2007

TR-2007022: The Schur Aggregation for Linear Systems and Determinants

V. Y. Pan

B. Murphy

G. Qian

R. E. Rosholt

The Schur Aggregation for Linear Systems and Determinants *

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

Abstract

According to our previous theoretical and experimental study, additive preconditioners can be readily computed for ill conditioned matrices, but it is not straightforward how such preconditioners can help us to facilitate the solution of linear systems of equations, computation of determinants, and other fundamental matrix computations. We develop some nontrivial techniques for this task. By applying the Sherman–Morrison–Woodbury formula and its new variations, we confine the original numerical problems to the computation of the Schur aggregates of smaller sizes. We overcome these problems by extending the classical algorithm for iterative refinement and applying advanced double-precision algorithms for high precision computation of sums and products.

2000 Math. Subject Classification: 65A12, 65F22, 65F05, 65F40

Key words: Additive preconditioning, Linear systems of equations, Determinants, SMW formula, Extended iterative refinement

*Supported by PSC CUNY Awards 68291–0037, 69330–0038, and 69350–0038. Some results of this paper have been presented at the International Conferences on the Matrix Methods and Operator Equations in Moscow, Russia, in June of 2005 and July 2007, on the Foundations of Computational Mathematics (FoCM'2005) in Santander, Spain, in July 2005, and on Industrial and Applied Mathematics, in Zürich, Switzerland, in July 2007, as well as at the SIAM Annual Meeting, in Boston, in July 2006, and at the International Workshop on Symbolic-Numeric Computation (SNC'07) in London, Ontario, Canada, in July 2007.

1 Introduction

Multiplicative preconditioning is a popular technique that facilitates the solution of ill conditioned linear systems of equations $A\mathbf{y} = \mathbf{b}$. The original basic idea is the transformation into equivalent but better conditioned linear systems $MAN\mathbf{x} = M\mathbf{b}$ for $\mathbf{y} = N\mathbf{x}$ and nonsingular matrices M and/or N, called preconditioners. (One of then can be the identity matrix I.) Such systems can be solved faster and/or more accurately (see [1], [3], [5], [18], and the extensive bibliography therein). More recently similar preprocessing was frequently directed towards compression of the spectrum of the singular values of an input matrix A into a smaller number of clusters, which achieves the same goal as preconditioning. The direction of preconditioning, however, remains highly important.

Multiplicative preconditioners are closely linked to the computation of the smallest singular values of an input matrix A and their associated singular vectors. Computation of these data is generally costly, and so preconditioning is usually applied to more special classes of inputs for which the data are readily available. Another problem is that the SVD-based preconditioners can easily destroy matrix structure.

Our point of departure is the alternative tchniques of *additive preconditioning* in [41]–[45], [47], [48], [51], [52], that is selecting an *additive preconditioner* P (hereafter *APC*) and mapping an ill conditioned input matrix A into its better conditioned *additive modification* $C = A + P$. Hereafter we write "*A*-" for "additive", "*APP*" for "additive preprocessor", and "*APC*" for "additive preconditioner". We observe the three following advantages of A-preconditioning.

- APCs are readily available for a large class of matrices
- We can extend the structure and sparseness of an input matrix to APCs
- A-preconditioning has a wider range of applications, which include eigensolving, the solution of singular and nonsingular linear systems of equations, and the computation of determinants.

According to the theoretical and extensive experimental study in [43]–[45], [47], [48], a well conditioned and properly scaled APP of a sufficiently large rank is likely to be an APC for a given ill conditioned matrix A as long as it is weakly random in the sense that it is generated by involving no data about the singular vectors of the matrix. In the present paper we show how such an APC can facilitate the solution of a linear system of equations $A\mathbf{y} = \mathbf{b}$ and produce the determinant of the matrix A as by-product (cf. Algorithms 7.1–7.4). At the end of Section 6 we comment on the classical problem of computing determinants and its present day importance.

Our algorithms involve some advanced techniques such as modification of the *SMW inversion formula* (by Sherman, Morrison, and Woodbury), extension of Wilkinson's *iterative refinement*, and algorithms for error-free multiplication and summation, for which we use the abbreviation *MSAs*. We refer the reader to [41], [51], [52] on A-preprocessing and A-preconditioning for computations in the null spaces and eigenspaces involving much simpler techniques.

We organize our presentation as follows. In the next section we demonstrate our approach by recursively applying rank-one modifications. In Section 3 we introduce basic definitions. In Section 4 we cover the SMW formula and its new variations. In Section 5 we link the singular values of the input matrix and of the auxiliary matrices involved in our computations. In Sections 6 and 7 we generalize our basic approach of Section 2 by allowing A-modifications of any rank. To support the resulting algorithms we extend the classical iterative refinement in Section 8. We comment on preserving matrix structure in our computations in Section 9 and on MSAs in Section 10. In Section 11 we describe our numerical tests, which confirm the efficiency of our algorithms. The tests have been performed jointly by all authors. Otherwise the paper is due to the first author.

2 Solving a linear system of equations with recursive rank-one modifications

Hereafter "ops" is our abbreviation for "arithmetic operations". M^H denotes the Hermitian transpose of a matrix M. (M^H is the transpose M^T if M is a real matrix.) We assume the customary notation for matrix computations in [2], [8], [24], [26], [57], [58], e.g., $\mathbf{v}$ is a vector, I_k denotes the $k \times k$ identity matrix, I is I_k for an unspecified k, $\sigma_j(A)$ is the j-th largest singular value of a matrix A of a rank ρ for $j = 1, \dots, \rho$, $||A|| = \sigma_1(A)$ is the 2-norm of a matrix A, and $\operatorname{cond} A = \sigma_1(A)/\sigma_\rho(A)$ is its condition number. A matrix A is ill conditioned if this number is large and is well conditioned otherwise. The concepts "large", "well conditioned" and "ill conditioned" can be quantified depending on the computational task and computer environment.

According to the cited study in [42]–[45], A-preconditioning with a random sparse and/or structured and well conditioned APP P of a rank r is likely to decrease the condition number of an $n \times n$ ill conditioned matrix A to the level of the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ provided the ratio $||P||/||A||$ is neither large nor small.

Now consider a nonsingular but ill conditioned linear system of n equations with n unknowns, $A\mathbf{y} = \mathbf{b}$, where the ratio $\sigma_1(A)/\sigma_{n-1}(A)$ is not large, whereas $\sigma_{n-1}(A) \gg \sigma_n(A)$. Suppose we have a rank-one APC $P = \mathbf{u}\mathbf{v}^H$ and a well conditioned A-modification $C = A + \mathbf{u}\mathbf{v}^H$. Apply the SMW inversion formula in our Theorem 4.1 in the case of $U = \mathbf{u}$ and $V = \mathbf{v}$ and obtain that

$$A^{-1} = C^{-1} + C^{-1}\mathbf{u}\mathbf{v}^H C^{-1}/g \text{ for } g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}.$$

This reduces the solution to well conditioned computations, apart from computing the value g, and we arrive at a new instance in the general class of *aggregation methods*. They successively a) aggregate an input I into an input I_1 of a smaller size, b) compute the solution for a given task but for the input

I_1, and c) disaggregate the solution Y_1 producing the solution Y for the original input I. In our case $I = A$, $I_1 = g$, $Y_1 = 1/g$, and $Y = A^{-1}$. The value $g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}$ is the Gauss transform of the 2×2 block matrix $\begin{pmatrix} C & \mathbf{u} \\ \mathbf{v}^H & 1 \end{pmatrix}$ and the Schur complement of its block C [24, pages 95 and 103]. We call this value a *Schur aggregate* and call the above methods the *(primal) Schur Aggregation*.

Aggregation methods for solving linear systems of equations are well known (see, e.g., the ones in [35], which have served as the springboard for the *Algebraic Multigrid*), but our novelty is the link to A-preconditioning.

The value g is absolutely small in virtue of our Theorem 5.3 (because $\operatorname{cond}_2 A$ is large, whereas $\operatorname{cond}_2 C$ is not) and thus must be computed within a small absolute error. To ensure this, we apply MSAs throughout and extend Wilkinson's iterative refinement when we compute the vectors $C^{-1}\mathbf{b}$ and $C^{-1}\mathbf{u}$ or $C^{-H}\mathbf{v}$ (see Section 8).

Next suppose that both ratios $\sigma_1(A)/\sigma_{n-1}(A)$ and $\sigma_{n-1}(A)/\sigma_n(A)$ are large. Then $\operatorname{cond} C$ is likely to be of the order of the former ratio, that is, is likely to satisfy $1 \ll \operatorname{cond} C \ll \operatorname{cond} A$.

