Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 1870–1886
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Additive preconditioning and aggregation in matrix computations☆

Victor Y. Pan[a,c,*], Dmitriy Ivolgin[b], Brian Murphy[a], Rhys Eric Rosholt[a], Islam Taj-Eddin[b], Yuqing Tang[b], Xiaodong Yan[b]

[a] *Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA*
[b] *Ph.D. Program in Computer Science, The City University of New York, New York, NY 10036, USA*
[c] *Ph.D. Programs in Mathematics and Computer Science, The City University of New York, New York, NY 10036 USA*

Received 15 December 2003; accepted 16 March 2004

Abstract

We combine our novel SVD-free additive preconditioning with aggregation and other relevant techniques to facilitate the solution of a linear system of equations and other fundamental matrix computations. Our analysis and experiments show the power of our algorithms, guide us in selecting most effective policies of preconditioning and aggregation, and provide some new insights into these and related subjects. Compared to the popular SVD-based multiplicative preconditioners, our additive preconditioners are generated more readily and for a much larger class of matrices. Furthermore, they better preserve matrix structure and sparseness and have a wider range of applications (e.g., they facilitate the solution of a consistent singular linear system of equations and of the eigenproblem).

Keywords: Matrix computations; Additive preconditioning; Aggregation; MSAs

1. Introduction

1.1. Background: Multiplicative preconditioning

Multiplicative preconditioning is a popular technique for facilitating the solution of linear systems of equations $A\mathbf{y} = \mathbf{b}$. Originally, preconditioning meant the transition to equivalent but better conditioned linear systems

☆ Supported by PSC CUNY Awards 66437-0035, 67297-0036 and 68291-0037. Some results of this paper have been presented at the International Conferences on the Matrix Methods and Operator Equations in Moscow, Russia, in June 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.

* Corresponding author at: Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA.
E-mail addresses: victor.pan@lehman.cuny.edu (V.Y. Pan), divolgin@gc.cuny.edu (D. Ivolgin), brian.murphy@lehman.cuny.edu (B. Murphy), rhys.rosholt@lehman.cuny.edu (R.E. Rosholt), itaj-eddin@gc.cuny.edu (I. Taj-Eddin), ytang@gc.cuny.edu (Y. Tang), xyan@gc.cuny.edu (X. Yan).
URL: http://comet.lehman.cuny.edu/vpan/ (V.Y. Pan).

0898-1221/$ - see front matter
doi:10.1016/j.camwa.2004.03.022

$MA\mathbf{y} = M\mathbf{b}$, $AN\mathbf{x} = \mathbf{b}$, or more generally $MAN\mathbf{x} = M\mathbf{b}$ for $\mathbf{y} = N\mathbf{x}$ and readily computable nonsingular matrices M and/or N, called preconditioners. Such systems can be solved faster and/or more accurately (see [1–3] and the bibliography therein). A more recent alternative goal is the compression of the spectrum of the singular values of an input matrix A into a smaller number of clusters, so that one can solve the resulting linear systems more readily with the Conjugate Gradient (hereafter CG) or GMRES algorithms. We, however, pursue the original goal of decreasing the condition number of an input matrix.

Multiplicative preconditioners are closely linked to the Singular Value Decomposition (SVD) of an ill-conditioned input matrix, in particular to the costly computation of its smallest singular values and the associated singular vectors. Furthermore, the SVD-based preconditioners can easily destroy matrix structure.

1.2. Our alternative approach

As an alternative, we propose SVD-free *additive preprocessing* $A \leftarrow C = A + P$, i.e., we add a preprocessor P to an ill-conditioned matrix A to obtain its *additive modification* C. Hereafter we write "A-" for "additive", "*APPs*" for "A-preprocessors", and "*APCs*" for "A-preconditioners", which are APPs P such that the input matrix A is ill-conditioned but its A-modification $C = A + P$ is not. Compared to the SVD-based multiplicative preconditioners, our APCs are obtained more readily and for a much larger class of matrices, better preserve matrix structure and sparseness, and have a wider range of applications. In particular, they facilitate eigensolving and computations in the null spaces of matrices even more readily than the solution of nonsingular linear systems of equations.

1.3. Contents and organization of our paper

In this paper we outline A-preprocessing and its applications to the most fundamental matrix computations. Our outline covers also the auxiliary techniques of aggregation, extension of Wilkinson's classical iterative refinement (see Section 7, and briefly the advanced multiplication and summation algorithms, hereafter referred to as MSAs (see Section 8). In the papers [4–13] we elaborate upon this outline and present further technical details, proofs, and extensions (e.g., to the case of rectangular input matrices) as well as the results of numerical tests that show the power of our approach. The tests have been designed by the first author and performed by his coauthors, mostly by his first two and his last two coauthors. Otherwise all this work together with all typos and other errors is due to the first author.

We organize our presentation as follows. In the next section we state some definitions. In Section 3 we cover APPs, including random, general, sparse, structured, primal and dual ones, and their effect on conditioning. In Sections 4 and 5 we apply our A-preprocessing to some fundamental matrix computations. In Section 4 we cover the simpler case of APPs of ranks one and two and in Section 5 APPs of any rank. We briefly comment on preserving matrix structure in A-preprocessing and aggregation in Section 6 and on MSAs in Section 8. We devote Section 7 to extended iterative refinement. We discuss the preceding and further research in Sections 9 and 10, respectively. In the Appendix we comment on the impact of preconditioning on polynomial root-finding.

2. Definitions

Hereafter we write

- $(S_1, \dots, S_k)$ for a $1 \times k$ block matrix with the blocks $S_1, \dots, S_k$
- $\mathrm{diag}(S_1, \dots, S_k)$ for a $k \times k$ block diagonal matrix with the diagonal blocks $S_1, \dots, S_k$
- $0_{k,l}$ for the $k \times l$ null matrix, filled with zeros
- 0_k for $0_{k,k}$
- I_k for the $k \times k$ identity matrix
- A^{T} for the transpose of a matrix M
- A^H for the Hermitian (that is complex conjugate) transpose of a matrix A, so that $A^H = A^{\mathrm{T}}$ where A is a real matrix
- A^{-H} for $(A^H)^{-1} = (A^{-1})^H$
- $\sigma_j(A)$ for the jth largest singular value of a matrix A
- $\|A\| = \sigma_1(A)$ for its 2-norm
- $\rho = \mathrm{rank}\, A$ for its rank,

- nul A for its nullity (nul $A = n - \text{rank } A$ for an $n \times n$ matrix A), and
- cond $A = \sigma_1(A)/\sigma_\rho(A)$ for its condition number.

A matrix A is *ill conditioned* if the value cond A is large and is *well-conditioned* otherwise. The concepts "large" and consequently "ill" and "well" are quantified depending on the context of the computations and computer environment.

We call a matrix *normalized* if its 2-norm equals one. In addition to the abbreviations A-, APPs, and APCs, introduced earlier, we write *ACs* for *A-complements*, that is APPs P such that the input matrix A is rank deficient but its A-modification $C = A + P$ is not. We represent an APP P as the product UV^H of two *generators* U and V, typically the matrices of full rank.

3. A-preprocessing

3.1. Random A-preprocessors

In the paper [7] we first prove that for a matrix A with nullity r, a random APP P having rank of at least r is likely to be an AC, that is, to define a full rank matrix $C = A+UV^H$. Then we estimate how likely a random APP is an APC. Suppose a matrix A has full rank ρ and let $\sigma_{\rho-r}(A) \gg \sigma_{\rho-r+1}(A)$ or let $U = V$, $A = A^H$, and $\sigma_{\rho-r}(A) \gg \sigma_\rho(A)$. Then according to the analysis and tests in [7], we are likely to arrive at an A-modification $C = A + UV^H$ with cond C of the order of $\sigma_1(A)/\sigma_{\rho-r}(A)$ if an APP UV^H

(a) randomly chosen in a fixed class of general, sparse, or structured matrices,
(b) well-conditioned,
(c) has a rank of at least r, and
(d) properly scaled so that the ratio $\|A\|/\|UV^H\|$ is neither very large nor very small.

More precisely, we have proved a randomized upper bound of the order of $\sigma_1(A)/\sigma_{\rho-r}(A)$ for a Hermitian nonnegative definite input matrix A and an APP UU^H chosen according to the rules (a)–(d) above. We proved such a bound also for a general APP UV^H chosen according to the rules (a)–(d) and a matrix A that lies near a well-conditioned singular matrix $\tilde{A}$ that has a nullity of at most r. The results of our extensive tests are in very good accordance with the above upper bound also for the $n \times n$ matrices A that have singular values s^i for a fixed $s > 1$ and $i = 1, \ldots, n$.

