2006

TR-2006006: Additive Preconditioning and Aggregation in Matrix Computations

Victor Y. Pan

Dmitriy Ivolgin

Brian Murphy

Rhys Eric Rosholt

Islam Taj-Eddin

See next page for additional authors

Authors

Victor Y. Pan, Dmitriy Ivolgin, Brian Murphy, Rhys Eric Rosholt, Islam Taj-Eddin, Yuqing Tang, and Xiaodong Yan

Additive Preconditioning and Aggregation in Matrix Computations *

Victor Y. Pan[1,a], Dmitriy Ivolgin[2],
Brian Murphy[1], Rhys Eric Rosholt[1],
Islam Taj-Eddin[2], Yuqing Tang[2], and Xiaodong Yan[2]

[1] Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
firstname.lastname@lehman.cuny.edu

[2] Ph.D. Programs in Computer Science
The City University of New York
New York, NY 10036 USA
firstnameinitial.lastname@gc.cuny.edu

[a] http://comet.lehman.cuny.edu/vpan/

June 11, 2006

Abstract

Multiplicative preconditioning is a popular tool for handling linear systems of equations provided the relevant information about the associated singular values is available. We propose using additive preconditioners, which are readily available for both general and structured ill conditioned input matrices and which preserve matrix structure. We introduce primal and dual additive preconditioning and combine it with two aggregation techniques. Our extensive analysis and numerical experiments show the efficiency of the resulting numerical algorithms for solving linear systems of equations and some other fundamental matrix computations. Our study provides some new insights into preconditioning, links it to various related subjects of matrix computations, and leads to some results of independent interest.

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

Key words: Matrix computations, Additive preconditioning, Aggregation.

1 Introduction

1.1 Additive preconditioning: why and how?

Multiplicative preconditioning is a popular technique for solving linear systems of equations $A\mathbf{y} = \mathbf{b}$. The original idea was to shift to some equivalent but better conditioned linear systems $BA\mathbf{y} = B\mathbf{b}$, $AC\mathbf{x} = \mathbf{b}$, or more generally $BAC\mathbf{x} = B\mathbf{b}$ where $\mathbf{y} = C\mathbf{x}$, so that a more accurate numerical solution $\mathbf{y}$ can be computed faster (see [1], the bibliography therein, and our next subsection). A more recent alternative is to compress the singular spectrum of a matrix into a smaller number of clusters of its singular values, but in this paper we pursue the original goal of *preconditioning*, that is, decreasing the condition number of an input matrix. Our approach to preconditioning, however, is different.

Effective multiplicative preconditioners for an input matrix are immediately defined by its Singular Value Decomposition (SVD), but the computation of the smallest singular values and the associated singular vectors of an ill conditioned matrix is costly. Furthermore, the SVD-based preconditioners can easily destroy the structure of a matrix.

As a remedy, we propose *additive preprocessing* $A \leftarrow C = A + UV^T$, i.e., we add a matrix UV^T (having a smaller rank and/or structured) to the input matrix A to decrease its condition number. Hereafter we use the abbreviations of *MPCs*, *APCs*, *M-* and *A-preconditioning* for multiplicative and additive preconditioners and for multiplicative and additive preconditioning, respectively. We avoid rounding errors in the matrices $A + UV^T$ by truncating all entries of the matrices U and V to short (that is, lower precision) numbers, which is another advantage versus M-preconditioning (cf. Section 4.1).

Effective APCs are quite readily available. For example (see the Acknowledgements), with a rank-one modification we can increase the absolute value of a small pivot entry in the Gaussian elimination and Cyclic Reduction algorithms without row/column interchange and thus without destroying matrix structure. Likewise, with small-rank modifications we improve conditioning of pivot blocks of small sizes in the block Gaussian elimination and block Cyclic Reduction.

According to our analysis and extensive experiments, one is likely to achieve a similar preconditioning effect for both general and structured nonsingular but ill conditioned input matrices of any size as long as a candidate APC is

a) random,

b) well conditioned,

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

d) a matrix of a sufficiently large rank (we specify how large in Section 4.4).

If all singular values of a matrix A are small except for a small number r_+ of them, then we are motivated to use dual APCs VU^T of larger ranks r_+ and the dual preconditioned matrices $C_- = A^{-1} + VU^T$. We avoid computing the

inverse matrix A^{-1} and define the matrices C_- implicitly by their inverses

$$(C_-)^{-1} = (A^{-1} + VU^T)^{-1} = A - AV(I_{r_+} + U^T AV)^{-1}U^T A. \tag{1.1}$$

Here and hereafter I_k denotes the $k \times k$ identity matrix. We call the latter equation the *dual SMW formula*, viewing it as a counterpart to the celebrated formula

$$(C - UV^H)^{-1} = C^{-1} + C^{-1}U(I_r - V^T C^{-1}U)^{-1}V^T C^{-1} \tag{1.2}$$

of Sherman, Morrison, and Woodbury (cf. [2, page 103], [3, page 155], and our Lemma 2.2), to which we refer as the *primal SMW formula* or just the *SMW formula*.

The matrix VU^T is likely to be an APC under requirements a) and b) and the following counterparts of the requirements c) and d),

e) the ratio $||A^{-1}||/||VU^T||$ is neither large nor small and

f) the rank of the APC VU is large enough (then again, we specify how large in Section 4.4).

For a structured input matrix we choose pseudo random APCs with the consistent structure (see Examples 4.1–4.6). This is in line with requirement a) because a very weak randomization is actually sufficient.

In sharp contrast, random M-preprocessing is helpless against ill conditioning because $\mathrm{cond}_2 A \le \prod_i \mathrm{cond}_2 F_i$ if $A = \prod_i F_i$. Here and hereafter $\mathrm{cond}_2 A$ denotes the condition number of a matrix A under the 2-norm of matrices.

We cover the generation of ACs and APCs in Part I of this paper and the improvement of APCs in Section 9.6.

1.2 Two impacts of preconditioning

1. *Preconditioning as a means of convergence acceleration*

Suppose the Conjugate Gradient algorithm is applied to a linear system $A\mathbf{y} = \mathbf{b}$ where A is a Hermitian matrix whose spectrum is not limited to a small number of clusters. Then the k iteration steps add the order of $k\sqrt{\mathrm{cond}_2 A}$ new correct bits per a variable (cf. [2, Theorem 10.2.6]), and so A-preconditioning enables convergence acceleration.

How much does this acceleration increase the computational cost per iteration? The basic operation of the algorithm is the multiplication of an input matrix by a vector, whose computational cost is little affected by small-rank modifications of the input as well as by its large-rank structured modifications.

The acceleration can be even stronger for the Wilkinson's iterative refinement (iterative improvement) algorithm in [2, Section 3.5.3], [3, Sections 3.3.4 and 3.4.5] because its k iterations add the order of $k\log(1/(||E||\,\mathrm{cond}_2 A))$ correct bits per a variable to the initial approximate solution to a linear system $A\mathbf{y} = \mathbf{b}$ provided the matrix $(A+E)^{-1}$ is available or an approximate solution $\tilde{\mathbf{z}} = (A+E)^{-1}\mathbf{v}$ to a linear system $A\mathbf{z} = \mathbf{v}$ is readily available.

A highly promising application area is the solution of a polynomial system of equations, which can be reduced to the solution of multi-level Toeplitz and

sparse linear systems [4] and [5]. One can multiply the coefficient matrix of such a system by a vector fast (and can hardly exploit this matrix structure otherwise), but the algorithms of the GMRES or Conjugate Gradient types are not much effective here because the matrices are typically ill conditioned and have singular values widely spread out. Structured APCs of larger ranks promise critical support.

Yet another example of the convergence acceleration with preconditioning is the approximation of the Moore–Penrose generalized inverse of a matrix A with Newton's iteration. $\log_2 \mathrm{cond}_2 A + \log_2 \log_2(1/\delta) + O(1)$ Newton's iteration steps are sufficient to yield the residual norm bound δ [6, Chapter 6], [7], [8] (on the earlier but quite advanced study, see [9]–[11]), and so here again we can expect dramatic convergence acceleration with preconditioning.

2. *Preconditioning for improving the accuracy of the output.*

With preconditioning we can obtain more accurate output by computing with the same precision. Such a power of preconditioning is well known for discretized solution of PDEs, eigen-solving, etc. There are still, however, some important areas where a potential help from preconditioning has not been claimed yet. Here are two examples.

The reduction of non-Hermitian and overdetermined linear systems of equations to normal linear systems is "the method of choice when the matrix is well conditioned" [12, page 118]. The users, however, are cautious about this reduction because it squares the condition number of the input matrix, which means the loss of the accuracy of the output. Here preconditioning can be a natural remedy.

Another example is numerical computation of the sign of the determinant of an ill conditioned matrix, which is critical for computing convex hulls and Voronoi diagrams and is required in many other fundamental geometric and algebraic computations (see our Section 8, the papers [13], [14]–[16], and the bibliography therein).

1.3 Utilizing APCs via aggregation

We propose two approaches that employ an APC UV^T to aggregate the original ill conditioned linear system into well conditioned computational problems of the same or smaller sizes. In some cases the aggregation can be numerically unstable, but we confine the instability to the summation stages and overcome it with the *floating-point summation subroutines* from Section 12.

We call these approaches the *Schur Aggregation* and the *Null Aggregation* and view them as the descendants of the *Aggregation Processes* in [17], which in the eighties evolved into the *Algebraic Multigrid*. We validate them with some nontrivial analysis and extensive experiments. Both approaches have primal and dual versions. Next we outline them for those readers who prefer not to wait for reading all details in Parts II and III.

The Schur Aggregation. The SMW formula reduces a linear system $A\mathbf{y} = \mathbf{b}$ to linear systems with the coefficient matrices $C = A + UV^T$ and

its Schur complement (Gauss transform) $G = I_r - V^T C^{-1} U$ in the matrix $\begin{pmatrix} C & U \\ V^T & I_r \end{pmatrix}$.

Let $\sigma_j(M)$ denote the jth largest singular value of a matrix M and let A be an $n \times n$ matrix. Assume an APC UV^T satisfying the requierments a)–d) for the APCs. Then based on our study in Section 5 and tests in Section 6, we should expect that all singular values of the matrix C lie in the line segment $[c_1\sigma_1(A), c_2\sigma_{n-r}(A)]$ for two moderate constants c_1 and c_2. Now suppose the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ is not large, that is, the matrix C is well conditioned. Then in virtue of our Theorem 7.4 all singular values of the matrix G lie in the line segment $[c^2\sigma_{n-r+1}(A) + c, c^2\sigma_n(A) - c]$ for a moderate constant c. Therefore, this matrix is well conditioned unless the ratio $\sigma_{n-r+1}(A)/\sigma_n(A)$ is large.

We use the Null Aggregation to optimize the ranks of the APCs.

The computation of the matrix G can be numerically unstable, but we overcome this problem by applying a variant of the Wilkinson's iterative refinement and the floating-point summation subroutines from Section 12.

To support dual A-preconditioning, we have a little harder task of estimating the norm $||A^{-1}||$ (versus the norm $||A||$ for primal APCs), but with the dual SMW formulae we avoid using iterative refinement. We just multiply the matrix A by vectors to reduce the original linear systems $A\mathbf{y} = \mathbf{b}$ to linear systems with the matrices $(C_-)^{-1}$ and $I + U^T AV$. They are expected to be well conditioned for random and pseudo random APCs of a larger rank, but we still need the floating-point summation subroutines at the stage of computing the matrix $I + U^T AV$.

The Null Aggregation. We can employ primal APCs UV^T without involving the Schur complements $G = I_r - V^T C^{-1} U$. Suppose A is a singular matrix. If $\operatorname{nul} A = \operatorname{rank}(UV^T)$, then the columns of the matrices $C^{-T}V = (V^T C^{-1})^T$ and $C^{-1}U$ span the left and right null spaces of the matrix A, respectively. We can use the binary search to compute the nullity $\operatorname{nul} A$ as the minimum integer r for which the matrix $C = A + UV^T$ is nonsingular or as the maximum integer r for which $AC^{-1}U = 0$ as well as $V^T C^{-1} A = 0$.

For a nonsingular ill conditioned matrix A, normalized so that $||A||_2 = 1$, we can perform similar computations numerically and output its *numerical nullity* $\operatorname{nnul} A$, that is, the number of its small singular values, which is equal to the minimum rank of an APC UV^T such that the matrix $C = A + UV^T$ can be well conditioned. In this case the matrix C is expected to be well conditioned for the APCs UV^T satisfying the requirements a)–d).

We have, however, the following more efficient alternative. For an APC UV^T of rank $\operatorname{nnul} A$, the columns of the matrices $C^{-1}U$ and $C^{-T}V$ form two approximate bases for the two *trailing singular spaces*, associated with the smallest singular values of the matrix A. (For a structured matrix A we can choose these approximate matrix bases structured, even though the singular spaces may have no exact structured matrix bases.) Based on these observations we can readily compute an APC of rank $\operatorname{nnul} A$ in two stages.

First choose a random or pseudo random matrix UV^T of a larger rank (e.g.,

of $2\,\mathrm{nnul}\,A$). Such a matrix is likely to define a well conditioned matrix C. Verify that this is the case by applying the known condition estimators [2, Section 3.5.4], [3, Sections 5.3] and then compute the matrix pairs

$$(U \leftarrow Q(C^{-1}U),\ \ V \leftarrow Q(C^{-T}V)) \tag{1.3}$$

where $Q(M)$ denotes the Q-factor in the QR factorization of a matrix M (cf. Section 9.6). The resulting APC UV^T has the rank $\mathrm{nnul}\,A$, whereas the matrix $C = A + UV^T$ remains well conditioned. The efficiency of this recipe has been confirmed by our extensive tests (cf. Table 6.2) as well as the tests in [18].

Similarly we can begin with a random or pseudo random dual matrix VU^T of a rank $r_+ > \mathrm{nnul}(A^{-1})$, e.g., $r_+ = 2\,\mathrm{nnul}(A^{-1})$. Then again such a dual matrix is likely to define a well conditioned matrix $(C_-)^{-1}$ in equation (1.1). Having verified that this is the case, we can apply the transform

$$(V \leftarrow Q((C_-)^{-1}V),\ \ U \leftarrow Q((C_-)^{-T}U)) \tag{1.4}$$

to yield an effective dual APC of the optimal rank $\mathrm{nnul}(A^{-1})$.

Our null space computations are also readily extended to supporting the inverse iteration for the algebraic eigenproblem (see Section 11) and to the solution of a linear system of equations $A\mathbf{y} = \mathbf{b}$, essentially equivalent to computing a null vector for the matrix $(-\mathbf{b}, A)$ (see Section 10.1). In this approach to linear solving as well as to the null space, eigenspace, and singular space computations, A-preconditioning does not involve the floating-point summation subroutines from Section 12.

