T. Moenck, J. H. Carter, Approximate Algorithms to Derive Exact Solutions to Systems of Linear Equations, *Proceedings of EUROSAM, Lecture Notes in Computer Science*, **72**, 63–73, Springer, Berlin, 1979.

[MP80] W. L. Miranker, V. Y. Pan, Methods of Aggregations, *Linear Algebra and Its Applications*, **29**, 231–257, 1980.

[ORO05] T. Ogita, S. M. Rump, S. Oishi, Accurate Sum and Dot Product, *SIAM Journal on Scientific Computing*, **26**, **6**, 1955–1988, 2005.

[ORO05a] S. M. Rump, T. Ogita, S. Oishi, Accurate Floating-Point Summation, Tech. Report 05.12, *Faculty for Information and Communication Sciences*, Hamburg University of Technology, November, 2005.

[P84] V. Y. Pan, How Can We Speed up Matrix Multiplication? *SIAM Review,* **26**, **3**, 393–415, 1984.

[P87] V. Y. Pan, Complexity of Parallel Matrix Computations, *Theoretical Computer Science*, **54**, 65–85, 1987.

[P88] V. Y. Pan, Computing the Determinant and the Characteristic Polynomials of a Matrix via Solving Linear Systems of Equations, *Information Processing Letters*, **28**, 71–75, 1988.

[P91] V. Y. Pan, Complexity of Algorithms for Linear Systems of Equations, in *Computer Algorithms for Solving Linear Algebraic Equations (State of the Art),* Spedicato, E., Ed., volume 77 of *NATO ASI Series,* Series F: Computer and Systems Sciences, 27–56, Springer, Berlin, 1991, and Academic Press, Dordrecht, the Netherlands, 1992.

[P92] V. Y. Pan, Can We Utilize the Cancelation of the Most Significant Digits? Tech. Report TR 92 061, *The International Computer Science Institute*, Berkeley, California, 1992.

[P92a] V. Y. Pan, Complexity of Computations with Matrices and Polynomials, *SIAM Review*, **34**, **2**, 225–262, 1992.

[P98] V. Y. Pan, Some Recent Algebraic/Numerical Algorithms, *Electronic Proceedings of IMACS/ACA'98*, 1998.
http:www-troja.fjfi.cvut.cz/aca98/sessions/approximate

[P01] V. Y. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Birkhäuser/Springer, Boston/New York, 2001.

[P02] V. Y. Pan, Randomized Acceleration of Fundamental Matrix Computations, *Proceedings of 19th International Symposium on Theoretical Aspects of Computer Science (STACS '02)*, *Lecture Notes in Computer Science*, **2285**, 215–226, Springer, Berlin, 2002.

[P04] V. Y. Pan, On Theoretical and Practical Acceleration of Randomized Computation of the Determinant of an Integer Matrix, *Zapiski Nauchnykh Seminarov POMI*, edited by E. A. Girsch, (in English), **316**, 163–187, St. Petersburg, Russia, 2004.

[Pa] V. Y. Pan, The Schur Aggregation and Extended Iterative Refinement, Technical Report TR 2007, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007.

[PIMR05] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning in Matrix Computations, Technical Report TR 2005 009, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, July 2005.

[PIMR06] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, I. Taj-Eddin, Y. Tang, X. Yan, Additive Preconditioning and Aggregation in Matrix Computations, Technical Report TR 2006 006, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, May 2006.

[PIMRa] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning for Matrix Computations, Technical Report TR 2007 003, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

[PIMRTa] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning and Aggregation in Matrix Computations, Technical Report TR 2007, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, 2007.

[PKRK06] V. Y. Pan, M. Kunin, R. Rosholt, H. Kodal, Homotopic Residual Correction Processes, *Math. of Computation*, **75**, 345–368, 2006.

[Pna] V. Y. Pan, Computations in the Null Spaces with Additive Preconditioning, Technical Report TR 2007, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007.

[PMQRa] V. Y. Pan, B. Murphy, G. Qian, R. E. Rosholt, Error-free Computations via Floating-Point Operations, Technical Report TR 2007, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, 2007.

[PR93] V. Y. Pan, J. Reif, Fast and Efficient Parallel Solution of Sparse Linear Systems, *SIAM J. on Computing*, **22**, **6**, 1227–1250, 1993.

[PS91] V. Y. Pan, R. Schreiber, An Improved Newton Iteration for the Generalized Inverse of a Matrix, with Applications, *SIAM J. on Scientific and Statistical Computing*, **12**, *5*, 1109–1131, 1991.

[PY99/01] V. Y. Pan, Y. Yu, Certification of Numerical Computation of the Sign of the Determinant of a Matrix, *Algorithmica*, **30**, 708–724, 2001. Proc. version in *Proc. 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, 715–724, ACM Press, New York, and SIAM Publications, Philadelphia, 1999.

[PYa] V. Y. Pan, X. Yan, Additive Preconditioning, Eigenspaces, and the Inverse Iteration, Technical Report TR 2007 004, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

[S98] G. W. Stewart, *Matrix Algorithms, Vol I: Basic Decompositions*, SIAM, Philadelphia, 1998.

[S01] G. W. Stewart, *Matrix Algorithms, Vol II: Eigensystems*, SIAM, Philadelphia, 1998 (first edition), 2001 (second edition).

[S05] A. Storjohann, The Shifted Number System for Fast Linear Algebra on Integer Matrices, *J. of Complexity*, **21**, **4**, 609–650, 2005.

[vdV03] H. A. van der Vorst, *Iterative Krylov Methods for Large Linear Systems,* Cambridge University Press, Cambridge, England, 2003.

[VVGM05] R. Vandebril, M. Van Barel, G. Golub, N. Mastronardi, A Bibliography on Semiseparable Matrices, *Calcolo*, **42**, **3**–**4**, 249–270, 2005.

[W86] D. Wiedemann, Solving Sparse Linear Equations over Finite Fields, *IEEE Transactions on Information Theory*, **32**, **1**, 54–62, January, 1986.

[Wa] X. Wang, Affect of Small Rank Modification on the Condition Number of a Matrix, *Computer and Math. (with Applications)*, in print.

[WZ07] *Symbolic-Numeric Computation* (Dongming Wang and Li-Hong Zhi, editors), Birkhäuser, Basel/Boston, 2007.

[Y97] C. Yap, Towards Exact Geometric Computation, *Comput. Geom. Theory Appl.*, **7**, 3–23, 1997.

[YD95] C. K. Yap, T. Dubhe, The Exact Computation Paradigm, in D. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, World Scientific Press, Singapore, 1995.