We can apply our A-preconditioning and aggregation to the ill (although better) conditioned linear systems $C\mathbf{z} = \mathbf{u}$ and $C\mathbf{w} = \mathbf{b}$ and continue the process recursively until we arrive at a well conditioned matrix. This is expected to occur in r recursive steps provided the ratio $\sigma_1(A)/\sigma_{n-r+1}(A)$ is large but the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ is not large. We call such an integer r *numerical nullity* of the matrix A and write $r = \operatorname{nnul} A$, complementing the numerical rank $\operatorname{nrank} A = n - \operatorname{nnul} A$. Overall the r recursive steps require $(2/3)n^3 + 5rn(n+r) + O(rn)$ ops.

In this paper we modify the above techniques in various ways and in particular obtain their variations with better numerical properties (see Section 6). This study is naturally extended to other matrix computations in [41], [48], [51], and [52].

3 Basic definitions

Here are our basic definitions in addition to the ones in the previous sections.

A matrix A is *normalized* if $||A|| = 1$ and is unitary if $A^H A = I$.

A matrix A of a rank ρ has the Frobenius norm $||A||_F^2 = \operatorname{trace}(A^H A) = \sum_{j=1}^{\rho} \sigma_j^2(A)$ such that $||A|| \le ||A||_F \le \sqrt{\rho}||A||$.

Hereafter we use the abbreviation "*SVD*" for "Singular Value Decomposition". The *compact SVD* of an $m \times n$ matrix A of a rank ρ is the decomposition

$$A = S^{(\rho)}\Sigma^{(\rho)}T^{(\rho)H} = \sum_{j=1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H$$

where $S^{(\rho)} = (\mathbf{s}_j)_{j=1}^{\rho}$ and $T^{(\rho)} = (\mathbf{t}_j)_{j=1}^{\rho}$ are unitary matrices, $S^{(\rho)H}S^{(\rho)} = I_\rho$, $T^{(\rho)H}T^{(\rho)} = I_\rho$, $\Sigma^{(\rho)} = \operatorname{diag}(\sigma_j)_{j=1}^{\rho}$ is a diagonal matrix, $\mathbf{s}_j$ and $\mathbf{t}_j$ are m- and

n-dimensional vectors, respectively, and $\sigma_j = \sigma_j(A)$ for $j = 1, \ldots, \rho$ are the singular values of the matrix A, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho > 0$.

The *Moore-Penrose generalized inverse* of an $m \times n$ matrix A of a rank ρ (also called its *pseudo inverse*) is the matrix $A^- = \sum_{j=1}^{\rho} \sigma_j^{-1} \mathbf{t}_j \mathbf{s}_j^H$. We write A^- instead of the customary A^+ in [24], [57], [58], and we write A^{-H} for $(A^H)^- = (A^-)^H$.

We have $A^- = A^{-1}$ if $m = n = \rho$,

$$A^- = (A^H A)^{-1} A^H \quad \text{if} \quad m \geq n = \rho, \tag{3.1}$$

$$A^- = A^H (A A^H)^{-1} \quad \text{if} \quad m = \rho \leq n, \tag{3.2}$$

$\operatorname{cond} A = \sigma_1/\sigma_\rho = ||A|| \;\; ||A^-||$. It follows that

$$\operatorname{cond}(MN) \leq (\operatorname{cond} M) \operatorname{cond} N. \tag{3.3}$$

Hereafter we represent our APCs as the products $P = UV^H$ of two rectangular matrices U and V of full ranks, thus emphasizing the role of the ranks of the APCs.

4 The SMW formula and its variations

4.1 The case of nonsingular matrices

For a 2×2 block matrix

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

the matrix $G_{22} = B_{22} - B_{21} B_{11}^- B_{12}$ (respectively, $G_{11} = B_{11} - B_{12} B_{22}^- B_{21}$) is the *block Gauss transform* of the matrix B and the *Schur complement* of its northwestern block B_{11} (respectively, southeastern block B_{22}) provided $B_{11}^- B_{11} = I$ and/or $B_{11} B_{11}^- = I$ (respectively, $B_{22}^- B_{22} = I$ and/or $B_{22} B_{22}^- = I$) [24, pages 95, 103], [57, page 155]. We immediately verify the following lemma.

Lemma 4.1. *Let the above block matrix B be nonsingular and let*

$$B^{-1} = \begin{pmatrix} W & X \\ Y & Z \end{pmatrix}$$

for the same block decomposition of the matrix B and for some matrices W, X, Y and Z. Then $W = G_{11}^{-1}$ (resp. $Z = G_{22}^{-1}$) if the block B_{11} (resp. B_{22}) is nonsingular.

Theorem 4.1. *For $n \times r$ matrices U and V and an $n \times n$ matrices A, let the matrix $C = A + UV^H$ be nonsingular. Then the matrices A and $G = I_r - V^H C^{-1} U$ are the respective Schur complements (block Gauss transforms) of the blocks I_r and C in the matrix $W = \begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix}$ such that*

$$\det W = \det A = (\det C) \det G. \tag{4.1}$$

Furthermore [24, page 50], [57, Corollary 4.3.2], if the matrix A is nonsingular, then so is the matrix G, and we have the Sherman–Morrison–Woodbury formula $(C - UV^H)^{-1} = C^{-1} + C^{-1}UG^{-1}V^HC^{-1}$.

Proof. Begin with the factorizations

$$\begin{aligned}\begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix} &= \begin{pmatrix} I_n & U \\ 0 & I_r \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & I_r \end{pmatrix}\begin{pmatrix} I_n & 0 \\ V^H & I_r \end{pmatrix} \\ &= \begin{pmatrix} I_n & 0 \\ V^HC^{-1} & I_r \end{pmatrix}\begin{pmatrix} C & 0 \\ 0 & G \end{pmatrix}\begin{pmatrix} I_n & C^{-1}U \\ 0 & I_r \end{pmatrix},\end{aligned}$$

which implies equations (4.1). Invert this factorization to obtain that

$$\begin{aligned}\begin{pmatrix} A^{-1} & X \\ Y & Z \end{pmatrix} &= \begin{pmatrix} I_n & 0 \\ -V^H & I_r \end{pmatrix}\begin{pmatrix} A^{-1} & 0 \\ 0 & I_r \end{pmatrix}\begin{pmatrix} I_n & -U \\ 0 & I_r \end{pmatrix} \\ &= \begin{pmatrix} I_n & -C^{-1}U \\ 0 & I_r \end{pmatrix}\begin{pmatrix} C^{-1} & 0 \\ 0 & G^{-1} \end{pmatrix}\begin{pmatrix} I_n & 0 \\ -V^HC^{-1} & I_r \end{pmatrix} \\ &= \begin{pmatrix} C^{-1} + C^{-1}UG^{-1}V^HC^{-1} & X \\ Y & Z \end{pmatrix}\end{aligned}$$

for some matrices X, Y, and Z. □

Remark 4.1. *Equation (4.1) also follows from the two equations* $\det A = (\det C)\det(I_n - C^{-1}UV^H)$ *(implied by the equation $A = C(I_n - C^{-1}UV^H)$) and* $\det(I_r - X^HY) = \det(I_n - YX^H)$ *[27, Exercise 1.14] for $n \times r$ matrices $X = V^H$ and $Y = C^{-1}U$. For $r = 1$, $U = \mathbf{u}$, and $V = \mathbf{v}$, (4.1) turns into the equation* $\det A = (1 - \mathbf{v}^HC^{-1}\mathbf{u})\det C$ *(cf. [17] and [27]).*

4.2 The SMW formula for the full rank matrices

Suppose the matrices A, U, V, and $C = A + UV^H$ of sizes $m \times n$, $m \times r$, $n \times r$, and $m \times n$, respectively, have full ranks. For $m \leq n$ deduce that the matrix $I_m - UV^HC^-$ is nonsingular and

$$A = (I_m - UV^HC^-)C,\ A^- = C^-(I_m - UV^HC^-)^{-1}, \tag{4.2}$$

whereas for $m \geq n$ deduce that the matrix $I_n - C^-UV^H$ is nonsingular and

$$A = C(I_n - C^-UV^H),\ A^- = (I_n - C^-UV^H)^{-1}C^- \tag{4.3}$$

(cf. equations (3.1) and (3.2)).