They, however, are used again in the alternative applications of the Null Aggregation to solving linear systems in Section 10.2. In particular we can represent the solution $\mathbf{y}$ to a linear system $A\mathbf{y} = \mathbf{b}$ as $C^{-1}(\mathbf{b} + U\mathbf{x})$. Here $\mathbf{x}$ is a vector satisfying the linear system $AC^{-1}(\mathbf{b} + U\mathbf{x}) = \mathbf{b}$, and we choose a primal APC UV^T such that the matrix $C = A + UV^T$ is nonsingular and well conditioned. The norm $||AC^{-1}U||_2$ tends to be small in our applications. Thus to compute the vector $\mathbf{x}$, we should first compute the matrix $AC^{-1}U$ with a high precision. Here again our variant of iterative refinement and the floating-point summation subroutines come to rescue.

Alternatively, we can rely on a dual APC VU^T and obtain that $\mathbf{y} = \mathbf{z} - VU^T\mathbf{b}$ wherever $(C_-)^{-1}\mathbf{z} = \mathbf{b}$ for the matrix $C_- = A^{-1} + VU^T$ (cf. (1.1)). To compute the matrix $(C_-)^{-1}$ we only need to solve linear systems with the $(\nu_+) \times (\nu_+)$ matrix $I + U^TAV$ where $\nu_+ = \mathrm{nnul}(A)^{-1}$, apart from performing some matrix-by-vector multiplications and from application of the floating-point summation subroutines to computing the matrix $I + U^TAV$. We refer the reader to the end of Section 10.2 on some further simplifications.

1.4 The contents and the organization of the paper

In this paper we cover primal and dual A-preprocessing in some detail, including its basic properties and the benefits of using it. We link it to Aggregation Processes, iterative refinement, M-preconditioning, factorization, and the SVD

and null space computations. We generate random and structured pseudo random APCs and study their affect on the conditioning of the input matrix, both theoretically and experimentally. Finally we propose various techniques for the application of the APCs to solving linear systems of equations and other fundamental matrix computations. Our initial experiments and theoretical analysis confirm the power of our approach.

Our study reveals various links in this area, e.g., among matrix aggregation (cf. [17], our Section 7.2 and Remark 9.2), generation of APCs and MPCs, computing null space bases, and approximation of trailing singular spaces of ill conditioned matrices. This should lead to new insights and new techniques of algorithm design. Further extensions are the subject of our current and future study (see Part IV).

Some by-products and auxiliary tools of our study, such as the dual SMW formula and the computation of a structured approximate matrix basis for the trailing singular space of an ill conditioned structured matrix (cf. Remark 9.4), should have independent interest and applications.

We present our results in the general case of rectangular input matrices and for completeness include the straightforward SVD-based MPCs and APCs as a natural counterpart to our SVD-free APCs.

We organize our paper as follows. We begin with some definitions and preliminary results in Section 2, devise SVD-based MPCs and APCs in Section 3 and SVD-free APCs in Section 4, and study the effect of SVD-free A-preprocessing on conditioning, at first theoretically in Section 5 and then experimentally in Section 6. Sections 3–6 form Part I of our paper.

We propose and analyze the Schur Aggregation for the transition from APCs to factorization and summarize the resulting recipes for the solution of linear systems of equations in Section 7. In Section 8 we specify simplifications of these recipes where we seek the sign and/or the value of the determinant of a matrix. Sections 7 and 8 form Part II of our paper.

We combine A-preprocessing and the Null Aggregation to compute the null vectors and null space bases for a singular matrix in Section 9, and we show further applications to numerical solution of linear systems of equations in Section 10 and the inverse iteration for the algebraic eigenproblem in Section 11.

In Section 12 we first point out some bibliography on the advanced algorithms for floating-point computation of sums and products and then for the sake of completeness of our exposition, specify some simpler basic algorithms for the same computation.

Sections 9–12 form Part III of our paper.

We present a concluding summary and a brief discussion in Part IV.

Our numerical tests have been designed by the first author and performed by the other authors, mostly by D. Ivolgin, X. Yan, and Y. Tang. Otherwise the paper (with all typos and other errors) is due to the first author.

1.5 Selective reading

After Section 4 the reader can go to Sections 5 and 6, Part II, or Part III. Each selective reading requires only a part of the definitions and results in Sections 2–4. Only Sections 2, 4, and 9 are prerequisits for reading Section 10.2.

Finally, at least on first reading one can skip Sections 3 (except maybe for subsection 3.1 and equations (3.10) and (3.11)), 7.3, and 8.3. We included these sections on SVD-based preprocessing for the sake of completeness of our presentation.

1.6 The preceding study

Small-rank modification is a known tool for decreasing the rank of a matrix [19], [20], fixing its small-rank deviations from the Hermitian, positive definite, and displacement structures, and supporting the divide-and-conquer eigen-solvers [2], [21], [22], but other than that has been rarely used in matrix computations. The discussions that followed the presentations of our work 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 relevant citations, namely, [23], a paper by Paige, Styan, and Wachter in the Journal of Statistical Comp. Simulation, 1975, and some old works by P. Lancaster. These sporadic touches to A-preconditioning, although important, were still rudimentary versus our present work. We are aware of no earlier use of the nomenclature of A-preconditioning or primal and dual APCs as well as of no attempts of devising and employing random and structured pseudo random primal and dual APCs, improving APCs based on the Null Aggregation, studying APCs systematically, relating them to aggregation, or applying them to numerical approximation of the bases for the trailing singular spaces of ill conditioned matrices.

The first author arrived at A-preconditioning while applying the inverse iteration for the rank-structured algebraic eigenproblem to polynomial root-finding. (This novel approach to polynomial root-finding was proposed in [24]. Its QR-based variation was proposed in [25], [26] and prompted a stream of further publications by many researchers.) Then this author observed applications of APCs to null space computations, constructed random and structured pseudo random ACs and APCs, estimated their affect on conditioning, defined the classes of primal and dual APCs, and studied the links of APCs to null space computation and aggregation. Finally he worked out the Schur Aggregation.

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 visit to their Institutes in 2005/06. Dr. Xinmao Wang responded to the present work with the paper [18] and initialized our work on numerical tests. The participants of the cited Conferences in

Moscow and Santander (particularly J. W. Demmel, G. H. Golub, V. B. Khazanov, L. Reichel, M. Van Barel, and a participant of FoCM'2005 who proposed the substitution of APCs for pivoting) made a number of valuable comments.

2 Basic definitions and preliminaries

Most of our basic definitions reproduce or slightly modify the customary definitions in [2], [3], [12], [22], [27], [28].

2.1 Vectors and matrices

For any set Δ in a ring $\mathbb{R}$ and a pair of positive integers m and n, $A = (a_{i,j})_{i=1,j=1}^{m,n} \in \Delta^{m\times n}$ is an $m \times n$ matrix with entries in this set and $\mathbf{v} = (v_i)_{i=1}^{n} \in \Delta^{n\times 1}$ is a column vector of dimension n with coordinates in this set. In this paper we can assume that the rings $\mathbb{R}$ are the field $\mathbb{C}$ of complex numbers or its subfield of real numbers.

A^T and $\mathbf{v}^T$ are the transposes of a matrix A and a vector $\mathbf{v}$. A^H and $\mathbf{v}^H$ are their Hermitian (that is, complex conjugate) transposes where $\mathbb{R} = \mathbb{C}$, so that $A^H = A^T$ and $\mathbf{v}^H = \mathbf{v}^T$ where A and $\mathbf{v}$ are real.

I_k is the $k \times k$ identity matrix, I is I_k for an unspecified k. $\mathbf{e}_i$ is its i-th column vector. 0 is the matrix of a proper size filled with zeros. $0_{g,h}$ is the $g \times h$ matrix 0. $0_k = 0_{k,k}$ is the $k \times k$ matrix 0.

A is a unitary matrix if $A^H A = I$.

QRP factorization of a (generally rectangular) matrix A is the triple (Q, R, P) of matrices defining its QR factorization with column pivoting [2, Sections 5.4.1 and 5.4.2], [3, Algorithms 4.1.2 and 5.2.1]. The matrices Q, R, and P are the respective Q-, R-, and P-factors of the matrix A. We assume that the Q-factor has the same size as the input matrix A, except that we delete the Q-columns corresponding to the zero rows of the R-factor.

$\mathrm{diag}(A, B)$ is the block diagonal matrix with the diagonal blocks A and B. (A, B) is the 1×2 block matrix with the blocks A and B.

$M_{C,r}$ is the number of arithmetic operations sufficient to multiply a fixed $m \times n$ matrix C by an $n \times r$ matrix, $M_{C,r} \le 2mnr - mr$.

2.2 Random matrices

Random sampling of elements from a finite set Δ is their selection from the set Δ at random, independently of each other, and under the uniform probability distribution on Δ. A matrix is *random* if its entries are randomly sampled (from a fixed finite set Δ).

A *random unitary* matrix is the Q-factor in the QRP factorization of a random matrix, which is the Q-factor in its QR factorization if this is a matrix of full rank.

Lemma 2.1. *[29] (cf. also [30], [31]). For a finite set Δ of cardinality $|\Delta|$, let a polynomial in m variables have total degree d and not vanish identically on*

the set Δ^m and let the values of its variables be randomly sampled from the set Δ. Then the polynomial vanishes with a probability of at most $d/|\Delta|$.

2.3 Spans, ranges, and null spaces

The linear space generated by a set of vectors is their span.

An $m \times n$ matrix A has full rank if its rank equals $\min\{m, n\}$. Otherwise the matrix is rank deficient. $\operatorname{rank} A$ is its rank, $\operatorname{range} A$ is its range, that is, the span of its column vectors, and for $m = n$, $\det A$ is its determinant. Such a matrix A is a *matrix basis* for its range if its column set is linearly independent.

$N(A) = RN(A) = \{\mathbf{x} : A\mathbf{x} = \mathbf{0}\}$ is the (right) null space of a matrix A. $LN(A) = \{\mathbf{x} : \mathbf{x}^T A = \mathbf{0}^T\} = N(A^H)$ is its left null space. A *null basis* and a *null matrix basis* for a matrix is a basis and a matrix basis for its null space, respectively. (Right) null vectors, left null vectors, (right) null bases, left null bases, (right) matrix bases, and left matrix bases of a matrix A are the vectors, bases, and matrix bases in the null spaces $N(A)$ and $LN(A)$ of the matrix A, respectively.

For an $m \times n$ matrix A of rank ρ, its left nullity $\operatorname{lnul} A = n - \rho$ and its right nullity $\operatorname{rnul} A = m - \rho$ are the dimensions of its left and right null spaces $N(A)$ and $LN(A)$, respectively. Its *nullity* $\operatorname{nul} A = \min\{m, n\} - \rho$ is the minimum of $\operatorname{lnul} A$ and $\operatorname{rnul} A$.

2.4 Normalization, numerical rank, numerical nullity, and rounding

$||A||_2$ and $||A||_F = \sqrt{\operatorname{trace}(A^H A)}$ are the 2-norm and the Frobenius norm of a matrix A, respectively.

A matrix or a vector is *normalized* if its 2-norm is equal to one.

For a fixed positive ϵ, the ϵ-rank and the ϵ-nullity of an $n \times n$ matrix A are the maximum rank and the maximum nullity, respectively, over all matrices $A + E$ in the ϵ-neibourhood $\{A + E : ||E||_2 \le \epsilon\}$ of the matrix A.

Numerical rank, $\operatorname{nrank} A$, and *numerical nullity*, $\operatorname{nnul} A$, are the ϵ-rank and the ϵ-nullity, respectively, for an unspecified small positive ϵ. (Left and right numerical and ϵ-nullities naturally arise as well, but we deal just with numerical and ϵ-nullities.)

Apart from its use in these definitions, ϵ denotes the *unit roundoff*, also called *machine epsilon*.

2.5 The SVDs

The *compact Singular Value Decomposition* of an $m \times n$ matrix A of a rank ρ (also called the *Compact SVD* of this matrix) 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 $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_\rho > 0$. We have $||A||_2 = \sigma_1$ and $||A||_F^2 = \sum_{j=1}^{\rho} \sigma_j^2$.

Write $l = \operatorname{lnul} A = m - \rho$, $r = \operatorname{rnul} A = n - \rho$ and, for a pair $S^{(\mathrm{nul})} = (\mathbf{s}_j)_{j=\rho+1}^{m}$ and $T^{(\mathrm{nul})} = (\mathbf{t}_j)_{j=\rho+1}^{n}$ of left and right unitary null matrix bases for the matrix A, define the square unitary matrices $S = (S^{(\rho)}, S^{(\mathrm{nul})}) = (\mathbf{s}_j)_{j=1}^{m}$ and $T = (T^{(\rho)}, T^{(\mathrm{nul})}) = (\mathbf{t}_j)_{j=1}^{n}$ and the $m \times n$ matrix $\Sigma = \operatorname{diag}(\Sigma^{(\rho)}, 0_{l,r})$. The equation $A = S\Sigma T^H$ is the *Singular Value Decomposition* of the matrix A, also called its *SVD* and *full SVD*.

Hereafter we write $\sigma_j = 0$ for $j > \rho$ and $\sigma_j = +\infty$ for $j < 1$. The scalars σ_j for $j \geq 1$ are the *singular values* of the matrix A, and the vectors $\mathbf{s}_j$ for $j = 1, \dots, m$ and $\mathbf{t}_j$ for $j = 1, \dots, n$ are the associated left and right *singular vectors*, respectively, so that the null vectors are the singular vectors associated with the singular value zero.

We have $A\mathbf{t}_j = \sigma_j \mathbf{s}_j$ and $\mathbf{s}_j^H A = \sigma_j \mathbf{t}_j^H$ for all j, $A^H = T\Sigma^T S^H$, $A^H A = T\Sigma^T \Sigma T^H$, and $AA^H = S\Sigma\Sigma^T S^H$.

2.6 Generalized inverses and conditioning of a matrix. (g, h) matrices

The *Moore-Penrose generalized inverse* of an $m \times n$ matrix A of a rank ρ (also called the *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 [2]), so that $A^- = A^{-1}$ if $m = n = \rho$,

$$A^- = (A^H A)^{-1} A^H \text{ if } m \geq n = \rho, \tag{2.1}$$

$$A^- = A^H (AA^H)^{-1} \text{ if } m = \rho \leq n. \tag{2.2}$$

Fact 2.1. *The matrices A^H and A^- share their singular spaces, whereas*
$\sigma_j(A^H) = \sigma_j(A) = 1/\sigma_j(A^-)$ *for* $h = 1, \dots, \operatorname{rank} A$,
$\sigma_j(A^H) = \sigma_j(A) = \sigma_j(A^-) = 0$ *for* $j > \operatorname{rank} A$.

By writing $0^- = 0$ and $a^- = 1/a$ for a nonzero scalar a, we combine the latter equations as follows, $\sigma_j(A^H) = \sigma_j(A) = \sigma_j^-(A^-)$ for all j.

$\operatorname{cond}_2 A = \sigma_1(A)/\sigma_\rho(A) = ||A||_2 ||A^-||_2$ is the condition number of a matrix A of a rank ρ (under the 2-norm). Effective condition estimators can be found in [2, Section 3.5.4] and [3, Section 5.3].

We write $n >> d$ where the ratio n/d is large.

A matrix A of a rank ρ is *ill conditioned* if $\sigma_1 >> \sigma_\rho$ and is *well conditioned* otherwise. For two nonnegative integers g and h, a matrix A of a rank $\rho > g + h$ is (g, h) *well conditioned* if the ratio $\sigma_{g+1}/\sigma_{\rho-h}$ is not large, that is, if all large jumps in the positive singular values, including the jumps on the boundaries, are confined to the g-head and h-tail of its SVD.