At the end of Section 5.8 we show some further recipes for generating APCs.

Rule (d) allows some freedom in scaling the matrices A, U and V, and we can and should scale them by the powers of two to avoid rounding errors.

Clearly, an APC UV^H remains an APC in small-norm perturbations of the generator matrices U and V, and so we can truncate or round their entries to fewer bits to decrease or avoid rounding errors in the computation of the matrix $C = A + UV^H$.

In every recursive step of our randomized search for the threshold integer r, we update the matrices U, V, and C and estimate their condition numbers. We only need a random number generator and crude estimates for these condition numbers and for the ratios $\|A\|/\|UV^H\|$. Obtaining such information is not costly (see [14, Sections 2.3.2, 2.3.3, and 3.5.4] and [15, Section 5.3]).

In contrast to the power of random APCs, random multiplicative preprocessing cannot help much against ill conditioning because cond $A \le \prod_i \text{cond } F_i$ if $A = \prod_i F_i$.

Our randomized upper bounds of the order $\sigma_1(A)/\sigma_{\rho-r}(A)$ on cond C can be compared to the sharp lower bound $\sigma_{r+1}(A)/\sigma_{\rho-r}(A) \le \text{cond } C$ for any matrix A and rank-r APP UV^H and $\sigma_1(A)/\sigma_{\rho-r}(A) \ge \text{cond } C$, where both matrices A and UV^H are Hermitian and positive definite (see [16]). In contrast one can change the eigenvalues of a Frobenius companion matrix at will by applying rank-one modification.

3.2. Structured and sparse A-preprocessors

Our extensive tests in [7] suggest that we are likely to arrive at APCs UV^H for which cond C has the order $\sigma_1(A)/\sigma_{\rho-r}(A)$ even if we relax rule (a) above and choose very weakly randomized matrices U and V. Such randomization is compatible with various patterns of structure and sparseness. We refer the reader to [7, Examples 4.1–4.6] on various sparse and structured APCs, and here is [7, Example 4.6].

Example 3.1 (*Structured and Sparse Hermitian APPs*). Let $k, n_1, \dots, n_k$ be positive integers (fixed or random) such that $kr + n_1 + \cdots + n_k = n$. For $i = 1, \dots, k$, let $0_{r,n_i}$ denote the $r \times n_i$ matrices filled with zeros and let T_i denote some $r \times r$ (fixed or random) structured and/or sparse well-conditioned matrices, e.g., the matrices of the discrete Fourier, sign or cosine transforms, matrices with a fixed displacement structure (e.g., Toeplitz, triangular Toeplitz, circulant, Hankel, or Cauchy matrices), semi-separable (rank structured) matrices, sparse matrices with fixed patterns of sparseness, or in the simplest case just the scaled identity matrices $c_i I_r$ (see [17–21] and the bibliography therein and in [22]). Let $U = P(T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T$. Choose an $n \times n$ permutation matrix P (in the simplest case let $P = I$) and define the APP UU^H.

In some applications we can generate the desired (e.g., sparse and/or structured) APCs by using neither SVD nor randomization. For example (see Acknowledgements), with a rank-one APC we can increase the absolute value of a small pivot entry in Gaussian elimination and Cyclic Reduction algorithms without destroying matrix structure. Likewise, with rank-r APCs we can improve conditioning of $r \times r$ pivot blocks of block Gaussian elimination and block Cyclic Reduction.

Finally, a simple way of creating APCs that preserve the structure of an input matrix (e.g., a block Toeplitz matrix with Toeplitz blocks) is by appending new structured blocks of rows and columns. We call this technique *AA-preprocessing*, "AA" standing for "additive" and "appending".

3.3. Dual A-preprocessing

For a nonsingular $n \times n$ matrix A we can add a *dual* APP VU^H of a rank $q \le n$ to the matrix A^{-1} and define the *dual A-modification* $C_- = A^{-1} + VU^H$. We can compute the matrices C_- and then $A^{-1} = C_- - VU^H$ by inverting the matrix

$$(C_-)^{-1} = (A^{-1} + VU^H)^{-1} = A - AVH^{-1}U^H A, \quad H = I_q + U^H AV. \tag{3.1}$$

We call the latter expressions the *dual SMW inversion formula*, which is our simple counterpart to the *primal SMW inversion formula* of Sherman, Morrison, and Woodbury [14, page 50], [15, Corollary 4.3.2],

$$A^{-1} = (C - UV^H)^{-1} = C^{-1} + C^{-1}UG^{-1}V^H C^{-1}, \quad G = I_r - V^H C^{-1}U. \tag{3.2}$$

If we only seek the solution $\mathbf{y}$ to a linear system $A\mathbf{y} = \mathbf{b}$, we can bypass the inversion of the matrix $(C_-)^{-1}$ by applying the formula

$$\mathbf{y} = A^{-1}\mathbf{b} = \mathbf{z} - VU^H\mathbf{b}, \qquad ((C_-)^{-1})^{-1}\mathbf{z} = \mathbf{b}. \tag{3.3}$$

By extending our analysis of A-preconditioning, we obtain that $\operatorname{cond} C_- = \operatorname{cond}((C_-)^{-1})$ is likely to be of the order of the ratio $\sigma_{q+1}(A)/\sigma_n(A)$ if $\sigma_q(A) \gg \sigma_q(A)$ or if $U = V$ and the matrix A is Hermitian and positive definite, in both cases provided a dual APC VU^H of a rank q has been chosen according to rules (a)–(c) in Section 3.1 together with the following counterpart of rule (d),

(e) the ratio $\|A^{-1}\|/\|VU^H\|$ is neither large nor small.

Such a choice requires crude estimates for the smallest singular value $\sigma_n(A)$, versus the largest one in the case of primal A-preconditioning.

Based on the above observations, we readily extend our recipes for computing APCs to computing dual APCs.

4. From APPs to the output. The case of APPs of ranks one and two

In this section and the next one, we facilitate the solution of some fundamental problems of matrix computations provided some pairs of suitable APPs and A-modifications are available. For simplicity we assume square input matrices and, in this section, cover the simpler case of APPs of ranks one and two.

4.1. The Schur aggregation

Assume a nonsingular but ill-conditioned linear system $A\mathbf{y} = \mathbf{b}$ and a rank-one APP $UV^H = \mathbf{u}\mathbf{v}^H$ such that the A-modification $C = A + UV^H$ is nonsingular and well-conditioned. Then SMW formula (3.2),

$$A^{-1} = (C^{-1} - \mathbf{u}\mathbf{v}^H)^{-1} = C^{-1} + C^{-1}\mathbf{u}(1 - \mathbf{v}^H C^{-1}\mathbf{u})^{-1}\mathbf{v}^H C^{-1},$$

reduces the solution of the linear system to well-conditioned computations except for the stage of computing the value $g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}$. This value is absolutely small under the above assumptions about the matrices A and C (see Theorem 5.1 in Section 5.1), and so its computation cancels its leading significant bits. We overcome the problem by extending the iterative refinement algorithm (Section 7) and applying MSAs (Section 8).

The scalar $g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}$ is the Gauss transform of the 2×2 block matrix $\begin{pmatrix} C & \mathbf{u} \\ \mathbf{v}^H & 1 \end{pmatrix}$ and is also called the Schur complement of its block C. For $n > 1$, this scalar is a *Schur aggregate*, and the reduction to its computation from our original task is the *(primal) Schur Aggregation*.

If $\operatorname{cond} A \gg \operatorname{cond}(\mathbf{b}, A)$, then we write $\mathbf{u} = \mathbf{b}$ and choose a random vector $\mathbf{v}$ scaled to satisfy requirement (d) in Section 3.1. In this case we can expect that $\operatorname{cond} C \ll \operatorname{cond} A$ for $C = A + \mathbf{u}\mathbf{v}^H$. Whenever we achieve the desired decrease in the condition number by choosing an APP $\mathbf{u}\mathbf{v}^H = \mathbf{b}\mathbf{v}^H$, we can simplify the expression for the solution $\mathbf{y}$ as follows,

$$\mathbf{y} = C^{-1}\mathbf{u}/g. \tag{4.1}$$

Now suppose $\sigma_1(A) \gg \sigma_2(A)$ and the ratio $\sigma_2(A)/\sigma_n(A)$ is not large. Then we define the dual A-modification $C_- = A^{-1} + \mathbf{v}\mathbf{u}^H$. According to Section 3.1, the matrix C_- is likely to be well-conditioned for two random properly scaled vectors $\mathbf{u}$ and $\mathbf{v}$. Instead of its direct computation, however, we first compute the inverse

$$h(C_-)^{-1} = h(A^{-1} + \mathbf{u}\mathbf{v}^H)^{-1} = hA - A\mathbf{v}\mathbf{u}^H A, \quad h = 1 + \mathbf{u}^H A\mathbf{v}, \tag{4.2}$$

and then the solution to the linear system $A\mathbf{y} = \mathbf{b}$ as follows,

$$\mathbf{y} = A^{-1}\mathbf{b} = \mathbf{z} - \mathbf{v}\mathbf{u}^H\mathbf{b}, \tag{4.3}$$

$$h(C_-)^{-1}\mathbf{z} = h\mathbf{b} \tag{4.4}$$

(cf. Eqs. (3.1) and (3.3) for $U = \mathbf{u}$). Besides computing the reciprocal of the scalar h and the inversion of a well-conditioned matrix $(C_-)^{-1}$, we only use additions and multiplications, which we can perform error-free by applying MSAs. The value h is the *dual Schur aggregate*, and its computation is the *dual Schur Aggregation*.