For $m \geq n$ substitute $C \leftarrow I_m$ and $V^H \leftarrow V^HC^-$ into the SMW formula in Theorem 4.1 and obtain that

$$(I_m - UV^HC^-)^{-1} = I_m + U(I_r - V^HC^-U)^{-1}V^HC^-. \tag{4.4}$$

For $m \leq n$ substitute $C \leftarrow I_n$ and $U \leftarrow C^-U$ into the SMW formula and obtain that

$$(I_n - C^-UV^H)^{-1} = I_n + C^-U(I_r - V^HC^-U)^{-1}V^H. \tag{4.5}$$

By combining equations (4.2)–(4.5), extend the SMW formula to rectangular matrices of full rank as follows,

$$A^- = C^- + C^-U(I_r - V^HC^-U)^{-1}V^HC^-. \tag{4.6}$$

Observe that $A^-A = I_n$ for $m \geq n$, $AA^- = I_n$ for $m \leq n$, and $G = I_r - V^HC^-U$ is the block Gauss transform of the block matrix $\begin{pmatrix} C & V^H \\ U & I_r \end{pmatrix}$ and the Schur complement of its block C. We call the matrix G a Schur aggregate and the transition to computations with this matrix the Schur Aggregation. Here is a simple flowchart for computing a Schur aggregate.

Flowchart 4.1. *Given a matrix A of full rank, generate an APP UV^H and successively compute the matrices*

- $C = A + UV^H$*, which should have full rank,*
- C^-U *or* V^HC^-*,*
- $G = I_r - V^HC^-U$*.*

For smaller ranks r one can solve linear systems with the matrix G by applying the algorithms of the CG/GMRES type, even if the matrix is ill conditioned [2], [24, Sections 10.2–10.4], [55], [60], but the conditioning of this matrix can become the central issue for larger ranks r.

Finally suppose we seek a solution Y of a matrix equation $AY = B$ and use an APC UV^H such that $UF = B$ for a matrix F. Then SMW formula (4.6) implies that

$$Y = C^-UG^{-1}F \tag{4.7}$$

where $C = A + UV^H$ and $G = I - V^HC^-U$. Indeed, post-multiply equation (4.6) by $B = UF$, obtain that

$$Y = A^-B = A^-UF = C^-UF + C^-UG^{-1}V^HC^-UF = C^-U(I_r + G^{-1}V^HC^-U)F,$$

and substitute the equation $I_r + G^{-1}V^HC^-U = G^{-1}$.

In particular if $F = 1$ and $U = B = \mathbf{u} = \mathbf{b}$ is a vector, then g is a scalar, and (4.7) turns into the following vector equation,

$$Y = \mathbf{y} = C^-\mathbf{b}/g. \tag{4.8}$$

4.3 The dual SMW formula

Assume that the matrices A, U, V, and $C_- = A^- + VU^H$ have full rank and deduce that the matrix $I_n + VU^HA$ is nonsingular and

$$(C_-)^- = A(I_n + VU^HA)^{-1} \quad \text{where} \quad m \geq n, \tag{4.9}$$

whereas the matrix $I_m + AVU^H$ is nonsingular and

$$(C_-)^- = (I_m + AVU^H)^{-1}A \quad \text{where} \quad m \leq n. \tag{4.10}$$

Write $q = \operatorname{rank}(VU^H)$, apply the SMW formula, and obtain that

$$(I_n + VU^H A)^{-1} = I_n - V(I_q + U^H AV)^{-1}U^H A$$

for $m \geq n$ and

$$(I_m + AVU^H)^{-1} = I_m - AV(I_q + U^H AV)^{-1}U^H$$

for $m \leq n$. Substitute these equations into (4.9) and (4.10) and in both cases obtain the *dual SMW formula*

$$(C_-)^- = (A^- + VU^H)^- = A - AVH^{-1}U^H A, \quad H = I_q + U^H AV. \qquad (4.11)$$

Equations (4.11) express the matrix $(C_-)^-$ via the inverse H^{-1} of the matrix H, which is the Schur complement of the block $-A^-$ in the block matrix $\begin{pmatrix} -A^- & U^H \\ V & I_q \end{pmatrix}$.

Due to the equation $((C_-)^-)^- = A^- + VU^H$, we can express the solution $\mathbf{y}$ to the linear system $A\mathbf{y} = \mathbf{b}$ as follows,

$$\mathbf{y} = \mathbf{z} - VU^H\mathbf{b}, \ (C_-)^-\mathbf{z} = \mathbf{b}. \qquad (4.12)$$

For $q < \min\{m, n\}$ we call the matrix H the *dual Schur aggregate* and the transition to the computations with this matrix the *dual Schur Aggregation*.

5 The norm and conditioning of a Schur aggregate

In this section we link the singular values of the matrices A, C and G in the SMW formula (4.6). This implies further estimates for the norm and conditioning of the Schur aggregate G. In particular if $\operatorname{rank}(UV^H) = \operatorname{nnul} A > 0$, then *the matrix G has a small norm and both matrices C and G are well conditioned.*

We deduce from equation (4.7) for $F = I$ and $B = U$ that $A^-U = C^-UG^{-1}$ where A and C are $m \times n$ matrices of full rank and $m \geq n$. Then bound (3.3) and the equation $\operatorname{cond} M = \operatorname{cond}(M^-)$ together imply that $\operatorname{cond}(A^-U) \leq (\operatorname{cond} C)(\operatorname{cond} U)\operatorname{cond} G$. For random well conditioned $m \times r$ matrices U and larger r we can expect that the ratio $\operatorname{cond} A/\operatorname{cond}(A^-U)$ is not very large, and then, informally speaking, numerical problems of computing with matrix A are translated to the computations with the matrices C and G. Next we deduce a similar property in the case of APCs UV^H of any rank provided the matrix C is well conditioned.

First we estimate the jth singular values of the matrix G^{-1}, $j = 1, \ldots, r$, in terms of the singular values $\sigma_j(A^-)$, $\sigma_1(C)$, and $\sigma_1(C^-)$. Theorem 5.2 is a special case of [58, Theorem 3.3.3] where $E = I_n$.

Theorem 5.1. *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 $\sigma_j(M)\sigma_-(W) \leq \sigma_j(MW) \leq \sigma_j(M)\sigma_+(W)$ and $\sigma_j(N)\sigma_-(W) \leq \sigma_j(WN) \leq \sigma_j(N)\sigma_+(W)$ for $j = 1, \ldots, \rho$ and $\rho \times \rho$ matrices M and N.*

Proof. The singular values are invariant in multiplication by a unitary matrix, and so we can consider just the case of a positive diagonal matrix W. In this case the claimed bounds follow from the Courant–Fischer Minimax Characterization [24, Theorem 8.1.2], [58, Theorem 3.3.2]. $\square$

Theorem 5.2. *We have $\sigma_j(W) - 1 \leq \sigma_j(W + I_n) \leq \sigma_j(W) + 1$ for an $n \times n$ matrix W and for $j = 1, 2, \ldots, n$.*

Theorem 5.3. *For positive integers m, n, and r, a normalized $m \times n$ matrix A, and a pair of matrices U of size $m \times r$ and V of size $n \times r$, write $C = A + UV^H$ and $G = I_r - V^H C^- U$. Suppose the matrices A and $C = A + UV^H$ have full rank $\rho \geq r$. Then the matrix G is nonsingular, and we have*

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

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

Proof. Let $m \geq n$. Deduce from equation (4.3) that the matrix $G_n = I_n - C^- UV^H$ is nonsingular. So is the matrix G as well because $\det G = \det G_n$ [27, Exercise 1.14].

Next combine equation (4.3) with Theorem 5.1 for $M = G_n^{-1}$, $W = C^-$, and $A^- = MW$, to obtain that

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

for $j = 1, \ldots, \rho$. Substitute $\sigma_-(C^-) = 1/\sigma_+(C)$ and $\sigma_+(C^-) = 1/\sigma_-(C)$ and obtain that

$$\sigma_j(A^-)\sigma_-(C) \leq \sigma_j(G_n^{-1}) \leq \sigma_j(A^-)\sigma_+(C) \text{ for } j = 1, \ldots, \rho. \tag{5.1}$$

Combine Theorem 5.1 for $W = C^- U$ and $N = G^{-1}$ with the equations and inequalities $\sigma_j(C^- UG^{-1}V^H) = \sigma_j(C^- UG^{-1})$ for $j = 1, \ldots, r$, $\sigma_-(C^- U) \geq \sigma_-(C^-) = 1/\sigma_+(C)$, and $\sigma_+(C^- U) \leq \sigma_-(C^+) = 1/\sigma_-(C)$ to deduce that

$$\sigma_j(G^{-1})/\sigma_+(C) \leq \sigma_j(C^- UG^{-1}V^H) \leq \sigma_j(G^{-1})/\sigma_-(C)$$

for $j = 1, \ldots, r$. Combine the latter bounds with Theorem 5.2 for $W = C^- UG^{-1}V^H$ and equation (4.5) to deduce that

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

and therefore

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

for $j = 1, \ldots, r$. Combine this equation with equation (5.1) and obtain the claimed bounds in the case of $m \geq n$.