A matrix is well conditioned if and only if it is $(0, 0)$ well conditioned. Otherwise, that is, if the jumps indeed occur, so that the ratios σ_1/σ_{g+1} and/or $\sigma_{\rho-h}/\sigma_\rho$ are large, then we say that the SVD of the matrix A has *ill* g-head and/or *ill* h-tail, respectively, and we can cure the illness with SVD-based MPCs.

Our treatment is less costly in the case of smaller g and h. (A matrix A has ill h-tail if $1 \le \operatorname{nnul} A \le h$ and has ill g-head if $1 \le \operatorname{nnul} A^- \le g$.)

We call a matrix A a (g, h) *matrix* if it is (g, h) well conditioned and if its SVD has ill g-head and ill h-tail. Clearly, this matrix is also a $(g + i, h + j)$ matrix for any nonnegative i and j such that $g + i + h + j < \rho$.

2.7 Schur complements and the Sherman–Morrison–Woodbury formula

For a 2×2 block matrix $F = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$, the matrix $S_{22} = A_{22} - A_{21}A_{11}^{-}A_{12}$ (respectively, $S_{11} = A_{11} - A_{12}A_{22}^{-}A_{21}$) is the *Schur complement* of the northeastern block A_{11} (respectively, the southeastern block A_{22}) in the matrix F provided $A_{11}^{-}A_{11} = I$ and/or $A_{11}A_{11}^{-} = I$ (respectively, $A_{22}^{-}A_{22} = I$ and/or $A_{22}A_{22}^{-} = I$) [2, page 103], [3, page 155]. We immediately verify the following lemma.

Lemma 2.2. *If the above matrices F, A_{11}, and A_{22} are nonsingular, then $F^{-1} = \begin{pmatrix} S_{11}^{-1} & X \\ Y & S_{22}^{-1} \end{pmatrix}$ for some matrices X and Y.*

Theorem 2.1. *[2, page 50]. 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 $S = I_r - V^H C^{-1} U$ are the respective Schur complements of the blocks I_r and C in the matrix $W = \begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix}$ such that*

$$\det W = \det A = (\det C) \det S. \tag{2.3}$$

If the matrix A is nonsingular, then so is the matrix S, and we have the Sherman–Morrison–Woodbury formula $(C - UV^H)^{-1} = C^{-1} + C^{-1}US^{-1}V^HC^{-1}$.

Proof. Begin with the factorizations

$$\begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix} = \begin{pmatrix} I_n & U \\ 0 & I_r \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & I_r \end{pmatrix} \begin{pmatrix} I_n & 0 \\ V^H & I_r \end{pmatrix} =$$

$$\begin{pmatrix} I_n & 0 \\ V^HC^{-1} & I_r \end{pmatrix} \begin{pmatrix} C & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I_n & C^{-1}U \\ 0 & I_r \end{pmatrix},$$

which implies the claimed equations for the determinants.

Invert this block factorization to obtain that

$$\begin{pmatrix} A^{-1} & X \\ Y & Z \end{pmatrix} = \begin{pmatrix} I_n & 0 \\ -V^H & I_r \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & I_r \end{pmatrix} \begin{pmatrix} I_n & -U \\ 0 & I_r \end{pmatrix} =$$

$$\begin{pmatrix} I_n & -C^{-1}U \\ 0 & I_r \end{pmatrix} \begin{pmatrix} C^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I_n & 0 \\ -V^HC^{-1} & I_r \end{pmatrix} =$$

$$\begin{pmatrix} C^{-1} - C^{-1}US^{-1}V^HC^{-1} & X \\ Y & Z \end{pmatrix}$$

for some matrices X, Y, and Z. □

Remark 2.1. *We can also deduce equation (2.3) from the two equations* $\det A = (\det C)\det(I_n - C^{-1}UV^H)$ *(implied by the equation* $A = C(I - C^{-1}UV^H)$*) and* $\det(I_r - XY) = \det(I_n - YX)$ *for* $n \times r$ *matrices* X^H *and* Y *[32, Exercise 1.14], which we would apply for* $X = V^H$ *and* $Y = C^{-1}U$*. For* $r = 1$*,* $U = \mathbf{u}$*, and* $V = \mathbf{v}$*, equation (2.3) turns into the classical equation* $\det A = (\det C)\det(I_n - C^{-1}\mathbf{u}\mathbf{v}^H) = (1 - \mathbf{v}^HC^{-1}\mathbf{u})\det C$ *(cf. [21] and [32]).*

2.8 Some abbreviations

"ops" stands for "arithmetic operations", "CG" for "Conjugate Gradient", "IPI" for "inverse power iteration", "IR–RI" for "inverse Rayleigh–Ritz iteration", "SMW" for "Sherman–Morrison–Woodbury", "PPs" for "preprocessors", "PCs" for "preconditioners", "A" for "additive", "ACs" for "additive complements", "M" for "multiplicative", and "MCs" for "multiplicative compressors". We also combine the abbreviations, so that "MPPs" stands for "multiplicative preprocessors", "APPs" for "additive preprocessors", "MPCs" for "multiplicative preconditioners", and "APCs" for "additive preconditioners".

2.9 MPPs and APPs (definitions)

Definition 2.1. Multiplicative preprocessors. *The pair of nonsingular matrices* M *and* N *is an* MPC *for a matrix* A *if* $\operatorname{cond}_2 A << \operatorname{cond}_2(MAN)$*. Such a pair of nonsingular matrices is an* MC *for a matrix* A *if this matrix is rank deficient, whereas the matrix* MAN *turns into a full rank matrix after the deletion of its zero rows and columns. If one of the matrices in the pair is the identity matrix, then the other matrix is also called an MPC or MC, respectively. MPPs cover both MPCs and MCs.*

Definition 2.2. Additive preprocessors. *For a pair of matrices* U *of size* $m \times r$ *and* V *of size* $n \times r$*, both having full rank* $r > 0$*, the matrix* UV^H *(of rank* r*) is an* APP (of rank r) *for any* $m \times n$ *matrix* A*, and the transition* $A \leftarrow C = A + UV^H$ *is* A-preprocessing of rank r *for the matrix* A*. An APP* UV^H *for a matrix* A *is an* APC *if* $\operatorname{cond}_2 A >> \operatorname{cond}_2 C$*, is* unitary *if the matrices* U *and* V *are unitary, and is an* AC *if the matrix* A *is rank deficient, whereas the matrix* $C = A + UV^H$ *has full rank. An AC of rank* r *is* balanced *if* $r = \operatorname{nul} A$*.*

PART I. SVD-BASED AND SVD-FREE PRECONDITIONING

We cover the SVD-based MCs, MPCs, ACs, and APCs in Section 3. In Section 4 we first comment on the benefits and limitations of using them and then shift to

our main subject of the SVD-free ACs and APCs. As we pointed out in Section 1.5, the reader, at least on first reading, can skip Section 3, except maybe for subsection 3.1 and equations (3.10) and (3.11).

3 SVD-based MPPs and APPs

3.1 Linear systems of equations, matrix factorization and MPPs

To solve a matrix equation $AY = B$, we can employ factorizations $F = MAN$ or $A = \widehat{M}\widehat{F}\widehat{N}$ where the matrices M, N, $\widehat{M}$, and $\widehat{N}$ are nonsingular (cf. Section 1.1). In this case $Y = NX$ where $FX = MB$ and $\widehat{N}Y = \widehat{X}$ where $\widehat{M}\widehat{Z} = B$, $\widehat{F}\widehat{X} = \widehat{Z}$. We are interested in MPPs M, N, $\widehat{M}$, and $\widehat{N}$ with which the transition to the new equations simplifies the solution. If needed, we can recursively factorize the matrices F, $\widehat{M}$, $\widehat{F}$, and/or $\widehat{N}$. Various other matrix computations also boil down to factorizations (see [2], [3], [22] and our Section 8).

For numerical computations the SVD is the most reliable although expensive factorization. In some cases it is sufficient to use just the head and/or tail of the SVD, that is, its parts associated with smaller numbers of the largest and/or smallest singular values of a matrix. Next we assume that such a less expensive partial information about the SVD is available and seek factorizations $F = MAN$ and/or $A = \widehat{M}\widehat{F}\widehat{N}$ such that the matrices F, $\widehat{M}$, $\widehat{F}$, and $\widehat{N}$

a) become full rank matrices after the deletion of their zero rows and/or columns and/or

b) are better conditioned than the matrix A.

The MPPs M, N, $\widehat{M}$, and $\widehat{N}$ are MCs in case a) and/or MPCs in case b).

We begin with formally defining the relevant parts of the SVDs.

3.2 The g-head, h-tails, extended h-tails, and (g,h)-SVDs

The *g-head*, (g,h)-*residue*, *h-tail*, and *extended h-tail* of the SVD are the triples $(S^{(g)}, \Sigma^{(g)}, T^{(g)})$, $(S_{g,h}, \Sigma_{g,h}, T_{g,h})$, (S_h, Σ_h, T_h), and $(S_h^{(\mathrm{ext})}, \Sigma_h^{(\mathrm{ext})}, T_h^{(\mathrm{ext})})$, respectively, where g and h are two nonnegative integers, $g+h \le \rho$, $S^{(g)} = (\mathbf{s}_j)_{j=1}^{g}$, $S_{g,h} = (\mathbf{s}_j)_{j=g+1}^{\rho-h}$, $S_h = (\mathbf{s}_j)_{j=\rho-h+1}^{\rho}$, $S_h^{(\mathrm{ext})} = (\mathbf{s}_j)_{j=\rho-h+1}^{m}$, $\Sigma^{(g)} = \mathrm{diag}(\sigma_j)_{j=1}^{g}$, $\Sigma_{g,h} = \mathrm{diag}(\sigma_j)_{j=g+1}^{\rho-h}$, $\Sigma_h = \mathrm{diag}(\sigma_j)_{j=\rho-h+1}^{\rho}$, $\Sigma_h^{(\mathrm{ext})} = \mathrm{diag}(\Sigma_h, 0_{l,r})$, $T^{(g)} = (\mathbf{t}_j)_{j=1}^{g}$, $T_{g,h} = (\mathbf{t}_j)_{j=g+1}^{\rho-h}$, $T_h = (\mathbf{t}_j)_{j=\rho-h+1}^{\rho}$, $T_h^{(\mathrm{ext})} = (\mathbf{t}_j)_{j=\rho-h+1}^{n}$.

The spaces generated by the columns of the matrices $S^{(g)}$, $T^{(g)}$, $S_h^{(\mathrm{ext})}$, and $T_h^{(\mathrm{ext})}$ are the *left* and *right g-leading* and *h-trailing singular spaces* of matrix A, respectively.

Write $A^{(g)} = \sum_{j=1}^{g} \sigma_j \mathbf{s}_j \mathbf{t}_j^H = S^{(g)}\Sigma^{(g)}T^{(g)H}$, $A_{g,h} = \sum_{j=g+1}^{\rho-h} \sigma_j \mathbf{s}_j \mathbf{t}_j^H = S_{g,h}\Sigma_{g,h}T_{g,h}^H$, and $A_h = \sum_{j=\rho-h+1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H = S_h\Sigma_h T_h^H$ and observe that $A = A^{(g)} + A_{g,h} + A_h$, $S = (S^{(g)}, S_{g,h}, S_h, S^{(\mathrm{nul})})$, $\Sigma = \mathrm{diag}(\Sigma^{(g)}, \Sigma_{g,h}, \Sigma_h, 0_{l,r})$, and $T = (T^{(g)}, T_{g,h}, T_h, T^{(\mathrm{nul})})$.

Now, for a pair of unitary matrices $S_{g,h}^{(ORT)}$ of size $m\times(\rho-g-h)$ and $T_{g,h}^{(ORT)}$ of size $n\times(\rho-g-h)$ such that

$$S_{g,h}^{(ORT)H}(S^{(g)},S_h,S^{(\mathrm{nul})})=0, T_{g,h}^{(ORT)H}(T^{(g)},T_h,T^{(\mathrm{nul})})=0, \tag{3.1}$$

write $l=\mathrm{lnul}\,A$ and $r=\mathrm{rnul}\,A$, define the pair of unitary matrices

$$\tilde{S}_{g,h}=(S^{(g)},S_{g,h}^{(ORT)},S_h,S^{(\mathrm{nul})}),\quad \tilde{T}_{g,h}=(T_g,T_{g,h}^{(ORT)},T_h,T^{(\mathrm{nul})}),$$

and obtain the factorizations

$$A=\tilde{S}_{g,h}^H\begin{pmatrix}\Sigma^{(g)}T^{(g)H}\\ S_{g,h}^{(ORT)H}A\\ \Sigma_hT_h^H\\ 0_{l,n}\end{pmatrix}, \tag{3.2}$$

$$A=(S^{(g)}\Sigma^{(g)},AT_{g,h}^{(ORT)},S_h\Sigma_h,0_{m,r})\tilde{T}_{g,h}, \tag{3.3}$$

$$A=\tilde{S}_{g,h}\,\mathrm{diag}(\Sigma^{(g)},S_{g,h}^{(ORT)H}AT_{g,h}^{(ORT)},\Sigma_h,0_{l,r})\tilde{T}_{g,h}, \tag{3.4}$$

which we call the *left* (g,h)-*SVD*, the *right* (g,h)-SVD, and the (g,h)-SVD or the *full* (g,h)-SVD, respectively.

3.3 SVD-based MPPs

Given the SVD $A=S\Sigma T^H$, we obtain the MCs $M=S^H$, $N=T$, and $(M,N)=(S^H,T)$ where the matrix A is rank deficient. Likewise, we obtain the MPCs $M=\Sigma^-S^H$, $N=T\Sigma^-$, $(M,N)=(\Sigma^-S^H,T)$, and $(M,N)=(S^H,T\Sigma^-)$ where the matrix A is ill conditioned. Furthermore, our next two theorems, immediately implied by equations (3.2)–(3.4), define MCs and MPCs wherever we know null bases (for MCs) or some proper g-head and/or (extended) h-tail of the SVD (for MPCs).

Theorem 3.1. *Let $A=S\Sigma T^H$ be the SVD of an $m\times n$ matrix A of a rank ρ and let (g,h) be any pair of nonnegative integers g and $h<\rho$, e.g., $g=h=0$. Then the deletion of the zero rows and columns of the matrices $\tilde{S}_{g,h}^HA$, $A\tilde{T}_{g,h}$, and $\tilde{S}_{g,h}^HA\tilde{T}_{g,h}$ turns them into full rank matrices of sizes $\rho\times n$, $m\times\rho$, and $\rho\times\rho$, respectively, so that the matrices $M=\tilde{S}_{g,h}^H$ and $N=\tilde{T}_{g,h}$ and the matrix pair $(M,N)=(\tilde{S}_{g,h}^H,\tilde{T}_{g,h})$ are MCs for a rank deficient matrix A.*

The theorem defines MCs for any pair of nonnegative integers g and $h<\rho$. Let us next define MPCs for fixed g, h, and a (g,h) matrix.