4.2. Computations in the null space of a matrix (solution of singular linear systems of equations)

Given an $n \times n$ matrix A of rank $n-1$, suppose we seek its nonzero null vector $\mathbf{y}$, such that $A\mathbf{y} = \mathbf{0}$. Let a rank-one APP $\mathbf{u}\mathbf{v}^H$ define a nonsingular A-modification $C = A + \mathbf{u}\mathbf{v}^H$. Then $A\mathbf{y} = \mathbf{0}$ for $\mathbf{y} = C^{-1}\mathbf{u}$, so that the problem is essentially reduced to solving a nonsingular linear system of equations $C\mathbf{y} = \mathbf{u}$. Now recall from Section 3.1 that the ratios $\sigma_n(C)/\sigma_{n-1}(A)$ and $(\operatorname{cond} C)/\operatorname{cond} A$ are likely to be neither large nor small for a pair of properly scaled random vectors $\mathbf{u}$ and $\mathbf{v}$. In this likely case, the A-modification C is well-conditioned if and only if so is the matrix A, and the transform $A \to C$ removes singularity with no sacrifice in numerical stability [8,12].

Now suppose the ratio $\sigma_1(A)/\sigma_{n-2}(A)$ is not large, but $\sigma_{n-2}(A) \gg \sigma_{n-1}(A)$. Then we have complication because the above technique defines ill-conditioned A-modification. We can, however, counter this defect by choosing an APC of rank two. Namely, for a pair of properly scaled $n \times 2$ well-conditioned random matrices $U = (\mathbf{u}, \mathbf{u}_1)$, $V = (\mathbf{v}, \mathbf{v}_1)$ and for the A-modifications $C = A + \mathbf{u}\mathbf{v}^H$ and $C_1 = C + \mathbf{u}_1\mathbf{v}_1^H = A + UV^H$, we can expect that $\sigma_{n-1}(C) \gg \sigma_n(C)$ but the ratios $\sigma_1(C)/\sigma_{n-1}(C)$ and $\sigma_1(C_1)/\sigma_n(C_1)$ are not large.

We deduce that $A\mathbf{y} = \mathbf{0}$ for the vectors $\mathbf{y} = C_1^{-1}U\mathbf{x}$ and $\mathbf{x} \neq \mathbf{0}$ such that $AC_1^{-1}U\mathbf{x} = \mathbf{0}$. This expresses a null vector $\mathbf{y}$ of an $n \times n$ matrix A via a null vector for the $n \times 2$ *null aggregate* $AC_1^{-1}U$. We call this technique the *Null Aggregation*. Numerical properties of the Null Aggregation are quite similar to the properties of the Schur Aggregation in Sections 4.1 and 5.1 because $AC_1^{-1}U = UG$ where $G = I_2 - V^H C_1^{-1}U$, and so the original problem is reduced to the case of 2×2 input if the matrix U has full rank two. The homogeneous linear system $G\mathbf{x} = \mathbf{0}$

has a nontrivial solution $\mathbf{x}$. Numerically, we should approximate the vector $\mathbf{x}$ by applying the orthogonalization and least-squares methods [14, Chapter 5], [15, Chapter 4], [23,24], but we must first compute the matrix G with a high precision, overcoming the cancellation of many leading significant bits in its diagonal entries.

4.3. Extension to eigensolving

An eigenvector of an $n \times n$ matrix A associated with an eigenvalue λ is a null vector of the matrix $\lambda I - A$, and we can incorporate A-preconditioning and the Null Aggregation into the inverse power (Rayleigh quotient) iteration for refining an eigenvalue/eigenvector pair. (Hereafter we use the abbreviation *IIRQ*.) The iteration effectively approximates a single eigenvalue or a cluster of eigenvalues separated from the other eigenvalues of an input matrix (cf. [25, Section 4.4]). Every iteration step of IIRQ essentially amounts to the solution of an ill-conditioned linear system of equations. A-preconditioning turns it into a well-conditioned linear system, which can be rapidly solved with the CG algorithms. They are particularly efficient for sparse and/or structured input matrices A. In spite of such a simplification of every IIRQ step, we do not need to increase the number of these steps, according to our analysis, whose results we have confirmed by extensive tests [8].

5. Extension to general ill-conditioned input matrix

Let us extend our methods to $n \times n$ nonsingular ill-conditioned matrices A with $\sigma_{n-k}(A) \gg \sigma_n(A) > 0$ or $\sigma_1(A) \gg \sigma_k(A)$ for $k > 1$. In this case we must use APPs of larger ranks to yield well-conditioned A-modifications C or C_-, and so the sizes of the Schur and null aggregates increase. Otherwise the extension is quite straightforward unless we run into ill-conditioned aggregates. (Surely this can occur where $\sigma_{i+1}(A) \gg \sigma_i(A)$ for more than one subscript i, but for larger dimensions n also where, say, $2 \le \sigma_{i+1}(A)/\sigma_i(A) \le 3$ for all i, in which case $\operatorname{cond} A \ge 2^{n-1}$.) If so, we must overcome some additional numerical problems. Next we outline the respective modifications of our aggregation methods. As in the previous section, we assume square matrices A.

5.1. The Schur aggregation

Suppose we have computed an APC UV^H of a rank $r < n$ and a well-conditioned nonsingular A-modification $C = A + UV^H$ for an ill-conditioned nonsingular $n \times n$ input matrix A. Now SMW formula (3.2) reduces the linear system $A\mathbf{y} = \mathbf{b}$ to the $r + 1$ linear systems $C(W, \mathbf{z}) = (U, \mathbf{b})$ and the n linear systems of equations $GX^H = V^H$, $G = I_r - V^H C^{-1} U$. The matrix $G = I_r - V^H C^{-1} U$ is the Gauss transform of the 2×2 block matrix $\begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix}$ and the Schur complement of its block C. We call it a Schur aggregate and the transition to it the (primal) Schur Aggregation.

Suppose we wish to solve a linear system $A\mathbf{y} = \mathbf{b}$ and can achieve the desired decrease in the condition number by choosing an APP UV^H and a vector $\mathbf{c}$ such that $\mathbf{b} = U\mathbf{c}$. Then we can extend Eq. (4.1) and simplify Eq. (3.2) respectively:

$$\mathbf{y} = C^{-1} U G^{-1} \mathbf{c}, \quad G = I_r - V^H C^{-1} U. \tag{5.1}$$

The following results in [11] relate the singular values of the matrices A, C, and G to each other.

Theorem 5.1 (*[11, Theorem 6.3]*). *For two positive integers n and $r < n$, an $n \times n$ matrix A, and a pair of $n \times r$ matrices U and V, write $C = A + UV^{\mathrm{T}}$ and $G = I_r - V^{\mathrm{T}} C^{-1} U$. Suppose the matrices A and $C = A + UV^{\mathrm{T}}$ have full rank $\rho \ge r$. Then the matrix G is nonsingular, and we have*

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

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

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

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

Suppose the matrix A is ill conditioned solely because of the gap $\sigma_{n-r}(A) \gg \sigma_{n-r+1}(A)$, that is the ratios $\sigma_1(A)/\sigma_{n-r}(A)$ and $\sigma_{n-r+1}(A)/\sigma_n(A)$ are not large. Let an APP UV^H be chosen according to rules (a)–(d) in Section 3.1. Then the matrix $C = A + UV^H$ is likely to be well-conditioned (cf. Section 3.1). If so, the matrix G is also well-conditioned by virtue of Corollary 5.1. Therefore, our A-preconditioning confines the original numerical problems to the computation of the Schur aggregate $G = I_r - V^H C^{-1} U$ of a small norm, and we solve them by means of extending iterative refinement and applying MSAs.