For $m \leq n$ proceed similarly but use equations (4.2) and (4.4) instead of (4.3) and (4.5), replace G_n with $G_m = I_m - UV^H C^-$ and furthermore, invoking Theorem 5.1 the first and the second time, replace $M = G_n^{-1}$ with $N = G_m^{-1}$ and replace $W = C^- U$ with $W = V^H C^-$, respectively. $\square$

Corollary 5.1. *Under the assumption of Theorem 5.3 we have*

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

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

Suppose A is an $n \times n$ nonsingular matrix such that $\operatorname{nnul} A = r$ and UV^H is a random, well conditioned and properly scaled APP of a rank r. Then the values $\sigma_{n-j+1}(A)/\sigma_1(A)$ are small for $j \leq r$ and are not small for $j > r$, whereas the value $\sigma_n(C)$ is likely to be of the order of $\sigma_{n-r}(A) \gg \sigma_{n-r+1}(A)$. Therefore, all singular values $\sigma_{r-j+1}(G) = 1/\sigma_j(G^{-1})$ for $j = 1, \ldots, r$ are likely to be of the order of at most $\sigma_{n-j+1}(A)$. Furthermore (cf. Corollary 5.1), $\operatorname{cond} G$ is likely to be of the order of $(\operatorname{cond} C)^2\sigma_{n-r+1}(A)/\sigma_n(A)$, whereas the 2-norm $||G|| = \sigma_1(G)$ is likely to be of the order of $\sigma_{n-r+1}(A)$. The latter value has the order of $\sigma_1(A)/\operatorname{cond} A$. Thus, as we claimed, the matrix G is expected to have a small norm and to be well conditioned if $||A|| \neq 1$ and if $\operatorname{nnul} A = r$.

Finally all our estimates for matrices A, C, and G are readily extended to the dual counterparts A^-, $(C_-)^-$ and H of these matrices.

6 The solution of linear systems of equations and the computation of determinants with the Schur Aggregation. High level description

Based on the study in the previous sections we can extend our algorithms in Section 2 by using

1. the dual SMW formula
2. APCs of ranks $r > 1$
3. APCs UV^H where $U\mathbf{f} = \mathbf{b}$ for some vector $\mathbf{f}$ (cf. (4.8)).

Let us comment on these three extensions and then on computing determinants.

1. We specify the dual process in Algorithms 7.2–7.4 in the next section. Here are our introductory comments. Recursive application of dual rank-one modifications naturally mimics the recursive process in Section 2 but has an advantage of avoiding most of divisions and restricting matrix inversions to inverting a single dual A-modification at the last recusive step, where this A-modification is well conditioned. We first apply formulae (4.11) and (4.12) for $U = \mathbf{u}$ and $V = \mathbf{v}$ and obtain that

$$h(C_-)^- = hA - A\mathbf{v}\mathbf{u}^H A = A(hI - \mathbf{v}\mathbf{u}^H A), \quad h = 1 - \mathbf{u}^H A\mathbf{v}, \tag{6.1}$$

$\mathbf{y} = \mathbf{z} - \mathbf{v}\mathbf{u}^H\mathbf{b}$, and $(C_-)^-\mathbf{z} = \mathbf{b}$. These equations define division-free reduction of a linear system $\{A\mathbf{y} = \mathbf{b}\} \rightarrow \{h(C_-)^-\mathbf{z} = h\mathbf{b}\}$.

For a pair of random vectors $\mathbf{u}$ and $\mathbf{v}$ (as well as for a random vector $\mathbf{u}$ and for $\mathbf{v} = \mathbf{u}$) scaled so that the ratio $||\mathbf{v}\mathbf{u}^H||/||A^-||$ is neither large nor small, we can expect (cf. [45]) that $\mathrm{cond}(C_-) = \mathrm{cond}((C_-)^-)$ has the order of $\sigma_2(A)/\sigma_{n-1}(A)$. If this ratio is large, we apply dual A-preconditioning and dual aggregation to the matrix $h_1(C_{-1})^- = h(C_-)^-$. This produces a matrix $h_2(C_{-2})^-$ with the condition number expected to be at the level of $\sigma_3(A)/\sigma_n(A)$. Recursively we expect to arrive at a well conditioned matrix $h_q(C_{-q})^-$ in q steps provided $\mathrm{nnul}(A^-) = q$ (see Algorithm 7.1 in the next section).

Overall in this transition we need to multiply $2q$ matrices of the size $n \times n$ by $2q$ vectors and in addition to perform either qn^2 ops including divisions (cf. equation (4.11)) or $2qn^2$ ops division-free. We can apply Gaussian elimination to solve the linear system $h(C_{-r})^-\mathbf{z}_r = h\mathbf{b}$ in $(2/3)n^3 + O(n^2)$ ops. The subsequent transition to the solution vector $\mathbf{y}$ requires only $O(qn)$ ops (cf. equation (4.12)).

Compared to the recursive process in Section 2 for $r = q$, the book-keeping for the back transition to the solution $\mathbf{y}$ is simplified, and we can save the order of q^2n ops at this stage, but we use extra qn^2 ops in a division-free version. Most important difference, however, is that at the stage of computing the Schur aggregates G we avoid numerical problem and do not need iterative refinement because the respective computation of the dual Schur aggregates H is division-free.

To support the dual recursive process we need a crude estimate for the value $\sigma_n(A)$, versus a crude estimate for $\sigma_1(A)$ in the recursive process in Section 2. The known numerically stable algorithms produce both estimates by using $O(n^2)$ ops [24, Sections 2.3.2, 2.3.3, and 3.5.4], [57, Section 5.3].

We can combine q recursive steps of the above dual process with r recursive steps of the primal process in Section 2. This is likely to decrease the condition number of the input matrix A to the level of $\sigma_{q+1}(A)/\sigma_{n-r}(A)$.

2. Suppose A is a nonsingular ill conditioned $n \times n$ matrix such that the ratios σ_1/σ_{n-r+1} and σ_{n-r}/σ_n (resp. σ_1/σ_q and σ_{q+1}/σ_n) are not large, but $\sigma_{n-r} \gg \sigma_{n-r+1}$ (resp. $\sigma_q \gg \sigma_{q+1}$), so that the matrix A is ill conditioned due to a single jump in the spectrum of its singular values. We can find the threshold value r (resp. q) of the rank of the APC by recursively testing the values $0, 1, 2, 4, 8, \ldots$ until we arrive at a well conditioned matrix C (resp. an ill conditioned matrix H) and then applying binary search to decrease this value.

Under this assumption a single primal APC UV^H of rank r (resp. a dual APC of rank q) can replace r primal (resp. q dual) rank-one recursive steps. Overall, at both A-preconditioning and aggregation stages, we perform about as many ops as in the case of recursive rank-one modifications, except that we need to perform about $2r^3$ (resp. $2q^3$) extra ops to invert the matrix G (resp. H). Moreover, in the dual case, some additional care is needed to avoid divisions. The extra effort, however, can be more than compensated by the well known benefits of applying block matrix multiplications [24], [57] and avoiding divisions.

We can extend the approach recursively by selecting proper ranks r_i for APCs $U_iV_i^H$ (resp. q_i for APCs $V_iU_i^H$) at the ith recursive step for $i = 1, 2, \ldots$. E.g., in Algorithm 7.4 we choose $q_i = 1$ for all i.

3. Equations (4.7) and (4.8) enable us to simplify the computation of the solution vector $\mathbf{y} = A^{-}\mathbf{b}$ to the linear system $A\mathbf{y} = \mathbf{b}$ versus the primal SMW formula. To incorporate these equations into the recursive process of A-preconditioning and aggregation, we choose the matrices $U = U_k$ and $F = F_k$ at the kth recursive step as follows, $U_1 = \mathbf{b}$, $F_1 = 1$, $U_k = (U_{k-1}, \mathbf{u_k})$, $F^T = (1, \mathbf{0}^T)$, $k = 2, 3, \ldots$. Then the kth recursive step involves a new (random) vector $\mathbf{u}_k$ such that $U_k F_k = \mathbf{b}$. The progress with improving the conditioning is the same as before except that the impact of the first step with $U_1 = \mathbf{b}$ decreases (resp. becomes nil) wherever the vector $\mathbf{b}$ lies near (resp. in) the range of the matrix A.

4. We refer the reader to [4], [13], [14], [48], [53], [56], and the extensive bibliography therein on the classical problem of computing determinants, currently highly important to the fundamental geometric and algebraic-geometric computations. In particular the computation of convex hulls and Voronoi diagrams essentially amounts to recursive computation of the signs of determinants. Performed numerically with unit roundoff (computer precision) ϵ, the sign computation can be certified if $\operatorname{cond} A < c/\epsilon$ for a constant c (cf. [53]). Generally numerical computation of both sign and value of the determinant is the hardest where the input matrix is ill conditioned, which makes preconditioning highly important in this area.