Theorem 3.2. *Under the assumptions of Theorem 3.1, let σ lie in the interval $[\sigma_{\rho-h},\sigma_{g+1}]$. Write $d_j=\sigma/\sigma_j$ for $j=1,\dots,g,\rho-h+1,\dots,\rho$, $D^{(g)}=\mathrm{diag}(d_j)_{j=1}^g$, $D_h=\mathrm{diag}(d_j)_{j=\rho-h+1}^{\rho}$, $D_{g,h,k}(\sigma)=\mathrm{diag}(D^{(g)},I_{\rho-g-h},D_h,I_k)$ for $k=l$ and $k=r$, $F_l=D_{g,h,l}(\sigma)\tilde{S}_{g,h}^HA$, $F_r'=A\tilde{T}_{g,h}D_{g,h,r}(\sigma)$, $F_{l,r}=$*

$D_{g,h,l}(\sigma)\tilde{S}^H_{g,h}A\tilde{T}_{g,h}$, *and* $F'_{l,r} = \tilde{S}^H_{g,h}A\tilde{T}_{g,h}D_{g,h,r}(\sigma)$. *Then* $\mathrm{cond}_2 F_l = \mathrm{cond}_2 F'_r = \mathrm{cond}_2 F_{l,r} = \mathrm{cond}_2 F'_{l,r} = \sigma_{g+1}/\sigma_{\rho-h}$, *so that the matrices* $M = D_{g,h,l}(\sigma)\tilde{S}^H_{g,h}$ *and* $N = \tilde{T}_{g,h}D_{g,h,r}(\sigma)$ *and the matrix pairs* $(M, N) = (D_{g,h,l}(\sigma)\tilde{S}^H_{g,h}, \tilde{T}_{g,h})$ *and* $(M, N) = (\tilde{S}^H_{g,h}, \tilde{T}_{g,h}D_{g,h,r}(\sigma))$ *are MPCs for a* (g, h) *matrix* A *of a rank* ρ.

One can extend factorizations (3.2)–(3.4), Theorem 3.2, and much of our subsequent study to the case where we replace unitary matrix bases and the singular spaces above with their well conditioned approximations.

3.4 Null-based ACs

ACs and APCs are more readily available than MCs and MPCs (see Section 4.1), and they still facilitate solving linear systems and computing determinants (see Sections 7.7, 8, and 10). We can immediately devise an AC for a matrix whose null basis is available. (Recall that the null space of a matrix is its singular space associated with its singular value zero.)

Algorithm 3.1. Computing a null-based AC.

INPUT: *an* $m \times n$ *matrix* A *of a rank* ρ *and the left and right unitary null matrix bases* $S^{(null)} = (\mathbf{s}_j)_{j=\rho+1}^{m}$ *and* $T^{(null)} = (\mathbf{t}_j)_{j=\rho+1}^{n}$ *of the matrix* A.

OUTPUT: *a pair of matrices* U *of size* $m \times r$ *and* V *of size* $n \times r$ *such that the matrix* $C = A + UV^H$ *has full rank* $q = \min\{m, n\}$.

COMPUTATIONS:

Fix a positive σ *and compute and output a pair of matrices*

$$U = (\sigma\mathbf{s}_j)_j,\ V = (\mathbf{t}_j)_j \quad \text{for} \quad j = \rho + 1, \dots, q. \tag{3.5}$$

To verify correctness of the algorithm, observe that

$$C = A + UV^H = \sum_{j=1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H + \sigma \sum_{j=\rho+1}^{q} \mathbf{s}_j \mathbf{t}_j^H. \tag{3.6}$$

3.5 SVD-based APPs

We can readily extend Algorithm 3.1 to computing an SVD-based APC.

Algorithm 3.2. Computing an SVD-based APC.

INPUT: *an* $m \times n$ *matrix* A *of a rank* ρ, *two nonnegative integers* g *and* h *such that* $0 < g + h \le \rho$, *the* g*-head and the* h*-tail of the compact SVD of the matrix* A, *and a positive* σ *in the range* $[\sigma_{g+1}, \sigma_{\rho-h}]$.

OUTPUT: *a pair of matrices* U *of size* $m \times r$ *and* V *of size* $n \times r$ *such that*

$$\mathrm{cond}_2(A + UV^H) = \max\{\sigma, \sigma_{g+1}\}/\min\{\sigma, \sigma_{\rho-h}\}. \tag{3.7}$$

Here $r = g + h$ *for* $\sigma_g > \sigma > \sigma_{\rho-h+1}$, $r = g + h - 1$ *for* $\sigma = \sigma_g > \sigma_{\rho-h+1}$ *and for* $\sigma_g > \sigma = \sigma_{\rho-h+1}$, *and* $r = \rho - h - 2$ *for* $\sigma = \sigma_g = \sigma_{\rho-h+1}$.

COMPUTATIONS:

Compute and output a pair of matrices U of size $m \times r$ and V of size $n \times r$ such that

$$UV^H = \sum_{j=1}^{g} (\sigma - \sigma_j)\mathbf{s}_j\mathbf{t}_j^H + \sum_{j=\rho-h+1}^{\rho} (\sigma - \sigma_j)\mathbf{s}_j\mathbf{t}_j^H,$$

$$U = ((\sigma - \sigma_j)\mathbf{s}_j)_j,\ V = (\mathbf{t}_j)_j \quad \text{for} \quad j = 1, \dots, g; \rho - h + 1, \dots, \rho. \quad (3.8)$$

To verify correctness of the algorithm, observe that

$$C = A + UV^H = \sum_{j=g+1}^{\rho-h} \sigma_j\mathbf{s}_j\mathbf{t}_j^H + \sigma\widehat{\sum}_j \mathbf{s}_j\mathbf{t}_j^H \quad (3.9)$$

where the symbol $\widehat{\sum}_j$ stands for sum over j ranging from one to g and from $n - h + 1$ to ρ.

Given the g-head and the extended h-tail of a (g, h) matrix A, we can combine Algorithms 3.1 and 3.2 to arrive at an APP UV^H such that the matrix $A + UV^H$ has full rank q and satisfies equation (3.7).

APCs of optimal rank. More surprisingly, one can choose an APC of rank $r = \max\{g, h\}$ that supports the same decrease of $\mathrm{cond}_2 A$ achieved with Algorithm 3.2 for $r = g + h$, and this is the optimum decrease for a fixed rank r [18]. To compute such an optimum APC, one first brings the input matrix $A = S^H\Sigma T$ to the diagonal form Σ and then recursively applies the following result [18].

Theorem 3.3. *For any numbers $a_1 \geq b_1 \geq b_2 \geq a_2 > 0$, there exist real numbers u and v such that the 2×2 matrix $\begin{pmatrix} a_1 - u^2 & -uv \\ -uv & a_2 - v^2 \end{pmatrix}$ has singular values b_1 and b_2.*

This optimum APC is Hermitian and/or real if so is the input matrix.

Remark 3.1. *Theorem 3.3 also supports computing an APC of rank $2r-1$ that collapses $2r$ distinct singular values given with the associated singular vectors into a single median value.*

3.6 SVD-based dual and primal/dual APCs

Numerically, we can have problems with the APCs defined by the g-heads rather than the h-tails of the SVDs of the matrix A. If we compute an APC UV^H for an $(g, 0)$ matrix (e.g., by applying Algorithm 3.2 for $h = 0 < g \leq \rho$), then computing the matrix $A + UV^H$ we would have rounding errors of the order of $\epsilon||A||_2 = \epsilon\sigma_1$ where ϵ is the unit round-off. These errors are large relatively to the output norm $||A + UV^H||_2 = \max\{\sigma, \sigma_{g+1}\}$ if $\sigma_1 >> \sigma_{g+1} \geq \sigma$, and thus can ruin the positive effect of A-preconditioning.

We can avoid these numerical problems by applying *dual A-preconditioning*

$$A \leftarrow (C_-)^- = (A^- + VU^H)^- = A - AV(I_r + U^H AV)^{-1}U^H A \tag{3.10}$$

to a $(g, 0)$ matrix A and a *dual APC* VU^H of rank g (cf. (1.1)). We easily prove this simple extension of the SMW formula to $C = -A^-$ in Section 7.6.

More generally, for a (g, h) matrix A we define *primal/dual A-preconditioning*

$$A \leftarrow (C_-)^- = (\tilde{C}^- + V_g U_g^H)^- = \tilde{C} - \tilde{C}V_g(I_r + U_g^H \tilde{C} V_g)^{-1} U_g^H \tilde{C} \text{ for } \tilde{C} = A + \tilde{U}_h \tilde{V}_h^H. \tag{3.11}$$

If we know the g-head and h-tail of the SVD, then by choosing

$$U_g = ((\frac{1}{\sigma} - \frac{1}{\sigma_j})\mathbf{s}_j)_j,\ V_g = (\mathbf{t}_j)_j \ \text{ for } \ j = 1, \dots, g,$$

$$\tilde{U}_h = ((\sigma - \sigma_j)\mathbf{s}_j)_j,\ \tilde{V}_h = (\mathbf{t}_j)_j \ \text{ for } \ j = \rho - h + 1, \dots, \rho,$$

and using the symbol $\widehat{\sum}_j$ from equation (3.9), we obtain that

$$(C_-)^- = \sum_{j=g+1}^{\rho-h} \sigma_j \mathbf{s}_j \mathbf{t}_j^H + \sigma \widehat{\sum}_j \mathbf{s}_j \mathbf{t}_j^H.$$

4 SVD-free ACs and APCs

4.1 SVD-free APPs versus SVD-based APPs and MPPs

We run into two problems with the SVD-based APPs and MPPs.

Realistically, approximations to the g-heads of the SVDs of an input matrix A for smaller positive g are readily available, but the task is more costly for the h-tails of the SVDs for positive h [2, Sections 9.1 and 9.2.9], [22, pages 366 and 367], [28], [33].

We use two remedies, that is, generate the SVD-free ACs and APCs to substitute them for the h-tails (this is our next subject) and approximate the h-tails with null matrix bases where the norm $||A_h||_2$ is small, that is, where the h-tail nearly vanishes. (Our null bases algorithms in Sections 9.1 and 9.5 also employ the SVD-free ACs and APCs.)

Preserving matrix structure is another problem. The structure is not so readily compatible with the SVD-based MPPs and APPs of larger ranks, expressed through unitary bases for singular spaces, but as we show later, is readily compatible with the SVD-free APPs of any ranks.

Let us point out another advantage of APPs. Suppose the matrix $C = A + UV^H$ is well conditioned. Then it would stay such if we perturb an APP UV^H. Thus we can safely truncate the entries of the matrices U and V, represent them with fewer bits, and *first compute an APP* UV^H *of a rank* r *in about* $2mnr$ *ops and then the matrix* C *in* mn *ops, performing all ops with low precision but error-free.* This is harder to achieve for MPPs because rounding errors propagate more in multiplications than in additions.

4.2 Some readily available SVD-free ACs and APCs

Recall the direct methods for solving linear systems of equations that rely on recursive elimination of the entries or smaller blocks of entries of the coefficient matrix. Such methods include Gaussian elimination with and without pivoting, nested dissection, cyclic reduction and various modifications and block versions of these algorithms [34]–[39]. Each recursive elimination step amounts to a rank-one or small-rank modification of the matrix. Clearly (cf. Section 1.1), A-preconditioning can increase the magnitudes of small pivot elements or improve conditioning of small pivot blocks without row and column interchange, that is, without destroying matrix structure.

4.3 Randomized ACs

Our next goals are error-free randomized computations of ACs and their extension to lower precision but error-free computation of APCs.

We begin with a theorem that links the ranks of random APPs and the A-preconditioned matrices. In the two lines below we sketch this theorem assuming that $q = \min\{m, n\}$ and writing "$\Longrightarrow$" for "implies". Then we state and prove the theorem formally.

$$\{\operatorname{rank} C = q\} \Longrightarrow \{r \geq \operatorname{nul} A\},$$

$$\{r \geq \operatorname{nul} A \text{ for random unitary } U \text{ and } V\} \Longrightarrow \{\operatorname{rank} C = q \text{ (likely)}\}.$$

Theorem 4.1. *For a finite set Δ of a cardinality $|\Delta|$ in a ring $\mathbb{R}$, $q = \min\{m, n\}$, and four matrices $A \in \mathbb{R}^{m\times n}$ of a rank ρ, $U \in \Delta^{m\times r}$, $V^T \in \Delta^{r\times n}$, and $C = A + UV^T$, we have*

a) $\operatorname{rank} C \leq r + \rho$,

b) $\operatorname{rank} C = r + \rho$ *with a probability of at least* $1 - \frac{2r}{|\Delta|}$ *if* $r + \rho \geq q$ *and either the entries of both matrices U and V have been randomly sampled from the set Δ or $U = V$ and the entries of the matrix U have been randomly sampled from this set,*

c) $\operatorname{rank} C = r + \rho$ *with a probability of at least* $1 - \frac{r}{|\Delta|}$ *if* $r + \rho \geq q$, *the matrix U (respectively V) has full rank r, and the entries of the matrix V (respectively U) have been randomly sampled from the set Δ.*

Proof. Part a) is verified immediately. Now let an $(r + \rho) \times (r + \rho)$ submatrix $A_{r+\rho}$ of the matrix A have rank ρ and let $C_{r+\rho} = A_{r+\rho} + (UV^T)_{r+\rho}$ denote the respective $(r + \rho) \times (r + \rho)$ submatrix of the matrix C. Then clearly, $\operatorname{rank} C = \operatorname{rank} C_{r+\rho} = r+\rho$ if $U = V$ and if the entries of the matrix U are indeterminates. Since $\det C_{r+\rho}$ is a polynomial of a total degree of at most $2r$ in these entries, part b) follows from Lemma 2.1. Part c) is proved similarly to part b). □

The following algorithm recursively generates random APPs UV^H of ranks $r = 0, 1, \ldots$ and stops where we either arrive at an AC or exceed a fixed upper

bound r^+ on r. Towards subsequent extension to numerical computation of APCs, we employ random unitary matrices U and V.

Algorithm 4.1. Randomized computation of a unitary AC.

INPUT: *a normalized $m \times n$ matrix A of an unknown rank $\rho \le q = \min\{m, n\}$, an integer $r(+) \ge q - \rho$, and a black box Subroutine FULL RANK that tests if a matrix has full rank.*

OUTPUT: *FAILURE or an integer r, $q - \rho \le r \le r(+)$ and a pair of unitary matrices U of size $m \times r$ and V of size $n \times r$ such that the matrix $C = A + UV^H$ has full rank q.*

INITIALIZATION:

Set $r \leftarrow 0$, $U \leftarrow \emptyset^{m \times 0}$, and $V \leftarrow \emptyset^{n \times 0}$ where $\emptyset^{i \times 0}$ is the empty $i \times 0$ matrix.

COMPUTATIONS:

1. If r exceeds $r(+)$, stop and output FAILURE.

2. Otherwise apply Subroutine FULL RANK to test if the matrix $C = A + UV^H$ has full rank. If so, output the integer r and the matrices U and V and stop.