If the ratio $\sigma_{n-r+1}(A)/\sigma_n(A)$ is large, then the matrices C and/or G are still ill conditioned. For larger ranks r the A-modification C is likely to be well-conditioned, but the aggregate G is not, whereas this property is reversed for smaller ranks r. (If $r = 1$, then cond $G = 1$, but cond C can be large.) A natural choice is the minimum rank for which the matrix $C = A + UV^H$ is well-conditioned. Under this choice all numerical problems are confined to computing and inverting the aggregate G, rather than to inverting the matrix C of a larger size. We can compute the matrices $C^{-1}U$ or $V^H C^{-1}$ and then G by combining the extended iterative refinement and MSAs, and we can recursively apply A-preconditioning and aggregation to invert this matrix.

5.2. The dual Schur aggregation

The dual Schur Aggregation is the Schur Aggregation associated with dual A-preprocessing, that is A-preprocessing of the matrix A^{-1}. Suppose we have computed a dual APC VU^H of a rank $q < n$ and the inverse $(C_-)^{-1} = A - AVH^{-1}U^H A$ of the dual A-modification $C_- = A^{-1} + VU^H$ (cf. Eq. (3.1)), where the matrix $H = I_q + U^H AV$ is the dual Schur aggregate. Then Eqs. (3.1) and (3.3) reduce a linear system $A\mathbf{y} = \mathbf{b}$ to linear systems with the coefficients given by the dual Schur aggregate H, whose condition number has the order of the ratio $\sigma_1(A)/\sigma_q(A)$.

Let the matrix C_- and its inverse be well-conditioned. Then, unless the latter ratio is large, the dual Schur aggregate H is also well-conditioned, and similarly to the case of primal Schur Aggregation, we confine the remaining numerical problems to the computation of this aggregate. We can overcome them by applying MSAs. Unlike the primal case, we compute the aggregate H with no matrix inversions and have no need for iterative refinement.

If the ratio $\sigma_1(A)/\sigma_q(A)$ is large, then the dual Schur aggregate H is ill conditioned, and we can reapply our techniques recursively to facilitate its inversion (see some details in Section 5.4).

In the dual aggregation the $q \times q$ aggregate H is computed involving no inverse of the A-modification $(C_-)^{-1}$. We invert the aggregate to compute the matrix $(C_-)^{-1}$, but in Algorithm 5.4 in Section 5.4 we avoid using divisions even at this stage.

The price of such an advantage versus the primal processes is the extra work for the crude estimation of the norms of the inverses (rather than of the matrices themselves) at all recursive steps.

5.3. Extension to computing determinants

Our recipes for solving linear systems of equations are readily extended to the highly important and extensively studied problem of computing the value and the sign of a matrix determinant. We refer the reader to [10,26–30], and the references therein for the extensive bibliography on the applications to the fundamental geometric and algebraic–geometric computations and on the algebraic and numerical computation of the determinants. Our algorithms from the two previous subsections can also readily handle the computation of determinants due to the following simple modifications of the SMW formulas (3.1) and (3.2),

$$\det A = (\det G) \det C = (1/\det H) \det((C_-)^{-1}). \tag{5.2}$$

5.4. Recursive primal and dual A-preconditioning and Schur aggregation

Next we summarize the previous sections by explicitly expressing the recursive process of primal and dual A-preconditioning and Schur aggregation. We assume some fixed policies RANK and DUAL RANK for the choice of the ranks r and q of the primal and dual APPs, respectively, generated in each recursive step. 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 can readily compute error-free the scalar $h = \det H$ and the adjoint matrix hH^{-1} (cf. Algorithms 5.3 and 5.4 and Remark 5.1).

We can round the entries of the matrices U and V 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. [7]).

Computation of the determinant in Algorithms 5.1, 5.2 and 5.4 below is optional and can be dropped.

Algorithm 5.1 (*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 v for the norm $\|A\|$.
3. Compute the matrix $U = v\tilde{U}$.
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 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 5.2 (*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 5.1).
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 v for the norm $\|A^{-1}\| = \operatorname{cond} A/\|A\|$.
3. Compute the matrix $U = v\tilde{U}$.
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 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 $U\mathbf{c} = \mathbf{b}$ for some vector $\mathbf{c}$ and use the simplified expression (5.1).

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 5.3 (*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 5.1), 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: $i \leftarrow 1$, $A_0 \leftarrow A$.
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 a crude estimate ν_i for the norm $\|A_{i-1}^{-1}\|$.
3. Compute the matrix $U_i = \nu_i \tilde{U}_i$.
4. Compute the $q_i \times q_i$ matrix

$$H_i = I_q + U_i^H A_{i-1} V_i. \tag{5.3}$$

5. Compute the scalar $h_i = \det H_i$ and the $q_i \times q_i$ matrix $\tilde{H}_i = \text{adj } H_i = h_i H_i^{-1}$.
6. Compute the matrix

$$A_i = h_i A_{i-1} - A_{i-1} V_i \tilde{H}_i^{-1} U_i^H A_{i-1}. \tag{5.4}$$

7. Compute a crude estimate κ_i for the condition number $\text{cond } A_i$. 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. Write $h_0 \leftarrow 1$ and recursively for $i = r, r-1, \ldots, 1$ compute the scalars $\det(A_{i-1}) = (1/h_i)\det(A_i/h_i)$ and the vectors $\mathbf{y}_{i-1} = h_i \mathbf{y}_i - V_i U_i^H \mathbf{b}$.
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 problems for the matrix A_r, which is supposed to be better conditioned due to A-preprocessing in Eqs. (5.3) and (5.4). 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 A5, and the norm and condition estimation at Stages A2 and A7, the algorithm is division-free.

Correctness of the algorithm follows from Eqs. (3.1), (3.3) and (5.2).

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 5.4 (*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: $i \leftarrow 1$, $A_0 \leftarrow A$, and $\mathbf{b}_0 \leftarrow \mathbf{b}$.
COMPUTATIONS:

STAGE A.

1. Generate the pair of nth dimensional normalized vectors $\tilde{\mathbf{u}}_i$ and $\mathbf{v}_i$, $\|\tilde{\mathbf{u}}_i\| = \|\mathbf{v}_i\| = 1$.
2. Compute a crude estimate ν_i for the norm $\|A_{i-1}^{-1}\|$.
3. Compute the vector $\mathbf{u}_i = \nu_i \tilde{\mathbf{u}}_i$.
4. Compute the scalar $h_i = 1 + \mathbf{u}_i^H A_{i-1} \mathbf{v}_i$.
5. Compute the matrix $A_i = h_i A_{i-1} - A_{i-1}\mathbf{v}_i \mathbf{u}_i^H A_{i-1}$.
6. Compute a crude estimate κ_i for the condition number $\text{cond } A_i$. 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. Write $h_0 = 1$ and recursively for $i = r, r-1, \ldots, 1$ compute the scalars $\det(A_{i-1}) = (1/h_i)\det(A_i/h_i)$ and the vectors $\mathbf{y}_{i-1} = h_i \mathbf{y}_i - \mathbf{v}_i \mathbf{u}_i^H \mathbf{b}$.
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 the norm and condition estimation at Stages A2 and A6, the algorithm is division-free.

Remark 5.1. By minimizing the rank q_i of the APPs in each step i we avoid the extra work for computing determinants h_i and adjoints $h_i H_i^{-1}$ but increase the number r of recursive steps. Each step requires estimating the norm and condition number and increases the displacement rank of the A-modifications (cf. [17] and Section 6). This suggests the following policy DUAL RANK at the ith recursive step: increment the value q_i recursively until the cost of computing the determinant and inverse at Stage A4 of Algorithm 5.2 or the determinant and adjoint at Stage A5 of Algorithm 5.3 exceeds the cost that we can save due to the decrease in the number of recursive steps.

5.5. Computations in the null space and eigenspace

Let us first extend our study in Section 4.2. Let A be an $n \times n$ singular matrix that has a rank $n - r$ and the nullity $\operatorname{nul} A = r$. Suppose UV^H is its (primal) APP of the rank r and the A-modification $C = A + UV^H$ is nonsingular. Then the matrix $V^H C^{-1}$ (resp. $C^{-1}U$) is a left (resp. right) *null matrix basis* for the matrix A, that is the rows (resp. columns) of this matrix span the left (resp. right) null space of the matrix A. The value $\operatorname{nul} A$ can be found by the discrete try-and-error search based on the following properties (i)–(iv): $\operatorname{nul} A$ is the minimum integer r such that for an APP UV^H of rank r

(i) the matrix $C = A + UV^H$ can be nonsingular,

(ii) is likely to be nonsingular if the APP is random,

(iii) $AC^{-1}U = 0$ provided the matrix $C = A + UV^H$ is nonsingular, and

(iv) we are likely to have $AC^{-1}U\mathbf{x} = \mathbf{0}$ for a random vector $\mathbf{x}$ provided the matrix $C = A + UV^H$ is nonsingular.