Our preconditioning recipes for solving linear systems of equations are immediately extended to the determinant computation due to the following simple modifications of the SMW formulas (4.6) and (4.11),

$$\det A = (\det G) \det C = (\det((C_{-})^{-1})) / \det H. \tag{6.2}$$

7 Recursive primal and dual A-preconditioning and Schur aggregation

Next we specify the recursive processes of primal and dual A-preconditioning and Schur aggregation.

We use the matrices U and V whose entries we can round to a fixed (smaller) number of bits to control or avoid rounding errors in computing the matrices $C = A + UV^H$ and $H = I_q + U^H AV$ (cf. [45]).

In each recursive step we choose ranks r and q for the primal and dual APPs according to some fixed policies RANK and DUAL RANK, respectively. E.g., in every step we can set $r = 1$ (resp. $q = 1$) or let r (resp. q) be the minimum rank of a primal APP UV^H (resp. dual APP VU^H) for which the matrix $C = A + UV^H$ (resp. $C_{-} = A^{-1} + VU^H$) is well conditioned. Another sample option is to let q be the rank of a dual APP for which we compute error-free the scalar $h = \det H$ and the adjoint matrix hH^{-1} (cf. Algorithms 7.3 and 7.4 and Remark 7.2).

Algorithm 7.1. Recursive primal A-preconditioning and Schur aggregation for determinant and inverse.

INPUT: *a nonsingular $n \times n$ matrix A and a policy RANK.*

OUTPUT: $\det A$ *and the matrix* A^{-1}.

COMPUTATIONS:

0. Choose a positive integer r according to the policy RANK.

1. Generate the pair of normalized $n \times r$ matrices $\tilde{U}$ and V, such that $||\tilde{U}|| = ||V|| = 1$.

2. Compute a crude estimate ν for the norm $||A||$.

3. Compute the matrix $U = \nu\tilde{U}$.

4. Compute the $n \times n$ matrix $C = A + UV^H$, its inverse C^{-1} and determinant $\det C$. (If this matrix is ill conditioned, set $A \leftarrow C$ and reapply the algorithm.)

5. Compute the $n \times n$ matrix $G = I_r - V^H C^{-1} U$, its inverse and determinant. (The computation of the matrix G may require high precision due to the cancellation of the leading bits in the representation of the entries. If this matrix is ill conditioned, set $A \leftarrow G$ and reapply the algorithm.)

6. Compute and output the $n \times n$ matrix $A^{-1} = C^{-1} + C^{-1}UG^{-1}V^H C^{-1}$ and the scalar $\det A = (\det C)\det G$ and stop.

Algorithm 7.2. Recursive dual A-preconditioning and Schur aggregation (determinant and inverse).

INPUT: *a nonsingular $n \times n$ matrix A and a policy DUAL RANK (cf. Remark 7.2).*

OUTPUT: $\det A$ *and the matrix* A^{-1}.

COMPUTATIONS:

0. Choose a positive integer q according to the policy DUAL RANK.

1. Generate the pair of normalized $n \times q$ matrices $\tilde{U}$ and V, such that $||\tilde{U}|| = ||V|| = 1$.

2. Compute a crude estimate ν for the norm $||A^{-1}|| = \operatorname{cond} A/||A||$.

3. Compute the matrix $U = \nu\tilde{U}$.

4. Compute the $q \times q$ matrix $H = I_q + U^H AV$ and its inverse and determinant. If this matrix is ill conditioned, set $A \leftarrow H$ and reapply the algorithm. (The computation of the matrix H may require high precision due to the cancellation of the leading bits in the representation of the entries. If this matrix is ill conditioned, set $A \leftarrow H$ and reapply the algorithm.)

5. Compute the matrix $(C_-)^{-1} = A - AVH^{-1}U^H A$ and its inverse and determinant. If the matrix C_- is ill conditioned, set $A \leftarrow (C_-)^{-1}$ and reapply the algorithm.

6. Compute and output the $n \times n$ matrix $A^{-1} = ((C_-)^{-1})^{-1} - V^H U$ and the scalar $\det A = (\det((C_-)^{-1}))/\det H$ and stop.

One can adjust and simplify these algorithms in the case where we only wish to solve a linear system of equations $A\mathbf{y} = \mathbf{b}$ rather than to invert a matrix A. In particular one can choose primal APCs UV^H such that $U\mathbf{f} = \mathbf{b}$ for some vector $\mathbf{f}$ and use the simplified expression (4.7).

Below is our detailed description of the dual recursive process for solving a linear system of equations and computing determinant, where we also change Stage 4 and compute the inverse H^{-1} as the pair of $\det H$ and $\operatorname{adj} H = (\det H)H^{-1}$, which are integral if so is the matrix H.

Algorithm 7.3. Recursive dual A-preconditioning and Schur aggregation (linear system and determinant).

INPUT: *a nonsingular $n \times n$ matrix A, a policy DUAL RANK (cf. Remark 7.2), a vector $\mathbf{b}$ of dimension n, and a reasonably large tolerance $t > 1$.*

OUTPUT: $\det A$ *and a vector* $\mathbf{y}$ *satisfying the linear system* $A\mathbf{y} = \mathbf{b}$.

INITIALIZATION: *$i \leftarrow 1$, $A_0 \leftarrow A$, $h_0 \leftarrow 1$, and compute a crude estimate ν_0 for the norm $||A_0^{-1}||$ (cf. Remark 7.1).*

COMPUTATIONS:

STAGE A.

0. Choose a positive integer q_i according to the policy DUAL RANK.

1. Generate the pair of normalized $n \times q_i$ matrices $\tilde{U}_i$ and V_i, such that $||\tilde{U}_i|| = ||V_i|| = 1$.

2. Compute the matrix $U_i = \nu_{i-1}\tilde{U}_i$.

3. Compute the $q_i \times q_i$ matrix

$$H_i = I_q + U_i^H A_{i-1} V_i, \tag{7.1}$$

the scalar $h_i = \det H_i$, and the $q_i \times q_i$ matrix $\tilde{H}_i = \operatorname{adj}\ H_i = h_i H_i^{-1}$.

4. Compute the matrix

$$A_i = h_i A_{i-1} - A_{i-1} V_i \tilde{H}_i U_i^H A_{i-1}. \tag{7.2}$$

5. Compute crude estimates ν_i for the norm $||A_i^{-1}||$ and κ_i for the condition number $\operatorname{cond} A_i$ *(cf. [24, Sections 2.3.2, 2.3.3, and 3.5.4], [26, Chapter 14], [57, Section 5.3]). If $\kappa_i > t$, then increment the integer parameter $i \leftarrow i + 1$ and go back to substage 1. Otherwise write $r \leftarrow i$ and go to Stage B.*

STAGE B. *Compute the scalar* $\det A_r$ *and the vector* $\mathbf{y}_r = A_r^{-1}\mathbf{b}_r$.

STAGE C. *Recursively for* $i = r, r-1, \dots, 1$ *compute the scalars* $\det A_{i-1} = (1/h_i)\det(A_i/h_i) = (1/h_i)^{n+1}\det A_i$ *and the vectors* $\mathbf{y}_{i-1} = h_i\mathbf{y}_i - V_i U_i^H \mathbf{b}_i$.

STAGE D. *Output the scalar* $\det A = \det A_0$ *and the vector* $\mathbf{y} = \mathbf{y}_0$ *and stop.*

For an ill conditioned matrix A the algorithm reduces the computation of the determinant $\det A$ and the solution of a linear system $A\mathbf{y} = \mathbf{b}$ to the same two computational problems for the matrix A_r, which is supposed to be better conditioned due to A-preprocessing in equations (7.1) and (7.2). Apart from the solution of these problems for the matrix A_r at Stage B, the inversion of the matrices H_i and computing their determinants at Stage A3, and the norm and condition estimation at the Initialization Stage and Stage A5, the algorithm is division-free.

Correctness of the algorithm follows from equations (4.11), (4.12), and (6.2).

Below is a specification of the algorithm under the policy $q_i = 1$ for all i where the matrices U_i and V_i turn into vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ and we still round their coordinates to a fixed (smaller) number of bits.

Algorithm 7.4. Recursive dual rank-one A-preconditioning and Schur aggregation.

INPUT: *a nonsingular $n \times n$ matrix A and a vector $\mathbf{b}$ of dimension n.*

OUTPUT: $\det A$ *and a vector $\mathbf{y}$ satisfying the linear system $A\mathbf{y} = \mathbf{b}$.*

INITIALIZATION: *$i \leftarrow 1$, $A_0 \leftarrow A$, $h_0 \leftarrow 1$, and compute a crude estimate ν_0 for the norm $||A_0^{-1}||$ (cf. Remark 7.1).*

COMPUTATIONS:

STAGE A.