3. Otherwise sample two normalized random vectors $\mathbf{u}$ and $\mathbf{v}$ of dimension n such that $U^H\mathbf{u} = \mathbf{0}$ and $V^H\mathbf{v} = \mathbf{0}$, set $r \leftarrow r + 1$, $U \leftarrow (U, \mathbf{u})$, and $V \leftarrow (V, \mathbf{v})$, and go to Stage 1.

In virtue of Theorem 4.1, Algorithm 4.1 is correct and is likely to output also $r = q - \rho = \operatorname{nul} A$.

Remark 4.1. *The latter feature enables uncertified randomized computation of the rank and the nullity of a matrix. We can, however, readily certify that $r = \operatorname{rnul} A$, e.g., by testing whether $AC^{-}U = 0$ (see Section 9.1).*

The most costly stage of the algorithm is the application of the subroutine FULL RANK to the matrices C. We can decrease their expected number $q - \rho$ to at most $2\lceil \log_2(q - \rho) \rceil$ by incorporating the binary search for the nullity.

We can readily extend the algorithm to computing dual ACs VU^H of rank g for a $(g, 0)$ matrix and primal/dual ACs $\tilde{U}_h \tilde{V}_h^H$ of rank h and $V_g U_g^H$ of rank g for a (g, h) matrix (cf. equations (3.10) and (3.11)).

4.4 Extension to randomized computation of APCs

To implement Algorithm 4.1 numerically, one should just replace Subroutine FULL·RANK with estimating whether $\operatorname{cond}_2 C$ exceeds a fixed tolerance bound. Applied to $(0, h)$ matrix A, the resulting algorithm either fails or outputs a well conditioned matrix $C = A + UV^H$. Here is a sketch of an extension of Theorem 4.1 that supports the algorithm,

$$\{\operatorname{nrank} \tilde{C} = q\} \Longrightarrow \{r \ge \operatorname{nnul} \tilde{A}\},$$

$\{r \geq \text{nnul}\,\tilde{A}$ and random unitary U and $V\} \Longrightarrow \{\text{nrank}\,\tilde{C} = q$ (likely)$\}$.

Typically we cannot choose unitary APPs with a desired structure, but according to our analysis in Section 5 and the test results in Section 6, for a normalized $(0, h)$ matrix A the output APP is likely to be an APC, even if it is not unitary but random, properly scaled, well conditioned, and has a rank $r \geq h$, that is, satisfies requirements a)–d) in Section 1.1. An APP UV^H is *scaled consistently* with a $(0, h)$ matrix A, and the pair of A and UV^H is scaled consistently, if both the ratio $||A||_2/||UV^H||_2$ and its reciprocal are bounded by a moderate constant, that is, if this ratio is neither large nor small.

For a $(g, 0)$ matrix A and a dual APPs VU^H of a rank $r \geq g$ and for a (g, h) matrix A and the primal/dual pair of APPs $\tilde{V}_h\tilde{U}_h^H$ of a rank $\tilde{r} \geq h$ and V_gU_g of a rank $r \geq g$, we modify the requirements of consistent scaling as follows: the ratios $||VU^H||_2/||A^-||_2$, $||V_gU_g^H||_2/||A^-||_2$, and $||\tilde{U}_h\tilde{V}_h^H||_2/||A||_2$ are neither large nor small. Then again, our analysis and tests show that random well conditioned and consistently scaled APPs are likely to be APCs unless nnul A and/or nnul(A^-) exceed the rank of the primal and/or dual APP, respectively.

To verify that scaling is consistent for a dual APP, we need a crude estimate for $||A^-||_2 = 1/\sigma_\rho$. For a (g, h) matrix A, we first shift to the $(g, 0)$ matrix $\tilde{A} = A + \tilde{U}_h\tilde{V}_h$ and then readily estimate the value $||\tilde{A}^-||_2$ (cf. Section 7.7).

4.5 Structured and sparse APPs

All APPs of small ranks are structured, but next we supply various examples of sparse and/or structured APPs of any rank. In our extensive tests, these APPs were typically APCs for all classes of tested input matrices, and the rare opposite cases were readily cured with the recipes in the previous subsection. Hereafter we call these APPs *pseudo random*. More examples of them are expected and welcome from the readers.

Example 4.1. Circulant APPs. *$UV^H = F^{-1}D_rF$, where F is the $n \times n$ unitary matrix $\frac{1}{\sqrt{n}}(\exp\frac{2\pi ij\sqrt{-1}}{n})_{i,j=0}^{n-1}$ of the discrete Fourier transform at the nth roots of unity, such that $F^H = F^{-1}$, and D_r is the $n \times n$ diagonal matrix that has exactly r nonzero entries. They are fixed or sampled at random in a fixed set $\mathbf{S}$ and placed at r fixed or random places on the diagonal. This is a circulant APP of rank r [6, Section 2.6]. It is sufficient to perform $O(n \max\{r, \log n\})$ ops to multiply it by a vector. The bound decreases to $O(n \log r)$ where the r nonzeros occupy r successive positions on the diagonal. If $\mathbf{S}$ is a real set, then the APP is Hermitian. If the set $\mathbf{S}$ lies in the annulus $\{x : d_- \leq |x| \leq d_+\}$, then $\text{cond}_2(UV^H) = \text{cond}_2 D_r \leq d_+/d_-$. E.g., UV^H is Hermitian and $\text{cond}_2(UV^H) \leq 3$ if $\mathbf{S} = [0.5, 1.5]$.*

Example 4.2. f-circulant APPs *[6, Section 2.6]. In the previous example replace the matrix F with the matrix FD_- where $D_- = \text{diag}(g^i)_{i=0}^{n-1}$ and g is a primitive nth root of a scalar f such that $|f| = 1$. Then the APP is f-circulant. (It is circulant for $f = 1$ and skew-circulant for $f = -1$.) One can*

immediately extend the bounds from the previous example both on the APP's condition number and the arithmetic cost of its multiplication by vectors.

Example 4.3. Hermitian Toeplitz-like APPs I. *Define a Hermitian and nonnegative definite Toeplitz-like APP UU^H for an $n \times r$ Toeplitz matrix U. This APP has a rank of at most r and a displacement rank of at most four, and it can be multiplied by a vector in $O(n \log r)$ ops. We fix a well conditioned matrix U or define it by q random parameters for a nonnegative integer $q < n+r$ varying them until we yield well conditioning, and we output FAILURE if this does not work.*

Example 4.4. Hermitian Toeplitz-like APPs II. *Define a Hermitian and nonnegative definite Toeplitz-like APP UU^H for an $n \times r$ Toeplitz matrix $U = (T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T$ where T_i are $r \times r$ Toeplitz matrices and $0_{r,n_i}$ are $r \times n_i$ null matrices for $i = 1, \dots, k$ where k and $n_0, \dots, n_k$ are positive integers (fixed or random) such that $kr + n_1 + \cdots + n_k = n$. This APP has a displacement rank of at most $2k \le 2\lfloor n/r \rfloor$ and can be multiplied by a vector in $O(kr \log r)$ flops. We fix or choose at random the Toeplitz matrices T_i such that the resulting matrix U is well conditioned. The matrices T_i are general or special, e.g., circulant, f-circulant, triangular Toeplitz or banded Toeplitz matrices. If they are the scaled identity matrices $c_i I_r$, then the columns of the matrix U are orthogonal, and we make it unitary by choosing the scalars $c_1, \dots, c_k$ such that $c_1^2 + \cdots + c_k^2 = 1$. For banded Toeplitz matrices T_i with a constant bandwidth we only need $O(kr)$ ops to multiply the APP by a vector.*

Example 4.5. Structured or sparse Hermitian APPs I. *Define an APP UU^H where $U = PW$, P is a fixed or random $n \times n$ permutation matrix (in the simplest case $P = I_n$) and W is a fixed or random $n \times r$ block (that is, a block formed with r fixed or randomly selected columns) of the $n \times n$ matrix of the discrete Fourier, sine or cosine transform [6, Section 3.11], or of any well conditioned matrix with a fixed structure such as the displacement structure of Toeplitz, Hankel, Vandermonde, or Cauchy types (cf. [6], [40]–[44], and the bibliography therein), a semi-separable (rank) structure (cf. [25], [45]–[47], and the bibliography therein), or sparse structure [34]–[39], [48] where we can allow random diagonal scaling.*

Example 4.6. Structured or sparse Hermitian APPs II. *Define a well conditioned APP UU^H where $U = P(T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T$ for an $n \times n$ permutation matrix P and integers $k, n_1, \dots, n_k$ chosen as in Example 4.4 but for all i let T_i be $r \times r$ fixed or random matrices with some fixed structures, e.g., the matrices F of the discrete Fourier transform, the matrices of sign or cosine transforms, or semi-separable (rank structured) matrices, or let them be sparse matrices with fixed patterns of sparseness.*

5 APPs and conditioning

In this section we assume a square matrix A. Part of our study extends to its rectangular submatrices and matrices embedding it.

First we consider a normalized singular well conditioned matrix A with nullity r and a random unitary APP UV^H of rank r and show that the matrix $C = A + UV^H$ is expected to be nonsingular and well conditioned. Then we extend the same result to a consistently scaled pair of a $(0, r)$ matrix A and a random and well conditioned APP UV^H of rank r.

In [18] this result is rederived directly for a general $(0, r)$ matrix A and further refined in the Hermitian and Hermitian positive definite cases. The results in the latter cases can be viewed as quantitative complement to the interlacing property of the singular values of the Hermitian matrices A and $C = A + UU^H$ [2, Theorem 8.5.3], [21, Section 3.2.1], [49], [50].

Our analysis of primal APPs for $(0, r)$ matrices A can be extended to dual APPs for $(g, 0)$ matrices A and to primal/dual APPs for (g, h) matrices A.

5.1 ACs and conditioning

We first factorize the preconditioned matrix C.

Theorem 5.1. *Let A be an $n \times n$ matrix of rank $\rho = n - r$, so that $r = \operatorname{nul} A$. Let U and V be $n \times r$ matrices such that the $n \times n$ matrix $C = A + UV^H$ is nonsingular. Let $A = S\Sigma T^H$ be the SVD of the matrix A, where the matrices S and T are unitary, $\Sigma = \operatorname{diag}(\Sigma_A, 0_r)$, and $\Sigma_A = \operatorname{diag}(\sigma_j)_{j=1}^{\rho}$ is the diagonal matrix of the singular values of the matrix A. Write*

$$S^H U = \begin{pmatrix} U_\rho \\ U_r \end{pmatrix},\ T^H V = \begin{pmatrix} V_\rho \\ V_r \end{pmatrix},\ R_U = \begin{pmatrix} I_\rho & U_\rho \\ 0 & U_r \end{pmatrix},\ R_V = \begin{pmatrix} I_\rho & V_\rho \\ 0 & V_r \end{pmatrix}$$

where the matrices U_r and V_r have size $r \times r$. Then

a) $C = SR_U \operatorname{diag}(\Sigma_A, I_r) R_V^H T^H$ and

b) the matrices R_U, R_V, U_r, and V_r are nonsingular.

Proof. Write $\tilde{C} = \Sigma + S^H U V^H T$. Observe that $C = A + UV^H = S\Sigma T^H + SS^H UV^H TT^H = S\tilde{C}T^H$, $R_U \Sigma R_V^H = \Sigma$, $S^H U = R_U \begin{pmatrix} 0 \\ I_r \end{pmatrix}$, and $T^H V = R_V \begin{pmatrix} 0 \\ I_r \end{pmatrix}$. Deduce that $\tilde{C} = R_U \Sigma R_V^H + R_U \operatorname{diag}(0, I_r) R_V^H = R_U \operatorname{diag}(\Sigma_A, I_r) R_V^H$. Substitute this expression into the equation $C = T\tilde{C}S^H$ to arrive at part a). Part b) follows because the matrices C and $\tilde{C}$ are nonsingular. □

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

$$\frac{||\operatorname{diag}(\Sigma_A, I_r)||_2}{||R_U^{-1}||_2 ||R_V^{-1}||_2} \le ||C||_2 \le ||\operatorname{diag}(\Sigma_A, I_r)||_2 ||R_U||_2 ||R_V||_2,$$

$$\frac{||\operatorname{diag}(\Sigma_A^{-1}, I_r)||_2}{||R_U||_2 ||R_V||_2} \le ||C^{-1}||_2 \le ||\operatorname{diag}(\Sigma_A^{-1}, I_r)||_2 ||R_U^{-1}||_2 ||R_V^{-1}||_2,$$

so that

$$\frac{\operatorname{cond}_2 \operatorname{diag}(\Sigma_A, I_r)}{(\operatorname{cond}_2 R_U)\operatorname{cond}_2 R_V} \le \operatorname{cond}_2 C \le (\operatorname{cond}_2 R_U)(\operatorname{cond}_2 R_V)\operatorname{cond}_2 \operatorname{diag}(\Sigma_A, I_r).$$

Proof. The corollary follows from Theorem 5.1 because $||S||_2 = ||S^{-1}||_2 = ||T||_2 = ||T^{-1}||_2 = 1$, $\operatorname{cond}_2 M = ||M||_2||M^-||_2$ and $||M^H||_2 = ||M||_2$ for any matrix M. □

Let us specify the estimate for $\operatorname{cond}_2 C$ provided the matrices U and V are unitary and the matrix A is scaled properly.

Theorem 5.2. *If the matrices U and V are unitary, then we have*

$$||R_U||_2 \le \sqrt{2},\ ||R_V||_2 \le \sqrt{2},$$

$$||R_U^{-1}||_2 \le 1 + \sqrt{2}||U_r^{-1}||_2,\ ||R_V^{-H}||_2 = ||R_V^{-1}||_2 \le 1 + \sqrt{2}||V_r^{-1}||_2.$$

Proof. The theorem follows because

$$R_U = \operatorname{diag}(I_\rho, 0) + (0, S^H U)\operatorname{diag}(0, I_r),$$

$$R_V = \operatorname{diag}(I_\rho, 0) + (0, T^H V)\operatorname{diag}(0, I_r),$$

$$R_U^{-1} = \operatorname{diag}(I_\rho, 0) + \begin{pmatrix} 0 & -U_\rho \\ 0 & I_r \end{pmatrix}\operatorname{diag}(0, U_r^{-1}),$$

$$R_V^{-H} = \operatorname{diag}(I_\rho, 0) + \begin{pmatrix} 0 & -V_\rho \\ 0 & I_r \end{pmatrix}\operatorname{diag}(0, V_r^{-1}),$$

the 2-norms of the matrices S, T, U, and V are equal to one, and $||(X, Y)||_2 = ||(X, Y)^H||_2 \le \sqrt{2}$ for a pair of unitary matrices X and Y. □

Theorem 5.3. *Under the assumptions of Theorem 5.1, suppose that*

$$\sigma_{n-r} \le 1 \le \sigma_1. \tag{5.1}$$

Then $||\operatorname{diag}(\Sigma_A, I_r)||_2 = ||A||_2$ *and* $||(\operatorname{diag}(\Sigma_A, I_r))^{-1}||_2 = \sigma_{n-r}$.