By combining properties (i) and (ii) with (iii) and (iv), we can compute the nullity $\operatorname{nul} A$ faster. E.g., we can first recursively test if the matrix C_i is nonsingular for $i = 0, 1, 2, 4, 8, \ldots$. If this first occurs for $i \le 2$ we output $\operatorname{nul} A = i$ and stop. If this first occurs for $i \ge 4$, we apply binary search for $\operatorname{nul} A$ based on the properties (i)–(iv) above.

If the singular matrix A is well-conditioned, then so is likely to be the nonsingular A-modification C as well, provided the APP UV^H is chosen according to rules (a)–(d) in Section 3.1. In this case A-preprocessing would remove singularity with no numerical sacrifice.

If $\operatorname{rank}(UV^H) = \operatorname{nul} A$ and if the matrix A is ill conditioned, then so is the A-modification $A + UV^H$, and we face numerical problems when we test its nonsingularity. To counter them we can recursively generate well-conditioned APPs $U_k V_k^H$ of recursively increasing rank until we arrive at a well-conditioned A-modification $C = A + U_k V_k^H$ (of full rank).

At this point the row (resp. column) span of the null aggregate $V^H C^{-1}$ (resp. $C^{-1}U$) contains the left (resp. right) null space of the matrix A. Moreover, we can obtain a left (resp. right) null matrix basis $Z^H V^H C^{-1}$ (resp. $C^{-1}UX$) for the matrix A as soon as we obtain a left (resp. right) null matrix basis Z^H (resp. X) for the null aggregate $V^H C^{-1} A$ (resp. $AC^{-1}U$), which has a smaller size. This is the Null Aggregation, having simple links to the Schur Aggregation if the matrix V (resp. U) has full rank. Indeed in this case we have

$$V^H C^{-1} A = GV^H \text{ (resp. } AC^{-1}U = UG) \quad \text{for } G = I_r - V^H C^{-1} U, \tag{5.5}$$

and we can compute the matrix basis Z^H (resp. X) as a left (resp. right) null matrix basis for the $r \times r$ Schur aggregate G. Numerically, we should compute the matrix Z^H (resp. X) by applying the orthogonalization and least-squares methods, but first we must approximate the matrix G with a high precision, to counter the cancellation of the leading significant bits in its entries. Then again we apply extended iterative refinement and MSAs, and if the matrix C is ill conditioned, we can invoke the primal and/or dual A-preconditioning and aggregation. If we need, we can extend the process recursively.

We study the computation of the right null space of a square matrix, but this can be readily extended. Indeed the left and right null spaces are linked via the equation $LN(A) = (N(A^{\mathrm{T}}))^{\mathrm{T}}$. Furthermore the case of $m \times n$ matrices A where $m \le n$ is reduced to the case where $m = n$ due to the following simple fact.

Fact 5.1. *We have $N(A) = N(B^H A)$ for a pair of $m \times n$ matrices A and B where $m \le n$ and* (a) *B is a full rank matrix or* (b) *$B = A$.*

For $m < n$ and $B = (I_m, 0)$, the transition $A \to B^{\mathrm{T}}A$ means just appending the $n - m$ rows of zeros at the bottom of the matrix A. For $B = A$, the matrix $B^H A = A^H A$ is positive definite with the condition number equal to $(\operatorname{cond} A)^2$.

We also recall the customary transition from an $m \times n$ matrix A to the $(m + n) \times (m + n)$ Hermitian indefinite matrix $\tilde{A} = \begin{pmatrix} 0 & A^H \\ A & 0 \end{pmatrix}$. By projecting all vectors in the null space $N(\tilde{A})$ into their leading subvectors of dimension n, we arrive at the null space $N(A)$. In this case $\operatorname{rank} \tilde{A} = 2 \operatorname{rank} A$, so that $\operatorname{nul} \tilde{A} = 2n - 2 \operatorname{rank} A = 2 \operatorname{nul} A$ for $m = n$.

Finally recall that the eigenspace associated with an eigenvalue λ of an $n \times n$ matrix A is the null space of the matrix $\lambda I_n - A$. Therefore we can readily extend both our approach to approximating the eigenspaces and our comments in Section 4.3. If the singular matrix $\lambda I_n - A$ with $\operatorname{nul}(\lambda I_n - A) = r$ is well-conditioned or, more generally, if λ represents a cluster of r eigenvalues isolated from the other eigenvalues, then our rank-r A-preprocessing can eliminate both singularity and ill conditioning. In particular we can achieve this by incorporating our A-preprocessing into the IIRQ, extended to computing bases for the respective eigenspace rather than just an eigenvector.

5.6. Extension to the solution of a linear system of equations

Clearly, the solution $\mathbf{y}$ to a linear system $A\mathbf{y} = \mathbf{b}$ can be computed as a subvector of a null vector $\tilde{\mathbf{y}} = (1/a, \mathbf{y}^{\mathrm{T}})^{\mathrm{T}}$ of the matrix $(-a\mathbf{b}, A)$, $a \approx \|A\|$, of a slightly larger size. We can map $A \leftarrow B^H A$ for an $n \times (n+1)$ matrix B, e.g., we can append a zero row to the matrix $(-a\mathbf{b}, A)$, to turn it into a square matrix $\tilde{A}$. Then we would choose generator matrices $\tilde{U}$ and $\tilde{V}$ of an appropriate full rank $\tilde{r}$, compute the A-modification $\tilde{C} = \tilde{A} + \tilde{U}\tilde{V}^H$, and narrow the search for the vector $\tilde{\mathbf{y}}$ to the space $\operatorname{range}(\tilde{C}^{-1}\tilde{U}) = N(\tilde{G})$ of a smaller dimension $\tilde{r}$ where $\tilde{G} = I_{\tilde{r}} - \tilde{V}^H \tilde{C}^{-1}\tilde{U}$.

5.7. The tail and head approximation

To describe another extension, suppose we apply the Null Aggregation to an $n \times n$ nonsingular but ill-conditioned matrix A that has exactly r small singular values. By zeroing them, we arrive at a singular well-conditioned $n \times n$ matrix $\tilde{A} \approx A$ having the nullity $r = n - \operatorname{rank} \tilde{A}$. For random and properly scaled APPs UV^H of rank r and for $C = A + UV^H$, the ranges of the aggregates $V^H C^{-1}$ and $C^{-1}U$ (that is their respective row and column spans) approximate the pair of left and right null spaces of the matrix $\tilde{A}$ and therefore the *r-tail* of the SVD of the matrix A, that is the pair of the left and right singular spaces associated with its r smallest singular values. Thus we call computation of the latter aggregates the *Tail Approximation*.

Dual A-preconditioning and A-modification extend this approach to the *Head Approximation*, that is to computing the aggregates $U^H(C_-)^{-1}$ of size $q \times n$ and $(C_-)^{-1}V$ of size $n \times q$. Their row (resp. column) span approximates some basis for the left (resp. right) singular spaces associated with the q largest singular values of the $n \times n$ matrix A provided all other $n - q$ singular values are small. We call this pair of left and right singular spaces the *q-head* of the SVD.

5.8. Refinement of APCs

Finally, if $\operatorname{cond} C$ for an A-modification $C = A + UV^H$ is too large, we are likely to decrease it if we recompute the APP by following rules (a)–(d) in Section 3.1. With a little more work and more confidence in success, however, we can apply the Null/Tail Approximation to improve an APC UV^H as follows,

$$(U \leftarrow Q(C^{-1}U), \qquad V \leftarrow Q(C^{-H}V)). \tag{5.6}$$

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.

Computation of the aggregates $C^{-1}U$ and $C^{-H}V$ is simpler where the matrix C is better conditioned, and we can more readily obtain a well-conditioned A-modification $C = A + UV^H$ if we choose an APP UV^H of a larger rank. As soon as we obtain such a well-conditioned A-modification C, we can extend transform (5.6) to obtain an APC of a smaller rank for which we still have $\operatorname{cond} C$ nicely bounded (cf. [4–9,12,16]). Specifically, assume 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, and proceed as follows.