1. Generate the pair of n-th dimensional normalized vectors $\tilde{\mathbf{u}}_i$ and $\mathbf{v}_i$, $||\tilde{\mathbf{u}}_i|| = ||\mathbf{v}_i|| = 1$.

2. Compute the vector $\mathbf{u}_i = \nu_{i-1}\tilde{\mathbf{u}}_i$.

3. Compute the scalar $h_i = 1 + \mathbf{u}_i^H A_{i-1}\mathbf{v}_i$.

4. Compute the matrix $A_i = h_i A_{i-1} - A_{i-1}\mathbf{v}_i\mathbf{u}_i^H A_{i-1}$.

5. Compute crude estimates ν_i for the norm $||A_i^{-1}||$ and κ_i for the condition number $\operatorname{cond} A_i$*. If $\kappa_i > t$, then increment the integer parameter $i \leftarrow i + 1$ and go back to substage 1. Otherwise write $r \leftarrow i$ and go to Stage B.*

STAGE B. *Compute the scalar* $\det A_r$ *and the vector $\mathbf{y}_r = A_r^{-1}\mathbf{b}_r$.*

STAGE C. *Recursively for $i = r, r-1, \dots, 1$ compute the scalars* $\det A_{i-1} = (1/h_i)\det(A_i/h_i) = (1/h_i)^{n+1}\det A_i$ *and the vectors* $\mathbf{y}_{i-1} = h_i\mathbf{y}_i - V_i U_i^H\mathbf{b}_i$.

STAGE D. *Output the scalar* $\det A = \det A_0$ *and the vector $\mathbf{y} = \mathbf{y}_0$ and stop.*

Apart from the computation of the determinant $\det A_r$ and the solution of a linear system $A_r\mathbf{y}_r = \mathbf{b}_r$ at Stage B and from the norm and condition estimation at the Initialization Stage and Stage A5, the algorithm is division-free.

Remark 7.1. *Recursive computations of the norms ν_i and condition numbers κ_i can be simplified based on orthogonal (QR or UTV) factorizations of the matrices A_i, which can be recomputed in $O(n^2)$ ops when the matrix A_i is modified by adding a rank-one matrix [24, Section 12.5], [57, Section 5.4].*

Remark 7.2. *By minimizing the rank q_i of the APPs in each step i of Algorithm 7.2 (resp. Algorithm 7.3) we avoid the extra work for computing determinants h_i and inverses H_i^{-1} (resp. adjoints $h_iH_i^{-1}$) at Stage A.4 (resp. A.3) of the algorithm but increase the number r of recursive steps. Each step requires estimating the norm and condition number and (in case of matrices A having a small displacement rank) increases the displacement rank of the A-modification (cf. [40]). This suggests the following policy DUAL RANK at the i-th recursive step: increment the value q_i recursively until the resulting incremental cost of computation at Stage A.4 (resp. A.3) of the algorithm exceeds the cost that we can save due to the decrease of the number of recursive steps.*

8 Extended iterative refinement

Consider the computation of the Schur aggregate $G = I_r - V^TC^{-1}U$ where the input matrix A is ill conditioned, whereas its A-modification C is not. We rely on Flowchart 4.1 where we compute the matrix $W = C^{-1}U$ from the matrix equation $CW = U$.

Under our assumptions on the matrices A and C, Theorem 5.3 implies that the norm $||G||$ is small, and so the computation of every diagonal entry of the Schur aggregate G annihilates a number of its leading significant bits. Therefore we must compute these entries with a high precision, and so we apply MSAs in this computation and extend Wilkinson's iterative refinement when we compute the matrix $C^{-1}U$.

In its classical form the refinement stops where the matrix $W = C^{-1}U$ is computed 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 [9] and [34] to improve the approximation further. As in the latter symbolic algorithm, we represent the output values as the sums of fixed-precision numbers (cf. Section 10).

Extended iterative refinement (Outline)

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 $i = 0, 1, \dots, k$. (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 $i = 0, 1, \dots, k$, so that the norm $||W_i - W||$ recursively decreases until it reaches the limit posed by rounding errors.) Here is a simple example for demonstration of our extension in the case where $n = 4$, $r = 1$, and the $r \times r$ Schur aggregate G turns into a scalar.

Example 8.1. $A = \begin{pmatrix} 63419461 & -29226193 & -41333003 & -8964 \\ -17439352 & -22167219 & -14775811 & -3204 \\ -38199953 & -59526299 & -19725060 & -4276 \\ -7074 & 3261 & 4611 & 1 \end{pmatrix}$

$U^T = \begin{pmatrix} 75776 & 258048 & 122880 & 118784 \end{pmatrix}$

$V^T = \begin{pmatrix} 128 & 148 & 72 & 148 \end{pmatrix}$

$C = \begin{pmatrix} 73118789 & -18011345 & -35877131 & 11205884 \\ 15590792 & 16023885 & 3803645 & 38187900 \\ -22471313 & -41340059 & -10877700 & 18181964 \\ 15197278 & 17583293 & 8557059 & 17580033 \end{pmatrix}$

$G_0 = 1$

$W_0^T = \begin{pmatrix} 0.00000000000008 \\ 0.00000000027075 \\ 0.00000146570198 \\ 0.00675746938214 \end{pmatrix}$

$G_1 = 2.190473991081632e-008$

$W_1 = \begin{pmatrix} -0.00000000124834 \\ 0.00000001025780 \\ -0.00000886188400 \\ 0.14800929926118 \end{pmatrix} * 1.0e-009$

$G_2 = 3.174438743663640e-016$

$W_2 = \begin{pmatrix} 0.00000000716215 \\ 0.00000003837446 \\ -0.00002747063331 \\ 0.21450242917524 \end{pmatrix} * 1.0e-017$

$G_3 = -7.918752906512810e-024$

$W_3 = \begin{pmatrix} 0.00000002240260 \\ 0.00000002020945 \\ -0.00010131901406 \\ -0.53500152161636 \end{pmatrix} * 1.0e-025$

$G_4 = -1.475542403337810e-030$

$W_4 = \begin{pmatrix} 0.00000005462336 \\ -0.00000003648089 \\ 0.00018978679691 \\ -0.90511944620263 \end{pmatrix} * 1.0e-033$

$G_5 = -1.341598391541817e-030$

$W_5 = \begin{pmatrix} -0.00000002134640 \\ 0.00000000681126 \\ -0.00005379511619 \\ -0.12995555745450 \end{pmatrix} * 1.0e-040$

$$G_6 = -1.341598389618088e - 030$$

$$W_6 = \begin{pmatrix} -0.00000007190044 \\ 0.00000001308189 \\ -0.00004562685927 \\ -0.44933111982233 \end{pmatrix} * 1.0e - 048$$

$$G = G_7 = -1.341598389618088e - 030$$

Theorem 5.3 defines a small upper bound on the norm $||G||$ if A is an ill conditioned matrix and if the matrix C is well conditioned. Therefore, we can have $G_i \approx 0$ for $i = 0, 1, \dots, k$ and some positive integer k. At the ith step of iterative refinement for $i \le k$ we need to store only the most recently computed matrix G_{i+1} overwriting G_i, and similarly 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 save 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 $i = 0, 1, \dots, k$ we seek error-free output because even small relative errors can completely corrupt the matrix G. To meet the challenge, we have two tools, namely, a) MSAs and b) the truncation of the entries of the matrices U, V, C, and W_i for all i.

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, we require that the matrices C^{-1} and $W_i \leftarrow C^{-1}U_i$ be computed within an error norm bound that ensures the decrease of the residual norms $u_i = ||U_i||$ (and consequently the error norm $e_i = ||E_i||$ since $E_i = C^{-1}U_i$) by a fixed factor ϕ exceeding one in each iteration (cf. Corollary 8.2). For numerical inversion of the matrix C under the desired norm bound, we can apply any direct or iterative algorithm (e.g., Gaussian elimination, possibly combined with the classical numerical iterative refinement, or Newton's iteration in [40, Chapter 6], [46], [50]).

Within the allowed perturbation norm, 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 estimate from above the norm of the input perturbation and the precision of computing that ensure the output error norm within the fixed tolerance bound. Then we perturb the input within the estimated norm bound to represent it with fewer bits. In particular we set to zero the absolutely smaller input entries and round every other entry to fewer bits. Finally we perform iterative refinement and verify that it converges as expected. Otherwise correct our policy of input perturbation.