Proof. Immediate by inspection. □

Corollary 5.2. *Write* $p_r = ||R_U^{-1}||_2||R_V^{-1}||_2 \le (1+\sqrt{2}||U_r^{-1}||_2)(1+\sqrt{2}||V_r^{-1}||_2)$. *Under bounds (5.1) and the assumptions of Theorem 5.1, suppose that the matrices U and V are unitary. Then*

a) $||A||_2/p_r \le ||C||_2 \le 1 + ||A||_2 \le 2||A||_2$,
b) $1/(2\sigma_{n-r}) \le ||C^{-1}||_2 \le p_r/\sigma_{n-r}$, *and*
c) $(\operatorname{cond}_2 A)/(2p_r) \le \operatorname{cond}_2 C \le 2p_r \operatorname{cond}_2 A$.

Proof. Part a) follows immediately because $||C||_2 \le ||A||_2 + ||UV^H||_2$ and $||UV^H||_2 = 1 \le \sigma_1 = ||A||_2$.

To prove part b), combine Corollary 5.1 with Theorems 5.2 and 5.3.

Part c) immediately follows from parts a) and b). □

5.2 Small-norm perturbation of an AC and conditioning

Our next theorem shows that our estimates in Corollary 5.2 change little in a small-norm perturbation of a singular matrix A into a generally nonsingular matrix $A + E$.

Theorem 5.4. *Under the assumptions of Corollary 5.2, let the matrix* $\tilde{C} = C + E$ *be nonsingular. Write* $\delta = ||E||_2$, *and* $\delta_C = \delta||C^{-1}||_2$. *Then we have*

a) $||\tilde{C}||_2 \le \delta + ||C||_2$,

b) if $\delta_C < 1$, *then* $||\tilde{C}^{-1}||_2 \le ||C^{-1}||_2(1 + (1 - \delta_C)^{-1}\delta_C)$, *so that* $\operatorname{cond}_2 \tilde{C} \le (1 + (1 - \delta_C)^{-1}\delta_C)(1 + \delta/||C||_2) \operatorname{cond}_2 C$, *and*

c) if the matrices C *and* E *are Hermitian and nonnegative definite, then* $||\tilde{C}^{-1}||_2 \le ||C^{-1}||_2$, *so that* $\operatorname{cond}_2 \tilde{C} \le (1 + \delta/||C||_2) \operatorname{cond}_2 C$.

Proof. Part a) is proved immediately, similarly to part a) of Corollary 5.2.

To prove part b), substitute the matrices $A + UV^H = \tilde{C}$, $A = C$, $U = -E$, and $V = I_n$ into the SMW formula and obtain that $\tilde{C}^{-1} = C^{-1}(I_n + E(I_n - C^{-1}E)^{-1}C^{-1})$. Part b) follows from this equation and the assumption that $\delta_C = \delta||C^{-1}||_2 < 1$.

Part c) follows because the matrix $E = \tilde{C} - C$ is nonnegative definite and the matrix C is positive definite. □

5.3 The impact of scaling APPs

Under the assumptions of Corollary 5.2, the ratio $\frac{\operatorname{cond}_2 C}{\operatorname{cond}_2 A}$ is not large unless the product $||U_r^{-1}||_2||V_r^{-1}||_2$ is large. Moreover, if we relax assumption (5.1), assume that $\sigma_1 = 1/\theta$ or $\sigma_{n-r} = \theta$ for $\theta > 1$, and ignore the resulting dynamics in the factors of $||U_r^{-1}||_2$ and $||V_r^{-1}||$, then the upper estimate $2p_r$ on this ratio (in Corollary 5.2c) would increase by roughly the factor of θ. As confirmed by the results of our extensive tests, we can avoid this increase by choosing well conditioned and consistently scaled APPs. We can apply error-free scaling by the powers of two.

For random unitary $n \times r$ matrices U and V, we can also view the $n \times r$ unitary matrices $S^H U$ and $T^H V$ in Theorem 5.1 as random, and then the norms of the inverses of their $r \times r$ southern-most submatrices U_r and V_r are likely to be reasonably bounded for smaller r. (If these norms are large, then $\operatorname{cond}_2 C$ is large, and if we detect this, we can resample the random matrices U and V.)

For consistent scaling we estimate the 2-norms of the matrix A and the APPs UV^H, VU^H, $\tilde{U}_h\tilde{V}_h$, and $V_g U_g$, and if a dual or primal/dual APC is sought, then also the norm $||A^-||_2$.

The effective algorithms in [3, Section 5.3.3] compute quite tight bounds on both norms $||A||_2$ and $||A^-||_2$. Here are some cruder low cost bounds in [2, Section 2.3.2 and Corollary 2.3.2],

$$1 \le ||A||_2 / \max_{i,j} |a_{i,j}| \le \sqrt{mn},$$

$$1 \le \sqrt{||A||_1||A||_\infty}/||A||_2 \le (mn)^{1/4}.$$

6 Numerical tests for generating APCs

In our tests we generated singular and ill-conditioned nonsingular matrices of 16 types, modified them with random and pseudo random APPs of eight types, and computed the condition numbers of the input and modified matrices. We run such tests for over 100,000 input instances and observed quite similar statistics for all selected classes of APPs. We display sample data in two tables for the 16 matrices and the APPs of a selected type.

In all tests we used the following CPU and memory configuration, operating system, mathematical application software, and random number generator:

CPU	AMD Athlon XP 2800+ 2.09GHZ
Memory	512MB
OS	Microsoft Windows XP Professional Version 2002 Service Pack 2
Platform	Matlab Version 7.0.0.19920(R14)
Random Number Generator	Matlab Statistics Toolbox's Uniform Distribution

Unless we specified otherwise, we sampled the entries of random matrices in the closed line interval $[-1, 1]$.

Since we mostly dealt with real (in particular integer or rational) matrices, we use the nomenclatures "symmetric" and "nonsymmetric" rather than "Hermitian" and "non-Hermitian" (cf. [2], [3]).

Throughout this section we assigned the values $n = 100$ and $\nu = 1, 2, 4, 8$ to the parameters n and ν.

6.1 Generation of singular input matrices A

We generated the following singular input matrices A. (We use abbreviations "s" for "symmetric" and "n" for "nonsymmetric".)

1n. *Nonsymmetric matrices of type I with nullity* ν. $A = A_\nu(\Sigma_\nu, G, H) = G\Sigma_\nu H^T$ are $n \times n$ matrices where G and H are $n \times n$ random unitary matrices with real entries (also called orthogonal [2], [3]), that is, the Q-factors in the QR factorizations of random real matrices; $\Sigma_\nu = \mathrm{diag}(\sigma_j)_{j=1}^n$ is the diagonal matrix filled with zeros and the singular values of the matrix A; $\sigma_{j+1} \le \sigma_j$ for $j = 1, \dots, n-1$, the values $\sigma_2, \dots, \sigma_{n-\nu-1}$ are randomly sampled in the semi-open interval $[0.1, 1)$, $\sigma_1 = 1$, $\sigma_{n-\nu} = 0.1$, $\sigma_j = 0$ for $j = n-\nu+1, \dots, n$, so that $\mathrm{cond}_2 A = 10$.

1s. *Symmetric matrices of type I with nullity* ν. The same as in part 1n, but for $G = H$.

2n. *Nonsymmetric matrices of type II with nullity* ν. $A = (W, WZ)$ where W and Z are random unitary matrices of sizes $n \times (n-\nu)$ and $(n-\nu) \times \nu$, respectively.

2s. *Symmetric matrices of type II with nullity* ν. $A = WW^H$ where W are random unitary matrices of size $n \times (n-\nu)$.

3n. *Nonsymmetric Toeplitz-like matrices with nullity ν.* $A = c(T, TS)$ for random Toeplitz matrices T of size $n \times (n-\nu)$ and S of size $(n-\nu) \times \nu$ and for a positive scalar c such that $||A||_2 \approx 1$.

3s. *Symmetric Toeplitz-like matrices with nullity ν.* $A = cTT^H$ for random Toeplitz matrices T of size $n \times (n-\nu)$ and a positive scalar c such that $||A||_2 \approx 1$.

4n. *Nonsymmetric Toeplitz-like matrices with nullity one.* $A = (a_{i,j})_{i,j=0}^{n-1}$ is an $n \times n$ Toeplitz matrix. Its entries $a_{i,j} = a_{i-j}$ are random for $i-j < n-1$. The entry $a_{n-1,0} = a_{n-1}$ is selected to ensure that the last row is linearly expressed through the other rows.

4s. *Symmetric Toeplitz-like matrices with nullity one.* $A = (a_{i,j})_{i,j=0}^{n-1}$ is an $n \times n$ Toeplitz matrix. Its entries $a_{i,j} = a_{i-j}$ are random for $|i-j| < n-1$, whereas the entry $a = a_{0,n-1} = a_{n-1,0}$ is a root of the quadratic equation $\det A = 0$. We have repeatedly generated the matrices A until we arrived at the equation having real roots.

6.2 Generation of nonsingular $(0, \nu)$ ill conditioned input matrices A

We modified the above matrices with nullity ν to turn them into nonsingular matrices with numerical nullity ν in two ways. (To our previous abbreviations "s" and"n", we add another "n" for "nonsingular".)

1nn and 1ns. *Matrices of type I having numerical nullity ν.* The same matrices as in parts 1n and 1s in the previous subsection except that now $\sigma_j = 10^{-16}$ for $j > n-\nu$, so that $\text{cond}_2 A = 10^{16}$.

2nn, 3nn, 4nn, 2ns, 3ns, and 4ns. *Matrices of type II and Toeplitz-like matrices having numerical nullity ν.* $A = W/||W||_2 + \beta I_n$ where we defined the matrices W in the same way as the matrices A in the previous subsection and set the scalar β equal to 10^{-16} in the symmetric case, so that $\sigma_1(A) = 1 + 10^{-16}$, $\sigma_j(A) = 10^{-16}$, $j = n-\nu+1, \dots, n$, whereas in the nonsymmetric case we iteratively computed a nonnegative scalar β such that $\sigma_1(A) \approx 1$ and

$$10^{-18} \le \sigma_{n-\nu+1}(A) \le 10^{-16}. \tag{6.1}$$

We began this iterative process with choosing $\beta = 10^{-16}$, which implied that $\sigma_j(A) \le 10^{-16}/(1+10^{-16})$ for $j = n-\nu+1, \dots, n$. If also $10^{-18} < \sigma_{n-\nu+1}(A)$, so that bounds (6.1) held, we output this value of β and stopped. Otherwise we recursively set $\beta \leftarrow 10^{-16}\beta/\sigma_{n-\nu+1}(A)$. We output the current value of β and stopped as soon as bounds (6.1) were satisfied for the resulting matrix A. If they were not satisfied in 100 recursive steps, we stopped and output failure.

6.3 Generation of APPs and the data on conditioning

In Tables 6.1 and 6.2 we display the data on generating APPs UV^H and the conditioning of the matrices $A + UV^H$.

In the first column of each table we display the type of the input matrix A.

The second and the third columns show the values of ν, denoting the nullity (or numerical nullity) of the basic matrix A, and $\operatorname{cond} A$, denoting its condition number.

The fourth and the fifth columns show the type and the rank r of the APP UV^H from Example 4.6 (or its correction $U_1V_1^H$ defined below) where we set $T_i = cI_r$ for all i with scalar c chosen to normalize the matrix U.

The sixth columns show the condition numbers $\operatorname{cond}_2 C$ of the preconditioned matrices $C = A + UV^H$.

Wherever we had $\operatorname{cond}_2 C > 10^5$, we computed an APP $U_1V_1^T$, the matrix $C_1 = A + U_1V_1^T$, and $\operatorname{cond}_2 C_1$, and then in the respective line of the table we displayed this condition number $\operatorname{cond}_2 C_1$ in the seventh column and the rank of the new APP $U_1V_1^H$ in the fifth column. Wherever we had $\operatorname{cond}_2 C \le 10^5$, we left the respective line blanc in the seventh column.

To generate the APP $U_1V_1^H$, we either reapplied the same rules as before but with the APP's rank r incremented by one (we did this for the tests covered in Table 6.1) or defined this APP by the formulae $U_1 \leftarrow C^{-1}U$, $V_1^H \leftarrow V^HC^{-1}$ (cf. equation (1.3) and Section 9.6), without changing the rank r (we defined such APPs $U_1V_1^H$ for the tests covered in Table 6.2).

We applied the same tests and obtained quite similar results for APPs of seven other types, namely,

a) and b) for APPs from Example 4.6 but restricted to the sparse Toeplitz APCs, such that $T_i = c_iI_r$ where we first randomly sampled the coefficients c_i from one of the sets $\{-1, 1\}$ for type a) or $\{-2, -1, 1, 2\}$ for type b) and then normalized the matrix U by scaling,

c) for APPs from the same example but with T_i being random circulant matrices,

d) for APPs from Example 4.1,

e) and f) for APPs from Example 4.3 with random parameters from one of the line intervals $[-1, 1]$ for type e) or $[-10, 10]$ for type f), and

g) random APPs.

For every selected APP UV^H in our tests we computed the matrices $C^{(p)} = A + 10^pUV^H$ for $p = -10, -5, 0, 5, 10$. In all our tests, the condition numbers of the matrices $C^{(p)}$ were always minimized for $p = 0$ and grew steadily (within the factor of $|p|$) as the integer $|p|$ grew. In Tables 6.1 and 6.2 we reported only the results for $p = 0$.