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

2. *(The Tail Approximation.)* Compute two properly scaled and well-conditioned matrix bases $T(U)$ and $T(V)$ for the singular spaces of the matrices $AC^{-1}U$ and $A^HC^{-H}V$, respectively, associated with the r smallest singular values of these matrices. If U and V are unitary matrices, then the matrices $T(U)$ and $T(V)$ can be also computed as two matrix bases for the left and right singular spaces associated with the r smallest singular values of the matrix $G = I_h - V^HC^{-1}U$ (cf. Eq. (5.5)).
3. *(Compression.)* Update the generators $U \leftarrow Q(C^{-1}UT(U))$ and $V \leftarrow Q(C^{-H}VT(V))$. Output them and the new APC UV^H.

Extensive tests in [4–7,16] have confirmed the efficiency of these recipes.

6. A-preprocessing and aggregation for structured matrices

For a Hermitian input matrix A, we choose generators $U = V$ and obtain Hermitian APP UU^H, A-modification $C = A + UU^H$, and aggregate $G = I_r - U^HCU$, and similarly in dual A-preconditioning and aggregation. More generally, *all our A-preprocessing and nonrecursive aggregation methods allow us to preserve matrix structure*. More precisely, if an input matrix A has the displacement or semi-separable (rank) structure [17,22], then we can choose a pair of generators U and V with consistent structure (see our Example 3.1 and [7, Examples 4.1–4.4]). Such a structure can be preserved in a few matrix additions, multiplications, and inversions [17,22], and this is all that is required in the transition to the APP UV^H, the A-modification $C = A + UV^H$ (provided it has full rank), and the aggregates V^HC^{-1} and $C^{-1}U$.

For an $n \times n$ structured matrix A with r small singular values, we arrive at the structured matrices $C^{-1}U$ of size $n \times r$ and V^HC^{-1} of size $r \times n$, which approximate matrix bases for the singular spaces associated with the r smallest singular values of the matrix A, even where these spaces have no structured matrix bases, that is no full-rank structured matrices whose rows (resp. columns) span these spaces.

Similar comments apply to the dual Schur Aggregation and to the Head Approximation.

If we apply aggregation recursively, the structure of an $n \times n$ input matrix gradually deteriorates and can completely disappear in $O(\log n)$ recursive steps.

Our Example 3.1 and [7, Examples 4.1–4.4] show that we can generate APPs having all most popular matrix structures. Furthermore, the *method of displacement transformation* (see below) enables us 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 [31,32].

Remark 6.1. 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 [33] (see its exposition also in [17, Sections 1.7, 4.8, and 4.9]). It was widely recognized due to the papers [34,35], where the general class of Vandermonde-like multipliers in [33] 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 [34,35] for devising fast and numerically stable Gaussian elimination for Toeplitz/Hankel-like linear systems. For A-preconditioning, however, one should seek transforms 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 [33].

Finally, if A is a Hermitian matrix, we can choose generators $U = V$ and obtain Hermitian matrices C, G and H.

7. Extended iterative refinement

For an ill-conditioned matrix A and well-conditioned matrices C and G, the primal Schur Aggregation can lead us to the task of computing the Schur aggregates $G = I - V^HC^{-1}U$ that have very small norms $\|G\|$ (see Section 5.1). Cancellation of many leading significant bits of the entries of the matrix G poses a numerical challenge. As we have pointed out, we meet it with extended iterative refinement and MSAs. (Similar problems arise and similar solution recipes work for the dual Schur Aggregation, for which, however, we much less depend on iterative refinement.)

We extend Wilkinson's classical iterative refinement in [14,15], and [36, Chapter 11] to compute the matrix $W = C^{-1}U$ with high accuracy. (We can compute the matrix V^HC^{-1} instead). We do not store the computed

segments of bits in the binary representation of the entries of the matrix W but immediately employ them into the multiplication $V^H W$, and store the respective segments that represent the entries of the matrix $G = I_r - V^H C^{-1} U$. More precisely, we begin storing these segments as soon as we arrive at a nonvanishing approximation to the matrix G that remains stable in some consecutive steps of iterative refinement. In a number of the initial refinement steps, the leading bits of the entries of the matrix G are cancelled because its norm is small.

In our extended iterative refinement we fix a sufficiently large integer k such that $\sum_{i=0}^{k} W_i \approx W$ and $I_r + \sum_{i=0}^{k} F_i \approx G = I_r - V^{\mathrm{T}} W$ and compute the matrices W_i and F_i for $i = 0, 1, \ldots, k$, $\sum_{i=0}^{k} W_i$, and $I_r + \sum_{i=0}^{k} F_i$ as follows (cf. [11]). Write $U_0 = U$ and $G_0 = I_r$ and successively compute the matrices $W_i \leftarrow C^{-1} U_i$, $U_{i+1} \leftarrow U_i - C W_i$, $F_i \leftarrow -V^{\mathrm{T}} W_i$, 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 $U_i \leftarrow U - C W_{i-1}$, $E_i \leftarrow C^{-1} U_i$, 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.)

Here is our policy of rounding. We allow to perturb matrices U and V within a fixed small norm bound as long as this keeps the A-modification $C = A + UV^H$ well-conditioned. Likewise, in our computation of the matrices $W_i = C^{-1} U_i$ we allow any errors within a fixed small norm bound as long as this ensures that the residual norm $u_i = \|U_i\|$ decreases by at least a fixed factor $1/\theta > 1$ in each iteration.

Within these restrictions 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 restrictions. Then we truncate the remaining (nonzero) entries to decrease the number of bits in their representation as much as possible under the same restrictions.

Apart from the approximation of the matrices C^{-1} and W_i within some fixed error norm bounds, we perform all other arithmetic operations error-free, that is we allow no errors at the stages of computing the matrices $C \leftarrow A + UV^H$, $U_{i+1} \leftarrow U_i - C W_i$, $F_i \leftarrow -V^{\mathrm{T}} W_i$, and $G_{i+1} \leftarrow G_i + F_i$ for $i = 0, 1, \ldots, k$. At these stages, computing with the double precision can be insufficient for some input matrices A, but then we meet the challenge with MSAs.

Let us recall some error and precision estimates from [11, Section 7].

Theorem 7.1. *Consider the subiteration*

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

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

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

The theorem implies that the sum $W_0 + \cdots + W_k$ approximates the matrix $W = C^{-1} U$ with the error matrix $-E_k$. The next theorem shows that, under some natural assumptions, the error norm E_i converges to zero as $i \to \infty$.

Theorem 7.2. *Assume that*

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

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

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

where $\|M\|_F$ is the Frobenius norm of a matrix M, $\|M\|_F^2 = \mathrm{trace}(M^H M) = \sum_{i=1}^{\rho} \sigma_i^2(M)$, $\|M\| \le \|M\|_F \le \sqrt{\rho}\|M\|$ if $\mathrm{rank}\, M = \rho$. Then we have $e_i \le \delta_i u_i$ and $u_{i+1} \le \|C\| e_i$ for all i, $e_{i+1} \le \theta_i e_i$ and $u_{i+1} \le \theta_i u_i$ for $i = 0, 1, \ldots, k-1$.

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.

Suppose the matrix C is well-conditioned, and so the ratios $r_i = \|\tilde{E}_i\|_F/\|C\|_F$ are small and $\operatorname{cond}(C - \tilde{E}_i) \approx \operatorname{cond} C$ (cf. [14, Section 3.3], [15, Theorem 3.4.9], [36]). Then we have

$$\theta_i = \delta_i\|C\| \approx 2(\operatorname{cond} C)^2 r_i\|C\|_F/\|C\| \le 2(\operatorname{cond} C)^2 r_i n.$$

Therefore the values θ_i tend to be significantly less than one.

Finally we recall the estimates from [11, Section 7] for the precision required in our error-free computation of the residual matrices U_i.

For a finite precision binary number $b = \sigma\sum_{k=t}^{s} b_k 2^k$, where $\sigma = \pm 1$ 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 $s(M) = \max_{i,j} s(m_{i,j})$, $t(M) = \min_{i,j} t(m_{i,j})$, $p(M) = s(M) - t(M) + 1$.

Theorem 7.3. *Suppose $p(W_i) \le \widehat{p}$ and/or $p(CW_i) \le \tilde{p}$ and bound $\theta_i \le 1/n$ for two integers $\widehat{p}$ and $\tilde{p}$ and all i. (The latter bounds imply convergence with linear rate for the iterative refinement in Theorem 7.1.) Then the precision $p(U_{i+1})$ of the representation of the matrices U_{i+1} is at most $\widehat{p} + \log_2(n/(n-1))$ for all i. Furthermore this precision is at most $\tilde{p} + \log_2(\|C\|n/(n-1))$ if the matrix C is filled with integers.*

8. MSAs

The computation of the residuals U_i in the extended iterative refinement is division-free, and we can perform it by applying MSAs. We can apply MSAs to obtain the same output by computing with lower precision, e.g., in the case where the bound in Theorem 7.3 is excessively high.