Estimates for the errors and the parameter $\theta = 1/\phi$

Theorem 8.1. *Consider the subiteration*

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

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 8.2. *Assume that*

$$W_i = (C - \tilde{E}_i)^{-1}U_i = C^{-1}U_i - E_i \quad \text{for all} \quad i.$$

Write $e_i = ||E_i||$, $u_i = ||U_i||$, and $\theta_i = \delta_i ||C||$ where

$$\delta_i = \delta(C, \tilde{E}_i) = 2||\tilde{E}_i||_F \max\{||C^{-1}||^2, ||(C - \tilde{E}_i)^{-1}||^2\}.$$

Then we have $e_i \le \delta_i u_i$ for all i, $e_{i+1} \le \theta_i e_i$, $u_{i+1} \le \theta_i u_i$ for $i = 0, 1, \ldots, k-1$.

Proof. We follow [41, Section 8] and begin with some auxiliary results.

Theorem 8.3. *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 8.1. *Let C and $C + E$ be two nonsingular matrices. Then*

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

Proof. See [24, Section 5.5.5]. □

Corollary 8.1. *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 8.3 and Corollary 8.1 and obtain that $u_{i+1} \le \theta_i u_i$ and $e_{i+1} \le \theta_i e_i$ for $\theta_i = \delta_i ||C||$ and for all i. Summarize our estimates and obtain Theorem 8.2. □ □

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. The theorem also shows local quadratic convergence, that is doubling the number of correct bits in every step. We accomodate such a progress, however, only until this number reaches the working precision of computing, and then we shift back to performing linearly convergent computations with this precision (see Corollary 8.3).

Let us next estimate the values θ_i. 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. [24, Section 3.3], [57, Theorem 3.4.9], [26]). In this case the values

$$\begin{aligned}\theta_i &= \delta_i||C|| \\ &= 2r_i \max\{\mathrm{cond}^2\, C, \mathrm{cond}^2(C - E_i)\}||C||_F/||C|| \\ &\approx 2(\mathrm{cond}\, C)^2 r_i||C||_F/||C|| \\ &\leq 2(\mathrm{cond}\, C)^2 r_i n\end{aligned}$$

tend to be significantly less than one.

Precision bounds

Finally we estimate the precision required 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_k is zero or one, we write $t(b) = t$, $s(b) = s = \lfloor \log_2 |b| \rfloor$, and $p(b) = s - t + 1$, so that $p(b)$ is the precision in the binary representation of b. For an $n \times n$ matrix $M = (m_{i,j})_{i,j}$ we write $s(M) = \max_{i,j} s(m_{i,j})$, $t(M) = \min_{i,j} t(m_{i,j})$, $p(M) = s(M) - t(M) + 1$. Then

$$\log_2(n||M||) \leq s(M) \leq \lfloor \log_2 ||M|| \rfloor, \tag{8.1}$$

and the absolute value of each entry of the matrix M is the sum of some powers 2^k for integers k selected in the range $[t(M), s(M)]$.

Lemma 8.2. *We have $t(U_{i+1}) \geq \min\{t(U_i), t(CW_i)\}$ for all i. Moreover $t(CW_i) \geq t(W_i)$ if the (scaled) matrix C is filled with integers.*

Proof. The lemma follows from the equations $U_{i+1} = U_i - CW_i$. □

Lemma 8.3. *We have $s(U_{i+1}) \leq s(U_i) + \log_2(\theta_i n)$ for all i.*

Proof. The lemma follows from the bounds $u_{i+1} \leq \theta_i u_i$ and (8.1). □

Lemma 8.4. *We have $s(U_{i+1}) \leq s(CW_i) + \log_2 f_i$ and $s(U_{i+1}) \leq s(W_i) + \log_2(f_i||C||)$ for $\theta_i < 1$, $f_i = \frac{\theta_i n}{|1-\theta_i|}$, and all i.*

Proof. First recall that $u_{i+1} \leq \theta_i u_i$, so that $|u_i - u_{i+1}| \geq |1/\theta_i - 1|u_{i+1}$. The equation $U_i - U_{i+1} = CW_i$ implies that $||CW_i|| = ||U_i - U_{i+1}|| \geq |u_i - u_{i+1}| \geq |1/\theta_i - 1|u_{i+1}$. Therefore $u_{i+1} \leq (f_i/n)||CW_i|| \leq (f_i||C||/n)||W_i||$. Combine these bounds with bound (8.1) for $M = U_{i+1}$, $M = CW_i$ and $M = W_i$. □

Corollary 8.2.

$$\begin{array}{lll} a)\ \mathit{If} & t(U_{i+1}) \geq t(U_i), \\ \quad \mathit{then} & p(U_{i+1}) \leq p(U_i) + \log_2(\theta_i n). \\ b)\ \mathit{If} & t(U_{i+1}) \geq t(CW_i), \\ \quad \mathit{then} & p(U_{i+1}) \leq p(CW_i) + \log_2 f_i. \\ c)\ \mathit{If} & t(U_{i+1}) \geq t(W_i), \\ \quad \mathit{then} & p(U_{i+1}) \leq p(W_i) + \log_2(f_i\|C\|). \end{array}$$

Recall that in virtue of Lemma 8.2, at least one of assumptions a) and b) is always satisfied, and if the matrix C is filled with integers, then so is one of assumptions a) and c) as well.

Corollary 8.3. *Suppose we have precision bounds $p(W_i) \leq \widehat{p}$ and/or $p(CW_i) \leq \tilde{p}$ for two integers $\widehat{p}$ and $\tilde{p}$ and let these bounds support some bound $\theta_i \leq 1/n$ for all i. (This implies convergence with linear rate for the iterative refinement in Theorem 8.1.) Then we have the uniform bound $\widehat{p} + \log_2(n/(n-1))$ on the precision $p(U_{i+1})$ of the representation of all matrices U_{i+1} for all i. If the matrix C is filled with integers, then we also have the bound $\tilde{p}+\log_2(||C||n/(n-1))$.*

We cannot say a priori for which minimum precision bounds $\widehat{p}$ and $\tilde{p}$ we can ensure the progress in iterative refinement, but we can find these bounds dynamically, by first performing the computations with the IEEE standard double precision and then (if needed) increasing it recursively until convergence is observed. MSAs can handle any precision growth, but in our tests the growth was limited. We used the double precision for W_i and regularly observed that $s(U_{i+1}) < s(W_i) + \log_2 n$, which was in line with Lemma 8.4.

Counting arithmetic operations

To conclude this section, let us estimate the overall number of ops in our computations. Assume a normalized r-matrix A and a well conditioned A-modification $C = A + UV^H$. Then $||G|| = O(1/\operatorname{cond} A)$ (see the end of Section 5), and we yield the matrix G within the error norm bound ϵ by applying $O((\log \operatorname{cond} A)/\log(1/\epsilon))$ steps of iterative refinement. We need $O(M_{A,r})$ double precision ops per step and therefore $O((M_{A,r} \log \operatorname{cond} A)/\log(1/\epsilon))$ double precision ops overall provided we can multiply the matrix A by an $n \times r$ matrix in $M_{A,r}$ ops and have a crude approximation to the inverse matrix C^{-1}. The computational cost is low for smaller integers r and, if the matrices A, UV^H, C and G share their structure and are represented with short generators, then also for larger integers r (see Section 9).

9 Matrix structure in the Schur Aggregation

If the matrix A can be multiplied by a vector fast, then we can choose the generators U and V to extend this property to the A-modification $C = A+UV^H$.

Next recall Flowchart 4.1 for performing the Schur Aggregation and assume matrices A and A^- having the same structure and represented with short generators. Then our estimate for the overall computational complexity of A-preprocessing decreases by the factors of $r/\log^h r$ where h ranges from zero to two, depending on the structure (see [10], [11], [19], [22], [23], [33], [40, Chapters 1 and 4], [49], and the bibliography therein and in [61]).

We apply two principles to A-preconditioning for structured matrices.

- The operations in Flowchart 4.1 as well as the inversion of the matrix G can be reduced essentially to a small number of matrix multiplications and inversions, which we perform economically by operating with short generators of the structured input and auxiliary matrices rather than their entries.
- If the matrix A has structure, we rely on [40, Section 1.5] and Lemma 4.1 to extend this structure to the matrices involved in Flowchart 4.1 as well as to G^{-1} (whereas matrix structure is easily lost in the SVD-based APCs of larger ranks).

All our comments above can be readily extended to the dual APPs.

Various APPs with most frequently used matrix structures have been presented in [45, Examples 4.1–4.6]. Furthermore, we can apply the *method of displacement transformation* (see the remark below) to extend the power of these APPs to other classes of sparse and/or structured matrices, even to the classes that contain no well conditioned matrices and thus contain no well conditioned APPs [20], [59].