Table 6.1: APPs and conditioning I

Type	ν	Cond(A)	r	Cond(C)	Cond(C_1)
1n	1	8.40E+16	1	3.21E+2	
1n	2	4.56E+16	2	4.52E+3	
1n	4	3.90E+18	5	2.09E+5	1.81E+3
1n	8	5.69E+16	8	6.40E+2	
1s	1	1.98E+16	1	5.86E+2	
1s	2	3.69E+16	2	1.06E+4	
1s	4	2.91E+16	4	1.72E+3	
1s	8	3.36E+16	8	5.60E+3	
2n	1	3.48E+16	1	8.05E+1	
2n	2	1.53E+17	2	6.82E+3	
2n	4	2.73E+16	4	2.78E+4	
2n	8	1.23E+17	8	3.59E+3	
2s	1	4.13E+16	1	1.19E+3	
2s	2	4.67E+16	2	1.96E+3	
2s	4	4.40E+16	4	1.09E+4	
2s	8	1.33E+18	8	9.71E+3	
3n	1	3.96E+16	1	2.02E+4	
3n	2	2.18E+17	2	1.53E+3	
3n	4	1.37E+18	4	6.06E+2	
3n	8	4.24E+17	8	5.67E+2	
3s	1	1.69E+17	1	2.39E+4	
3s	2	4.58E+16	2	2.38E+3	
3s	4	1.39E+17	4	1.69E+3	
3s	8	1.60E+17	8	6.74E+3	
4n	1	1.22E+17	1	4.93E+2	
4n	2	3.26E+16	2	4.48E+2	
4n	4	5.99E+16	4	2.65E+2	
4n	8	1.23E+17	8	1.64E+2	
4s	1	3.22E+15	1	1.45E+3	
4s	2	2.34E+16	2	5.11E+2	
4s	4	1.09E+17	4	7.21E+2	
4s	8	2.29E+16	8	2.99E+2	

Table 6.2: APPs and conditioning II

Type	ν	Cond(A)	r	Cond(C)	Cond(C_1)
1n	1	2.63E+16	1	2.81E+2	
1n	2	2.98E+16	2	1.66E+3	
1n	4	3.85E+16	4	4.26E+3	
1n	8	3.55E+17	8	8.60E+2	
1s	1	5.10E+16	1	5.29E+2	
1s	2	2.22E+16	2	3.24E+4	
1s	4	2.96E+16	4	3.96E+4	
1s	8	2.88E+16	8	1.69E+3	
2n	1	1.06E+17	1	1.86E+2	
2n	2	3.58E+16	2	4.05E+2	
2n	4	9.90E+16	4	5.84E+3	
2n	8	8.29E+16	8	1.10E+4	
2s	1	1.25E+16	1	8.34E+2	
2s	2	2.71E+16	2	9.63E+2	
2s	4	5.91E+16	4	8.90E+3	
2s	8	5.49E+16	8	1.81E+4	
3n	1	1.85E+17	1	3.63E+3	
3n	2	9.71E+16	2	2.13E+4	
3n	4	1.76E+17	4	2.49E+3	
3n	8	3.70E+17	8	7.61E+2	
3s	1	1.30E+17	1	6.03E+3	
3s	2	1.03E+17	2	2.15E+4	
3s	4	7.20E+16	4	1.46E+4	
3s	8	8.98E+16	8	1.73E+6	9.93E+2
4n	1	1.74E+18	1	1.08E+3	
4n	2	9.08E+16	2	2.04E+2	
4n	4	2.57E+16	4	5.81E+1	
4n	8	7.66E+15	8	3.33E+1	
4s	1	2.60E+16	1	4.21E+2	
4s	2	2.55E+16	2	1.88E+2	
4s	4	7.80E+16	4	8.95E+2	
4s	8	1.81E+16	8	3.83E+2	

PART II. SCHUR AGGREGATION, LINEAR SYSTEM SOLVING AND DETERMINANTS

Our next goal is the transition from APPs to factorization, solving linear systems of equations, and computing matrix determinants based on the primal and dual SMW formulae and iterative refinement.

7 Transition from APPs to MPPs

7.1 Some variations of the SMW formula

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. Deduce that

$$A = (I_m - UV^H C^-)C,\ A^- = C^-(I_m - UV^H C^-)^{-1} \tag{7.1}$$

for $m \geq n$ and

$$A = C(I_n - C^- UV^H),\ A^- = (I_n - C^- UV^H)^{-1} C^- \tag{7.2}$$

for $m \leq n$ (cf. equations (2.1) and (2.2)) where the matrices $I_m - UV^H C^-$ and $I_n - C^- UV^H$ are nonsingular.

Substitute $A \leftarrow I_m$ and either $V^H \leftarrow -V^H C^-$ for $m \geq n$ or $U \leftarrow -C^- U$ for $m \leq n$ into the SMW formula in Theorem 2.1 and obtain that

$$(I_m - UV^H C^-)^{-1} = I_m + U(I_r - V^H C^- U)^{-1} V^H C^- \tag{7.3}$$

for $m \geq n$ and

$$(I_n - C^- UV^H)^{-1} = I_n + C^- U(I_r - V^H C^- U)^{-1} V^H \tag{7.4}$$

for $m \leq n$.

By combining equations (7.1)–(7.4), we extend the SMW formula to rectangular matrices of full rank as follows,

$$A^- = C^- + C^- U(I_r - V^H C^- U)^{-1} V^H C^-. \tag{7.5}$$

Recall that $A^- A = I_n$ for $m \geq n$, $AA^- = I_n$ for $m \leq n$, and $I_r - V^H C^- U$ is the Schur complement of the block C in the block matrix $\begin{pmatrix} C & V^h \\ U & I_r \end{pmatrix}$.

7.2 Schur Aggregation

Assume a consistently scaled pair of a $(0, r)$ matrix A and a random or pseudo random well conditioned APP UV^H of a rank r. Then, according to the two previous sections, the matrix $C = A + UV^H$ is expected to be well conditioned. The Schur complement $I_r - V^H C^- U$ is perfectly conditioned for $r = 1$. Even if it is ill conditioned for a smaller $r > 1$, we can still readily factorize it or apply to it the CG or GMRES algorithms [2, Sections 10.2–10.4], [48], [51]–[56].

For $r < \min\{m, n\}$ the transition from matrices A to the Schur complements $I_r - V^H C^- U$ is said to be the *Schur Aggregation*.

7.3 SVD-based computation of Schur complements

Motivated by equations (7.1)–(7.5), we seek the Schur complement $I_r - V^H C^- U$. Here are the explicit expressions for it in terms of the trailing singular values of the matrix A provided the APP UV^H is defined by equations (3.8) for $g = 0$.

Theorem 7.1. *Given a real σ, four positive integers m, n, ρ, and $h < \rho$, and three matrices S_h, Σ_h, and T_h defining the h-tail of the SVD of an $m \times n$ matrix A of rank ρ, let equations (3.8) for $g = 0$ define a pair of matrices $U = ((\sigma - \sigma_j)\mathbf{s}_j)_{j=\rho-h+1}^{\rho}$ and $V = (t_j)_{j=\rho-h+1}^{\rho}$, and let $C = A + UV^H$. Then $I_r - V^H C^{-1} U = \sigma^- \mathrm{diag}(\sigma_j)_{j=\rho-h+1}^{\rho}$.*

Proof. We have

$$C = A + UV^H = \sum_{j=1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H + \sum_{j=\rho-h+1}^{\rho} (\sigma - \sigma_j)\mathbf{s}_j \mathbf{t}_j^H = \sum_{j=g+1}^{\rho-h} \sigma_j \mathbf{s}_j \mathbf{t}_j^H + \sigma \sum_{j=\rho-h+1}^{\rho} \mathbf{s}_j \mathbf{t}_j^H,$$

$$C^- = \sum_{j=g+1}^{\rho-h} \sigma_j^{-1} \mathbf{s}_j \mathbf{t}_j^H + \sigma^{-1} \sum_{j=\rho-h+1}^{\rho} \mathbf{s}_j \mathbf{t}_j^H$$

for U and V defined by equations (3.8). Therefore,

$$V^H C^- U = \sigma^{-1} \mathrm{diag}(\sigma - \sigma_j)_{j=\rho-h+1}^{\rho},\ I_r - V^H C^- U = \sigma^{-1} \mathrm{diag}(\sigma_j)_{j=\rho-h+1}^{\rho}.$$

□

7.4 The norms and conditioning of the Schur complements in the Schur Aggregation

Let us estimate the singular values of the matrix $I_r - V^H C^- U$ in terms of the r smallest singular values of an $n \times n$ normalized matrix A provided the matrix C is well conditioned. We rely on a simple corollary from the Courant–Fischer Minimax Characterization [2, Theorem 8.1.2], [22, Theorem 3.3.2].

Theorem 7.2. *Let W denote an $m \times n$ matrix of full rank $\rho = \min\{m, n\}$ and let $\sigma_1(W), \dots, \sigma_\rho(W)$ denote its singular values in the nonincreasing order. Write $\sigma_+(W) = \sigma_1(W)$, $\sigma_-(W) = \sigma_\rho(W)$. Then we have $\sigma_j(M)\sigma_-(W) \le \sigma_j(MW) \le \sigma_j(M)\sigma_+(W)$ and $\sigma_j(N)\sigma_-(W) \le \sigma_j(WN) \le \sigma_j(N)\sigma_+(W)$ for $j = 1, \dots, \rho$, an $m \times m$ matrix M, and an $n \times n$ matrix N.*

Theorem 7.3. *Under the assumption of Theorem 7.2 let $m = n$. Then we have $|\sigma_j(W) - 1| \le \sigma_j(W + I_n) \le \sigma_j(W) + 1$ for $j = 1, 2, \dots, n$.*

Proof. Let $S\Sigma T^H$ be the SVD of the matrix W. Then

$$\sigma_j^2(W + I_n) = \sigma_j^2(\Sigma + S^H T) = \sigma_j(\Sigma^2 + T^H S\Sigma + \Sigma S^H T + I_n).$$

By applying the Courant–Fischer Minimax Characterization, we obtain $-\sigma_j(F) \le \sigma_j^2(W + I_n) - \sigma_j(\Sigma^2 + I_n) \le \sigma_j(F)$ and $\sigma_j(F) \le 2\sigma_j(W)$ where $F = T^H S\Sigma + \Sigma S^H T$. The claimed bounds follow because $\sigma_j(\Sigma^2 + I_n) = \sigma_j^2(W) + 1$. □

Theorem 7.4. *For positive integers m, n, $r \le \min\{m,n\}$, and $q = \max\{m,n\}$, a normalized $m \times n$ matrix A, and a pair of unitary 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 that the matrices A and $C = A + UV^H$ have full rank. Then the matrix G is nonsingular, and we have $\sigma_j(G^{-1}) \ge \sigma_-^2(C)/\sigma_{q-j}(A) - \sigma_-(C)$ for $j = |m-n|+1, \dots, r$. Furthermore, if $\sigma_j(G^{-1}) \ge \sigma_+(C)$ (in particular if $\sigma_-^2(C)/\sigma_{q-j}(A) - \sigma_-(C) \ge \sigma_+(C)$)), then we also have $\sigma_j(G^{-1}) \le \sigma_+^2(C)/\sigma_{q-j}(A) + \sigma_+(C)$ for $j = |m-n|+1, \dots, r$.*

Proof. Let $m \ge n$. Deduce from equation (7.1) that the matrix $H = I_m - UV^H C^-$ is nonsingular. So is the matrix G as well because $\det G = \det H$ [32, Exercise 1. 14]. Now combine equation (7.1) with Theorem 7.2 for $M = H$ and $W = C$ to obtain that

$$\sigma_-(C) \le \sigma_j(A)/\sigma_j(H) = \sigma_j(A)\sigma_{m-j+1}(H^{-1}) \le \sigma_+(C) \text{ for } j = 1, \dots, \rho. \quad (7.6)$$

By combining Theorem 7.2 for $W = C^-$ and $M = UG^{-1}V^H$ and Theorem 7.3 for $W = H^{-1}$ with equation (7.3), deduce that

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

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

Therefore, $\sigma_j(G^{-1}) \ge \sigma_-(C^-)(\sigma_j(H^{-1}) - 1)$, and if $\sigma_j(G^{-1}) \ge \sigma_+(C)$, then also $\sigma_j(G^{-1}) \le \sigma_-(C^+)(\sigma_j(H^{-1}) + 1)$ for $j = 1, \dots, r$. By combining these bounds with the ones in (7.6), obtain the theorem for $m \ge n$. For $m \le n$ proceed similarly but use equations (7.2) and (7.4) instead of (7.1) and (7.3) and replace M with N when you invoke Theorem 7.2. □

If $m = n = q$, then Theorem 7.4 supplies some estimates for all singular values $\sigma_{r-j}(G)$ of the matrix G for $j = 1, \dots, r$ in terms of the values $\sigma_+(C)$, $\sigma_-(C)$, and the respective trailing singular values $\sigma_{n-j}(A)$ of the matrix A.

Due to the results in Section 5, we should expect that all singular values of the matrix C lie in or near the line segment $[\sigma_1(A), \sigma_{n-r}(A)]$. Now unless the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ is large, the matrix C is well conditioned, and then in virtue of Theorem 7.4 all singular values of the matrix G lie the c^2-dilation of the line segment $[\sigma_{n-r+1}(A) - 1/c, \sigma_n(A) + 1/c]$ for $c = \sigma_+^2(C)/\sigma_-^2(C))$. If furthermore the ratio $\sigma_{n-r+1}(A)/\sigma_n(A)$ is not large, then all singular values of the matrix G have roughly the same order of magnitude, and so the matrix G is well conditioned.

7.5 Solving linear systems with APCs and iterative refinement

In this section we use floating-point summation and product algorithms that enable us to avoid unnecessary loss of the input information. In Section 12 we describe some elementary algorithms of this kind (including the ones from our

earlier works [58] and [59]) and give pointers to the more advanced ones. We refer to these algorithms as the *floating-point subroutines from Section 12*.

Now suppose that A and C are nonsingular $n \times n$ matrices and A is a $(0, r)$ matrix. Then equation (7.5) reduces a linear system with the matrix A to linear systems with the matrices C and $G = I_r - V^H C^{-1} U$, whereas Theorem 7.4 bounds all singular values of the Schur complement G.

If the matrix G is ill conditioned, we can reapply to it the Schur Aggregation, and this can be repeated recursively. In each recursive application, the singular value of the matrix G are defined by a shorter segment of the spectrum of the singular values of the matrix A and thus inherit fewer and ultimately no jumps from these singular values.

We can also compute the matrix C by recursively updating the matrix A with small-rank or just rank-one modifications, but this step requires high precision solution of some ill conditioned linear systems of equations. (The situation is different for the dual Schur Aggregation.)

Now suppose that the value $\sigma_{\rho-r+1}(A)$ is small, then the theorem implies that all r singular values of the matrix G are small, that is, the norm $||G||_2$ is small. If in addition the matrix C is well conditioned, then the theorem also implies that the matrix G is well conditioned unless the ratio $\sigma_{\rho-r+1}(A)/\sigma_\rho(A)$ is large.

Since the norm $||G||_2$ is small, it may seem that we must use a high precision when we compute the matrices $W = C^{-1}U$ (or $V^H C^{-1}$) and $V^H C^{-1} U \approx I_r$. We, however, use a variant of the iterative refinement in [2, Section 3.5.3], [3, Sections 3.3.4 and 3.4.5], [57, Chapter 11] and proceed numerically by using the single/double precision and the floating-point subroutines from Section 12.

The customary iterative refinement algorithm computes the double precision numerical approximation W_0 to the matrix $W = C^{-1}U$ by adding at least p correct bits to an entry of the matrix W at every iteration until the matrix W_0 is computed. Here $p \geq 1$ if the matrix C is well conditioned. The algorithm does not improve the approximation to the matrix W any further, but our variant does this, even though we keep computing with the single and double precision apart from the application of the algorithms from Section 12.

We fix a sufficiently large integer k and compute the matrices $W = \sum_{i=0}^{k} W_i$ and $G = I_r - V^H W = I_r + \sum_{i=1}^{k} F_i$ implicitly, by writing $U_0 = U$ and successively computing the matrices $W_i \leftarrow C^{-1}U_i$, $U_{i+1} \leftarrow U_i - CW_i$, $F_i \leftarrow -V^H W_i$ for $i = 0, \dots, k$, and finally $G \leftarrow (I_r + F_0) + \sum_{i=1}^{k} F_i$.

Let us estimate rounding errors in the subiteration $U_0 = U$, $W_i \leftarrow C^{-1}U_i$, $U_{i+1} \leftarrow U_i - CW_i$, $i = 0, \dots, k$. Write $E_i = C^{-1}U_i - W_i$, $Z_i = U_{i+1} - U_i + CW_i$, $e_i = ||E_i||_2$, $u_i = ||U_i||_2$, and $z_i = ||Z_i||_2$, and extend the analysis in [3, Section 3.3.4] and [57, Chapter 11] to our modified iteration.