More precisely, effective MSAs in [36–39] and the bibliography therein compute the sum and products with double or k-fold precision for any k, but the computations slow down for $k > 2$. We can avoid the slowdown by means of the double-precision simulation of multi-precision computations. The recent summation algorithm in [13] complements various techniques of this kind in [39–42] and in the bibliography therein.

The algorithm in [13] is tuned to computing nearly a vanishing sum, whose absolute value is negligible compared to the maximum absolute value of the summands. It combines Dekker's splitting algorithm in [43] with the techniques of real modular reduction from [44] (see also [45]) and solves the problem by performing double-precision additions and (rarely or even never) extracting the exponents of some floating-point binary numbers. The latter operation is noncostly [46,47], is not needed where we round binary numbers by chopping (truncating) the least significant bits, and is rarely needed where we round them to the nearest values.

MSAs can be applied to the evaluation of any polynomial and, in combination with algorithms that approximate reciprocals and with error analysis, can be extended to the approximate evaluation of a rational function, but we use them essentially just for computing sums and dot products.

9. The preceding study

Small-rank modification is a known tool for decreasing the rank of a matrix [48,49], fixing its small-rank deviations from the Hermitian, positive definite, and displacement structures, and supporting the divide-and-conquer eigensolvers [14,25,50], but these important works have not been linked to conditioning of the input and auxiliary matrices. The discussions that followed the presentations by the first author at the International Conferences on the Matrix Methods and Operator Equations in Moscow, Russia, in June 20–25, 2005, and on the Foundations of Computational Mathematics (FoCM'2005) in Santander, Spain, June 30–July 9, 2005, revealed only a few other touches to what we call A-preconditioning. They were sporadic and rudimentary versus our present work. We are aware of no earlier use of the nomenclature of A-preconditioning and APCs as well as of no attempts of devising and employing random and/or structured primal and dual APCs, adjusting Sherman–Morrison–Woodbury formula respectively, studying APCs and their affect on the rank and conditioning systematically, linking the APCs to aggregation, iterative refinement, and MSAs, improving APCs based on the Null Aggregation, or applying them to the null space computations and to numerical approximation of the bases for trailing singular spaces of ill-conditioned matrices.

We have introduced A-preconditioning to accelerate the steps of the inverse iteration for the algebraic eigenproblem, which we applied to a semi-separable generalized companion matrix, seeking the roots of the associated

polynomial [51,52]. Exploiting semi-separable matrix structure for polynomial root-finding was an innovation, which has become a popular research direction (cf. [53–55]).

10. Further research subjects

We have introduced new areas of A-preconditioning and aggregation and related them to some most fundamental matrix computations. Clearly, many subjects in these areas invite further theoretical and experimental study, e.g.,

- analysis and refinement of recursive numerical application of the Schur and Null Aggregation to singular and nonsingular ill-conditioned matrices having multiple jumps in the spectrum of their singular values
- combined application of the primal and dual Schur Aggregation to solving linear systems of equations and computing determinants
- the approximation and error analysis for the Tail and Head Approximation-based on A-preconditioning
- decreasing the running time and memory space in MSAs.

Recalling the aggregation methods in [56], based on multiplicative preconditioning and evolved into the *Algebraic Multigrid* in the 1980s, we now ask whether our A-preconditioning and aggregation methods will eventually evolve into *A-Algebraic Multigrid*. Can they be extended to yield other classes of effective preconditioning and/or aggregation methods?

Seeking some pointers to such extensions, we recall *trilinear aggregating* in [57]. The latter technique has been an indispensable ingredient in the design of the currently fastest algorithms for $n \times n$ matrix multiplication. This includes the fastest known algorithms for both immense dimensions n (cf. [58]) and moderate dimensions n from 20 to, say, 10^{20} (cf. [57,59]). See [60,61] on efficient numerical implementations.

Our approach can be naturally extended to the computation of matrix functions according to the map $f(A) \leftarrow g(C, U, V)$ for $C = A + UV^H$, e.g., $\exp(A) \leftarrow \exp(C)\exp(UV^H)$.

Our further research directions also include applications to solving systems of multivariate polynomial equations via elimination methods and reduction to the algebraic eigenproblem [62,63].

Acknowledgements

E. E. Tyrtyshnikov, S. A. Goreinov, and N. L. Zamarashkin from the Institute of Numerical Analysis of the Russian Academy of Sciences in Moscow, Russia, and B. Mourrain from the INRIA in Sophia Antipolis, France, provided the first author of this paper with the access to the computer and library facilities during his visits to their Institutes in 2005/06. X. Wang was the first reader of our papers on A-preconditioning and aggregation and has replied with his new advance in [16]. Helpful and encouraging were the interest and comments to our work from the participants of the cited Conferences in Moscow and Santander (particularly from J. W. Demmel, G. H. Golub, V. Olshevsky, L. Reichel, M. Van Barel, and the participant of FoCM'2005 who proposed the substitution of APCs for pivoting in the Gaussian elimination algorithm).

Appendix. Application to polynomial root-finding

Polynomial root-finding is an example of further applications of A-preconditioning and aggregation. This is a classical and highly developed subject but is still an area of active research [64,65]. Increasingly popular are the matrix methods for approximating the polynomial roots as the eigenvalues of an associated *generalized companion* matrix. Matlab computes relatively crude approximations to the polynomial roots by applying the QR eigensolver to the Frobenius companion matrix. Malek and Vaillancourt in [66,67] and Fortune in [68] recursively applied the QR algorithm to a diagonal plus rank-one (hereafter *DPR1*) generalized companion matrix, updating it whenever new approximations to the roots were computed. This process turned out to be highly effective.

A similar approach in [52] employed the IIRQ iteration instead of the QR iteration. This decreased the running time per iteration step from quadratic to linear due to the structure of the DPR1 matrices (and similarly for the Frobenius matrices and for the shift-and-invert enhancement of the Lanczos, Arnoldi, Jacobi–Davidson, and other eigensolvers). The idea of exploiting DPR1 matrix structure for polynomial root-finding, first succeeded in [52], has led to a popular research direction.

According to the test results in [52] the IIRQ iteration is quite effective for the DPR1 and Frobenius matrices, so that the algorithm is already slightly superior to the Durand–Kerner's (Weierstrass') celebrated root-finder. Application of A-preconditioning and aggregation should further enhance the power of this approach.