Remark 9.1. *By using appropriate structured multipliers, one can transform a matrix with the structure of a Cauchy, Vandermonde, Toeplitz, or Hankel type into a matrix with any other of these structures and can exploit such transforms to devise more effective algorithms. This method of displacement transformation was proposed in [38] (see its exposition also in [40, Sections 1.7, 4.8, and 4.9]). It was widely recognized due to the papers [21], [25], where the general class of Vandermonde-like multipliers in [38] was specialized to the Fourier transform multipliers, which transform the structures from the Toeplitz/Hankel into the Cauchy/Vandermonde types. This transform was used in [21], [25] for devising fast and numerically stable Gaussian elimination for Toeplitz/Hankel-like linear systems. For A-preconditioning, however, one should seek transition into the opposite direction, from Cauchy/Vandermonde-like matrices, which tend to be ill conditioned, to the Toeplitz/ Hankel-like structures. In this case the Fourier multipliers are not generally sufficient, but one can apply the original Vandermonde-like multipliers from [38].*

10 Multiplication/summation algorithms (an outline)

Effective MSAs in [12], [26], [31], [36], [54], and the bibliography therein compute the sum and products with double or k-fold precision for any k, but the computations slow down for $k > 2$. Additive preconditioning for linear systems of equations, however, leads us to operating with the sums $s = t_1 + \cdots + t_h$ that nearly vanish compared to $\max_j |t_j|$. Moreover, in some cases we need these sums error-free, which one can handle by using multi-precision arithmetic, with respective slow down of the computations. To avoid the slow down, we simulate multi-precision arithmetic with double precision computations. At this stage any effective MSA (e.g., an appropriate one in [12], [26], [31], [36], or [54]) can be incorporated into our construction. In the remainder of this section we outline a variant that employs our summation algorithm in [47], which extends the techniques of real modular reduction from [39] (see also [15]).

In our next comments on the resulting MSAs, "addition" can stand 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. Generally their sum is a multi-precision number, but it can be implicitly represented with the set "dpn-ν" by using double precision. Likewise, we can implicitly represent a $((p+1)\nu)$-bit floating point number with a dpn-ν where $p+1$ is the double precision.

The MSAs incorporate the Dekker's and Veltkamp's algorithms in [7] to compute the product of a dpn-μ and a dpn-ν error-free as a dpn-γ for $\gamma \leq 2\mu\nu$. To add a dpn-μ and a dpn-ν we just combine them into a dpn-$(\mu + \nu)$.

To save some memory space without losing accuracy, we perform *compressing summation* where we are given a dpn-μ whose absolutely larger elements may immensely exceed the absolute value of their sum. The compressing summation outputs a (compressed) dpn-ν for the nearly minimum $\nu < \mu$ that represents precisely the same sum.

We adopt compressing summation from [47], where we perform some sequences of usual floating-point additions rarely or never interrupted with the computation of the exponent of the current floating-point approximation of the sum of h numbers that we must compute. Namely, such interruptions never occur if we adopt rounding by chopping the least significant bits. If instead we use rounding to the closest number, then we should compute this exponent but only 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 apply in our MSAs. Accessing exponents of floating point numbers can be inexpensive. The IEEE floating point standard defines the function $\log b(x)$ to extract the significand and exponent of a floating point number (cf. [16], [28], [47]).

11 Numerical tests

We tested our Algorithm 7.1 for correctly computing determinants, and this also implied testing correctness of solving linear systems involved. For comparison we also computed the same determinants by applying the Matlab Subroutine **det**.

We generarted the input matrices $A = PML$ by following [53]. Here 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 $[-\eta, \eta]$ for a fixed positive η. It followed that $\det A = (-1)^k$. We generated such matrices for $k = 2n$ and $k = 2n - 1$ and for $\eta \geq 5,000$ and for $n = 4, 8, 16, 32, 64$.

We generated APCs UV^T according to the recipes in [41], [45] (also cf. [52]). We first randomly generated candidate APPs UV^H 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 following recipe from [41], [45], and [52],

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

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 $\eta = 5,000$, 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 η.

References

[1] O. Axelsson, *Iterative Solution Methiods*, Cambridge University Press, Cambridge, England, 1994.

[2] 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.

[3] M. Benzi, Preconditioning Techniques for Large Linear Systems: a Survey, *J. of Computational Physics*, **182**, 418–477, 2002.

[4] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science*, **210**, **1**, 173–197, 1999.

[5] K. Chen, *Matrix Preconditioning Techniques and Applications*, Cambridge University Press, Cambridge, England, 2005.

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

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

[8] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

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

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

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

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

[13] J. Demmel and Y. Hida, Fast and Accurate Floating Point Summation with Application to Computational Geometry, *Numerical Algorithms*, **37**, 101–112, 2004.

[14] I. Z. Emiris, V. Y. Pan, Improved Algorithms for Computing Determinants and Resultants, *J. of Complexity*, **21**, **1**, 43–71, 2005.

[15] 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.

[16] Agner Fog, *How to Optimize for the Pentium Family of Microprocessors*, www.agner.org, 1996–2004, last updated 2004-04-16.

[17] G. H. Golub, Some Modified Matrix Eigenvalue Problems, *SIAM Review*, **15**, 318–334, 1973.

[18] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[19] J. R. Gilbert, H. Hafsteinsson, Parallel Symbolic Factorization of Sparse Linear Systems, *Parallel Computing*, **14**, 151–162, 1990.

[20] W. Gautschi, G. Inglese, Lower Bounds for the Condition Number of Vandermonde Matrices, *Numerische Math.*, **52**, 241–250, 1988.

[21] I. Gohberg, T. Kailath, V. Olshevsky, Fast Gaussian Elimination with Partial Pivoting for Matrices with Displacement Structure, *Math. of Computation*, **64**, 1557–1576, 1995.

[22] I. Gohberg, V. Olshevsky, Complexity of Multiplication with Vectors for Structured Matrices, *Linear Algebra and Its Applications*, **202**, 163–192, 1994.

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

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

[25] G. Heinig, Inversion of Generalized Cauchy Matrices and the Other Classes of Structured Matrices, *Linear Algebra for Sigmal Processing, IMA Volume in Math. and Its Applications*, **69**, 95–114, 1995.

[26] N. J. Higham, *Accuracy and Stability in Numerical Analysis*, SIAM, Philadelphia, 2002 (second edition).

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

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

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

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

[31] 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 Math. Software*, **28**, 152–205, 2002.

http://crd.lbl.gov/ xiaoye/XBLAS/.

[32] 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.

[33] R. J. Lipton, D. Rose, R. E. Tarjan, Generalized Nested Dissection, *SIAM J. on Numerical Analysis*, **16**, **2**, 346–358, 1979.

[34] 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.

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

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

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

[38] V. Y. Pan, On Computations with Dense Structured Matrices, *Math. of Computation*, **55, 191**, 179–190, 1990.

[39] 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.

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

[41] V. Y. Pan, Null Aggregation and Extensions, Technical Report TR 2007009, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007.

[42] 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 2006006, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, May 2006.

[43] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning in Matrix Computations, Technical Report TR 2005009, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, July 2005.

[44] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning and Aggregation in Matrix Computations, Technical Report TR 2007002, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

[45] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning for Matrix Computations, Technical Report TR 2007003, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007.

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

[47] V. Y. Pan, B. Murphy, G. Qian, R. E. Rosholt, Error-free Computations via Floating-Point Operations, Technical Report TR 2007010, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007, and *Computers and Math. (with Applications)*, in press.

[48] V. Y. Pan, B. Murphy, G. Qian, R. E. Rosholt, I. Taj-Eddin, Numerical Computation of Determinants with Additive Preconditioning, Technical Report TR 2007011, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, April 2007.

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

[50] 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.

[51] V. Y. Pan, X. Yan, Additive Preconditioning, Eigenspaces, and the Inverse Iteration, Technical Report TR 2007004, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

[52] V. Y. Pan, X. Yan, Null Space and Eigenspace Computations with Additive Preprocessing, *Proceedings of the International Workshop on Symbolic–Numeric Computation (SNC 2007)*, 152–160, July 2007, London, Ontario, Canada, (Jan Verschelde and Stephen Watt eds.), ACM Press, New York, 2007.

[53] V. Y. Pan, Y. Yu, Certification of Numerical Computation of the Sign of the Determinant of a Matrix, *Algorithmica*, **30**, 708–724, 2001.

[54] 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*, *SIAM Journal on Scientific Computing*, in print.

[55] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Co., Boston, 1996 (first edition) and SIAM Publications, Philadelphia, 2003 (second edition).

[56] J. Shevchuk, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates, *Discrete and Computational Geometry*, **18**, 305–363, 1997, available at www.cs.cmu.edu/ quake/robust.html

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

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

[59] E. E. Tyrtyshnikov, How Bad Are Hankel Matrices? *Numerische Math.*, **67, 2**, 261–269, 1994.

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

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