References

[1] A. Greenbaum, Iterative Methods for Solving Linear Systems, SIAM, Philadelphia, 1997.
[2] M. Benzi, Preconditioning techniques for large linear systems: A survey, J. Comput. Phys. 182 (2002) 418–477.
[3] K. Chen, Matrix Preconditioning Techniques and Applications, Cambridge University Press, Cambridge, England, 2005.
[4] 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.
[5] 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.
[6] 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.
[7] 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, March 2007.
[8] V.Y. Pan, X. Yan, Additive preconditioning, eigenspaces, and inverse iteration, Technical Report TR 2007004, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2007.
[9] 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, 2007.
[10] V.Y. Pan, B. Murphy, G. Qian, R.E. Rosholt, 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, 2007.
[11] V.Y. Pan, B. Murphy, R.E. Rosholt, M. Tabanjeh, The Schur aggregation for solving linear systems of equations, in: Marc Moreno Masa, Stephen Watt (Eds.), Proceedings of the third International Workshop on Symbolic–Numeric Computation, SNC 2007, July 2007, London, Ontario, Canada, ACM Press, New York, 2007, pp. 142–151.
[12] V.Y. Pan, X. Yan, Null space and eigenspace computations with additive preprocessing, in: Marc Moreno Masa, Stephen Watt (Eds.), Proceedings of the International Workshop on Symbolic–Numeric Computation, SNC 2007, July 2007, London, Ontario, Canada, ACM Press, New York, 2007, pp. 152–160.
[13] V.Y. Pan, B. Murphy, G. Qian, R.E. Rosholt, Error-free computations via floating-point operations, Technical Reports TR 2007010 and TR 2007013, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2007, Comput. Math. Appl. (in press)..
[14] G.H. Golub, C.F. Van Loan, Matrix Computations, 3rd ed., The Johns Hopkins University Press, Baltimore, Maryland, 1996.
[15] G.W. Stewart, Basic Decompositions, in: Matrix Algorithms, vol. I, SIAM, Philadelphia, 1998.
[16] X. Wang, Affect of small rank modification on the condition number of a matrix, Comput. Math. Appl. 54 (2007) 819–825.
[17] V.Y. Pan, Structured Matrices and Polynomials: Unified Superfast Algorithms, Birkhäuser, Springer, Boston, New York, 2001.
[18] J.J. Dongarra, I.S. Duff, D.C. Sorensen, H.A. van Der Vorst, Numerical Linear Algebra for High-Performance Computers, SIAM, Philadelphia, 1998.
[19] I.S. Duff, A.M. Erisman, J.K. Reid, Direct Methods for Sparse Matrices, Clarendon Press, Oxford, England, 1986.
[20] R.J. Lipton, D. Rose, R.E. Tarjan, Generalized nested dissection, SIAM J. Numer. Anal. 16 (2) (1979) 346–358.
[21] V.Y. Pan, J. Reif, Fast and efficient parallel solution of sparse linear systems, SIAM J. Comput. 22 (6) (1993) 1227–1250.
[22] R. Vandebril, M. Van Barel, G. Golub, N. Mastronardi, A bibliography on semiseparable matrices, Calcolo 42 (3–4) (2005) 249–270.
[23] C.L. Lawson, R.J. Hanson, Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, New Jersey, 1974. Reissued with a survey of recent developments by SIAM, Philadelphia, 1995.
[24] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, Philadelphia, 1996.
[25] G.W. Stewart, Eigensystems, first ed., in: Matrix Algorithms, vol II, SIAM, Philadelphia, 1998, 2001, second ed.
[26] H. Brönnimann, I.Z. Emiris, V.Y. Pan, S. Pion, Sign determination in residue number systems, Theoret. Comput. Sci. 210 (1) (1999) 173–197.
[27] V.Y. Pan, Y. Yu, Certification of numerical computation of the sign of the determinant of a matrix, Algorithmica 30 (2001) 708–724.
[28] J. Shevchuk, Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete Comput. Geom. 18 (1997) 305–363. Available at: www.cs.cmu.edu/quake/robust.html.
[29] J. Demmel, Y. Hida, Fast and accurate floating point summation with application to computational geometry, Numer. Algorithms 37 (1–4) (2004) 101–112.
[30] I.Z. Emiris, V.Y. Pan, Improved algorithms for computing determinants and resultants, J. Complexity 21 (1) (2005) 43–71.
[31] W. Gautschi, G. Inglese, Lower bounds for the condition number of vandermonde matrices, Numer. Math. 52 (1988) 241–250.
[32] E.E. Tyrtyshnikov, How bad are hankel matrices? Numer. Math. 67 (2) (1994) 261–269.
[33] V.Y. Pan, On computations with dense structured matrices, Math. Comput. 55 (191) (1990) 179–190.
[34] I. Gohberg, T. Kailath, V. Olshevsky, Fast gaussian elimination with partial pivoting for matrices with displacement structure, Math. Comput. 64 (1995) 1557–1576.
[35] G. Heinig, Inversion of generalized cauchy matrices and the other classes of structured matrices, in: Linear Algebra for Signal Processing, IMA Volume in Math. and Its Applications, vol. 69, 1995, pp. 95–114.
[36] N.J. Higham, Accuracy and Stability in Numerical Analysis, second ed., SIAM, Philadelphia, 2002.

[37] J. Demmel, Y. Hida, Accurate and efficient floating point summation, SIAM J. Sci. Comput. 25 (2003) 1214–1248.
[38] 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 Trans. Math. Software 28 (2002) 152–205. http://crd.lbl.gov/xiaoye/XBLAS/.
[39] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput. 26 (6) (2005) 1955–1988.
[40] U. Kulish, W.L. Miranker, Computer Arithmetic in Theory and Practice, Academic Press, New York, 1980.
[41] M. Malcolm, On accurate floating-point summation, Commun. ACM 4 (1971) 731–736.
[42] D.M. Priest, Algorithms for arbitrary precision floating point arithmetic, in: R. Kornerup, D.W. Matula (Eds.), Proc. 10th Symposium on Computer Arithmetic, IEEE Press, 1991, pp. 132–145.
[43] T.J. Dekker, A Floating-point Technique for Extending the Available Precision.
[44] V.Y. Pan, Can we utilize the cancellation of the most significant digits? Tech. Report TR 92 061, The International Computer Science Institute, Berkeley, California, 1992.
[45] I.Z. Emiris, V.Y. Pan, Y. Yu, Modular arithmetic for linear algebra computations in the real field, J. Symbolic Comput. 21 (1998) 1–17.
[46] IA-32 Intel Architecture Software Developers Manual, Volume 1: Basic Architecture, Intel Corporation, Mt. Prospect, Illinois, 2001. Order Number 245470.
[47] Agner Fog, How to optimize for the Pentium family of microprocessors. www.agner.org, 1996–2004. Last updated 2004-04-16.
[48] M.T. Chu, R.E. Funderlic, G.H. Golub, A rank-one reduction formula and its applications to matrix factorizations, SIAM Rev. 37 (4) (1995) 512–530.
[49] L. Hubert, J. Meulman, W. Heiser, Two purposes for matrix factorization: A historical appraisal, SIAM Rev. 42 (1) (2000) 68–82.
[50] G.H. Golub, Some modified matrix eigenvalue problems, SIAM Rev. 15 (1973) 318–334.
[51] V.Y. Pan, Univariate polynomial root-finding with lower precision and higher convergence rate, Technical Report TR 2002003, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2002.
[52] D.A. Bini, L. Gemignani, V.Y. Pan, Inverse power and Durand/Kerner iteration for univariate polynomial root-finding, Comput. Math. Appl. 47 (2–3) (2004) 447–459. Also Technical Report TR 2002020, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2002.
[53] D.A. Bini, L. Gemignani, V.Y. Pan, Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equation, Numer. Math. 3 (2005) 373–408. Also Technical Report 1470, Department of Math., University of Pisa, Pisa, Italy, July 2003.
[54] D.A. Bini, L. Gemignani, V.Y. Pan, Improved initialization of the accelerated and robust QR-like polynomial root-finding, Electron. Trans. Numer. Anal. 17 (2004) 195–205.
[55] V.Y. Pan, D. Ivolgin, B. Murphy, R.E. Rosholt, Y. Tang, X. Wang, X. Yan, Root-finding with Eigen-solving, in: Dongming Wang, Lihong Zhi (Eds.), Symbolic–Numerical Computation, Birkhäuser, Basel, Boston, 2007, pp. 219–245.
[56] W.L. Miranker, V.Y. Pan, Methods of aggregations, Linear Algebra Appl. 29 (1980) 231–257.
[57] V.Y. Pan, How can we speed up matrix multiplication? SIAM Rev. 26 (3) (1984) 393–415.
[58] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comput. 9 (3) (1990) 251–280.
[59] J. Laderman, V.Y. Pan, H.X. Sha, On practical algorithms for accelerated matrix multiplication, Linear Algebra Appl. 162–164 (1992) 557–588.
[60] I. Kaporin, A practical algorithm for faster matrix multiplication, Numer. Linear Algebra Appl. 6 (8) (1999) 687–700.
[61] I. Kaporin, The aggregation and cancellation techniques as a practical tool for faster matrix multiplication, Theoret. Comput. Sci. 315 (2–3) (2004) 469–510.
[62] D. Bondyfalat, B. Mourrain, V.Y. Pan, Computation of a specified root of a polynomial system of equations using eigenvectors, Linear Algebra Appl. 319 (2000) 193–209.
[63] B. Mourrain, V.Y. Pan, Multivariate polynomials, duality and structured matrices, J. Complexity 16 (1) (2000) 110–180.
[64] NAG Fortran Library Manual, Mark 13, vol. 1, 1988.
[65] V.Y. Pan, Solving a polynomial equation: Some history and recent progress, SIAM Rev. 39 (2) (1997) 187–220.
[66] F. Malek, R. Vaillancourt, Polynomial zerofinding iterative matrix algorithms, Comput. Math. Appl. 29 (1) (1995) 1–13.
[67] F. Malek, R. Vaillancourt, A composite polynomial zerofinding matrix algorithm, Comput. Math. Appl. 30 (2) (1995) 37–47.
[68] S. Fortune, An iterated eigenvalue algorithm for approximating roots of univariate polynomials, J. Symbolic Comput. 33 (5) (2002) 627–646. Proc. version in Proc. Intern. Symp. on Symbolic and Algebraic Computation, ISSAC'01, 121–128 ACM Press, New York, 2001.