As in the classical iterative refinement algorithm, approximation of the matrix $C^{-1}U$ is self-correcting, but in our variant these errors are of the order of the norms $||U_i - CW_i||_2$ versus the norms $||U - C\sum_{j=0}^{i} W_j||_2$ in the classical iteration. The residual norms $||U_i||_2$ are decreasing to zero as $i \to \infty$, and so are the error norms e_i.

Theorem 7.5. *We have* $U_{i+1} = Z_i + CE_i$ *and consequently* $u_{i+1} \le z_i + e_i||C||_2$ *for all* i.

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

Lemma 7.1. *Let* C *and* $C + E$ *be two matrices of full rank. Then*

$$||(C+E)^- - C^-||_2 \le ||(C+E)^- - C^-||_F \le 2||E||_F \max\{||C^-||_2^2,\ ||(C+E)^-||_2^2\}.$$

Proof. See [2, Section 5.5.5]. □

Corollary 7.1. *Assume that* $W_i = (C + \tilde{E}_i)^{-1}U_i = C^{-1}U_i + E_i$. *Then* $e_i \le \delta u_i$ *where* $\delta = \delta(C, \tilde{E}_i) = 2||\tilde{E}_i||_F \max\{||C^{-1}||_2^2, ||(C + \tilde{E}_i)^{-1}||_2^2\}$.

Assume that $z_i \le \gamma u_i$ for a constant γ, combine Theorem 7.5 and Corollary 7.1, and obtain that $u_{i+1} \le \theta u_i$ for $\theta = \delta||C||_2 + \gamma$ for all i. Recall that $U = U_0$ and summarize our estimates in the following corollary.

Corollary 7.2. *Under the above assumptions we have* $e_i \le \delta u_i \le \delta\theta^i||U||_2$ *for* $i = 1, 2, \dots, k$.

The corollary shows linear convergence of the error norms e_i to zero where $\theta < 1$. This basic inequality is realistic provided the matrix C is well conditioned, the norm $||C^{-1}||_2$ is not large, and the values γ and $||\tilde{E}||_F$ are small.

Finally, observe that $W = W_0 + \cdots + W_i$ and consequently $U = U_0 + \cdots + U_i$ and $G = I_r - V^H(W_0 + \cdots + W_i)$ if $e_i = 0$. Therefore, the linear convergence $e_i \to 0$ as $i \to \infty$ is extended to the linear convergence $W_0 + \cdots + W_i \to W$, and consequently, $U_0 + \cdots + U_i \to U$, $F_0 + \cdots + F_i \to F$, and $I_r + F_0 + \cdots + F_i = I_r - V^H(W_0 + \cdots + W_i) \to G$.

It remains to compute the products $F_i = -V^H W_i$ for $i = 0, \dots, k$ and the sum $G = I_r + \sum_{i=0}^{k} F_i$. At this stage, however, unlike the self-correcting computation of the matrices W_i, we cannot correct new errors and must just keep them small. Typically, for $r = \operatorname{nnul} A$, the norm $||G||_2$ is smaller than $||I_r||_2 = 1$ (see Theorem 7.4), and so we should expect a substantial or even dramatic loss of the leading bits in the representation of the entries of the matrix G at this stage of its computations. Fortunately, however, the floating-point subroutines from Section 12 are highly effective exactly for this task, and it remains to invoke them.

Moreover, as long as we fill the matrix V with sufficiently short binary numbers from a fixed range and compute the matrices W_i with single (or if needed even a shorter) precision, the double precision computation of the matrices $F_i = -V^H W_i$ is error-free. Then we only need to control the errors at the summation stage of computing the matrix $G = I_r + \sum_{i=0}^{k} F_i$, and we can achieve this by applying the floating-point subroutines from Section 12.

Let us finally estimate the overall number of iterative steps required in our computations. We guarantee the output values with the full relative precision by stopping in $O(\log(\epsilon \operatorname{cond}_2 A))$ invocations of the iterative refinement algorithm.

Overall, this means using $O(\log \mathrm{cond}_2 A - p)$ steps of iterative refinement where each step contributes p new correct bits per entry. For a $(0, r)$ matrix A, the entire computation requires $O(M_{A,r} \log \mathrm{cond}_2 A)$ single or double precision ops provided $M_{A,r}$ ops are sufficient to multiply the matrix A by an $n \times r$ matrix (see Section 2.1). The computational cost bound is low for smaller integers r and, if both matrices A and UV^H have the same structure, then also for larger integers r (see Section 7.8).

7.6 The dual and primal/dual Schur Aggregations

We devised the Schur aggregation for preconditioning $(0, h)$ matrices. Let us outline the dual extension to preconditioning $(g, 0)$ matrices.

Assume that the matrices A and $C_- = A^- + VU^H$ have full rank and deduce that the matrices $I_m + AVU^H$ and $I_n + VU^H A$ are nonsingular and that

$$(C_-)^- = (I_m + AVU^H)^{-1} A \quad \text{where} \quad m \geq n, \tag{7.7}$$

$$(C_-)^- = A(I_n + VU^H A)^{-1} \quad \text{where} \quad m \leq n. \tag{7.8}$$

By applying the primal SMW formula, obtain that

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

for $m \leq n$ and

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

for $m \geq n$. Substitute these equations into (7.7) and (7.8) and in both cases obtain the equation

$$(C_-)^- = (A^- + VU^H)^- = A - AV(I_r + U^H AV)^{-1} U^H A, \tag{7.9}$$

which we already displayed in equation (3.10). Equations (7.7) and (7.8) define factorization of a matrix A, whereas the inversion of the MPPs $I_m + AVU^H$ and $I_n + VU^H A$ is reduced to the inversion of the matrix $I_r + U^H AV$, which is the Schur complement of the block $-A^-$ in the block matrix $\begin{pmatrix} -A^- & U^H \\ V & I_r \end{pmatrix}$.

Note that for $m = n$ the computation of the solution $\mathbf{y} = A^- \mathbf{b} = C_- \mathbf{b} - UV^H \mathbf{b}$ to a linear system $A\mathbf{y} = \mathbf{b}$ can be reduced to computing the matrix $(C_-)^-$ and solving the linear system $(C_-)^- \mathbf{z} = \mathbf{b}$ for the vector $\mathbf{z} = C_- \mathbf{b}$.

Based on equation (7.9), we immediately devise a dual version of the Schur aggregation. In particular, the SVD-based expression for the Schur complement in Theorem 7.1 is preserved if we redefine the matrices U and V as $U = ((\frac{1}{\sigma} - \frac{1}{\sigma_j})\mathbf{s}_j)_{j=1}^g$ and $V = (t_j)_{j=1}^g$. Furthermore, the analysis and the recipes in Sections 7.4 and 7.5 are readily extended to the dual APPs and MPPs based on equation (7.9).

If the matrix C_- is well conditioned, then we can extend our analysis in the previous section to $r = g$ and the dual version of the Schur Aggregation.

For $m = n$, a nonsingular $(g, 0)$ matrix A, and a random or pseudo random, well conditioned, and consistently scaled APP VU^H of a rank of at least g, we expect that the matrices $C_- = A^- + VU^H$ and $I_r + U^H AV$ are well conditioned and the norm of the latter matrix is small relatively to the norm $||A^-||_2$ (see Theorem 7.4 for A replaced with A^- and UV^H with VU^H).

The computation of the Schur complement $I_r + U^H AV$ involves only matrix multiplications and the addition of the matrix I_r. We represent the matrix A as the sum $\sum_{i=1}^{k} A_i$ such that all entries of all matrices A_i are lower-precision numbers, perform all multiplications with double precision but with no errors, and finally apply the floating-point subroutines from Section 12 to obtain the matrix $I_r + U^H AV$ with negligible errors.

If the dual Schur complement is ill conditioned, we can apply the dual Schur Aggregation recursively. Likewise, we can recursively apply the dual Schur Aggregation to the matrix $(C_-)^-$ if, for our choice of the integer r, this is a $(g, 0)$ matrix for a positive g. We can reapply the same process, updating the integer r and the matrix $(C_-)^-$ recursively until we arrive at a well conditioned matrix $(C_-)^-$. This process avoids the pitfalls that we pointed out for the recursive construction of the matrix C in the primal aggregation. By choosing $r = 1$, we can always deal with a scalar dual Schur complement and thus simplify its inversion, although our overall asymptotic count of the single/double precision ops does not change.

Clearly, the primal and dual Schur Aggregation can be combined to treat (g, h) matrices A.

7.7 Summary on the Schur Aggregation for solving linear systems of equations

Let us summarize the Schur Aggregation for solving a linear system of equations $A\mathbf{y} = \mathbf{b}$ where we are given a pair of nonnegative integers g and h such that A is a (g, h) matrix. Afterwards we shall comment on computing such a pair of g and h. For the sake of completeness we include the SVD-based preconditioners, but we refer the reader to Section 4.1 on the benefits of using the SVD-free ones.

In Section 10 we describe three alternative algorithms based on the Null Aggregation for the same task of linear solving.

1. For any pair of g and h, we can apply the SVD-based MPPs (cf. Section 3.3) as long as the g-head and the h-tail of the SVD of the matrix A are available.

2. If $g = 0$, we can alternatively compute at first a crude approximation to the norm $||A||_2$, then a random or pseudo random well conditioned and consistently scaled APP UV^H of rank h (cf. Section 3.3) and the matrices $C = A + UV^H$ and $G = I_h - V^H C^- U$ expected to be well conditioned (see Sections 5 and 7.5), and finally, based on equation (7.5) for $r = h$, reduce the solution of the linear system $A\mathbf{y} = \mathbf{b}$ to solving some linear systems with these two matrices.

3. If $h = 0$, we can alternatively compute a crude approximation to the norm $||A^-||_2$, then compute a random or pseudo random well conditioned and

consistently scaled dual APP VU^H of rank g and the matrices $I_r + U^H AV$ and $(C_-)^-$ in equation (7.9) expected to be well conditioned (see Section 5 and the previous subsection), and finally employ equation (7.9) for $r = g$ to reduce the solution of the original linear system $A\mathbf{y} = \mathbf{b}$ to solving some linear systems with these two matrices.

4. For any pair of g and h, as long as the g-head of the SVD of the matrix A is available, we can combine recipes 1 for $h = 0$ and 2 as follows. First factorize the matrix A (cf. Section 3.3 for $h = 0$) to reduce the original linear system to a linear system with one of the matrices $\tilde{S}_{g,0}^H A$, $A\tilde{T}_{g,0}$, or $\tilde{S}_{g,0}^H A\tilde{T}_{g,0}$. Verify that this is a $(0, h)$ matrix (as expected) and then apply recipe 2 to it.

5. For any pair of g and h, as long as the h-tail of the SVD of the matrix A is available, we can combine recipes 1 for $g = 0$ and 3 as follows. First factorize the matrix A (cf. Section 3.3 for $g = 0$) to reduce the original linear system to linear systems with one of the matrices $\tilde{S}_{0,h}^H A$, $A\tilde{T}_{0,h}$, or $\tilde{S}_{0,h}^H A\tilde{T}_{0,h}$. Verify that this is a $(g, 0)$ matrix (as expected) and then apply recipe 3 to it.

6. For any pair of g and h, we can combine recipes 2 and 3 as follows. First apply recipe 2 to the input matrix A to compute an APP $\tilde{U}_h \tilde{V}_h^H$ of rank h and to reduce the original linear system of equations to linear systems with the coefficient matrix $\tilde{C} = A + \tilde{U}_h \tilde{V}_h^H$ and the respective $h \times h$ Schur complement. Verify that $\tilde{C}$ is a $(g, 0)$ matrix (as expected) and apply recipe 3 to solve the linear systems with this matrix.

We can apply the SVD algorithms in [2, Sections 9.1 and 9.2.9], [22, pages 366 and 367], [28], and in the bibliography therein to approximate the singular values of a matrix A and to compute the g-head and the h-tail of its SVD for fixed g and h. Even crude approximations to the singular values can suffice to yield smaller values of g and h for which A is a (g, h) matrix.

We can avoid computing the g-head and the h-tail if we apply recipes 2, 3, and 6 and incorporate our algorithms in Sections 4.4–4.5 for generating APCs, and we can alternatively compute the desired integers g and h as the minimum nonnegative integers for which the matrices $(C_-)^-$ in equation (7.9) and $C = A + UV^H$ are well conditioned, respectively, but we favor the computation of the integer parameters g and h via the application of the algorithms in Section 9.6, based on the Null Aggregation.

Our algorithms involve the condition and norm estimators, in particular to define consistent scaling for primal and dual APPs. One can apply the estimators in [2, Section 3.5.4] and [3, Sections 5.3]. (Dealing with the primal APCs UV^H and $\tilde{U}_h \tilde{V}_h^H$, we need a crude estimate for the norm $||A||_2$, whereas dealing with the dual APCs VU^H or $V_g U_g^H$, we need crude estimates for the 2-norms of the $(g, 0)$ matrix A^- or $\tilde{C}^- = (A + \tilde{U}_h \tilde{V}_h^H)^-$, expected to be a $(g, 0)$ matrix.)

All our recipes reduce the solution of the original linear system $A\mathbf{y} = \mathbf{b}$ to linear systems that are expected to be well conditioned and/or to have $r \times r$ coefficient matrices for $r = g$ or $r = h$. If the latter assumptions do not hold, we can reapply our techniques to these linear systems.

Using this approach with random or pseudo random APPs, we expect that performing the order of $(M_{A,g} + M_{A,h}) \log \mathrm{cond}_2 A$ ops with single or double precision is sufficient for the solution of a linear system with a (g, h) coefficient matrix A.

7.8 Preserving and exploiting matrix structure in the Schur Aggregation

To perform the Schur Aggregation with primal SVD-free APPs, we compute the matrices

1. $C = A + UV^H$,
2. $C^- U$ or $V^H C^-$,
3. $G = I_r + V^H C^- U$, and
4. G^{-1}

where we assume that the matrices A and C have full rank.

In the case of a larger rank r of the APP, our complexity estimates for the computations in the two previous subsections dramatically decrease if both APP and matrix A have the same displacement or rank structure. The decrease is by the factors of $n/\log^h n$ for an $n \times n$ matrix A and for h ranging from zero to two, depending on the structure.

There are two basic principles in this approach.

First, the operations at Stages 1–4 amount or can be reduced essentially to matrix multiplication and inversion, which we can perform economically by operating with the rank or displacement generators rather than the matrix entries (see [6, Chapters 1 and 4], [40]–[47], and the bibliography therein).

Secondly, as long as the matrix A has such a structure, we can extend it readily to the matrices involved in Stages 1–4 based on [6, Section 1.5] and our Lemma 2.2. This enables accelerated computations.

Our comments in this subsection can be also readily extended to the dual APPs. In Remark 9.4 in Part III we extend them to the Null Aggregation.

Remark 7.1. *Some degeneracy problems can arise at Stages 3 and 4 for matrices with Cauchy-like and Vandermonde-like structure, but we can avoid these problems by shifting to the computations with Toeplitz/Hankel-like structures. This is achieved readily with the* method of displacement transformation*, proposed in [41] (see its exposition also in [6, Sections 1.7, 4.8, and 4.9]). Its idea of transforming the displacement structure into the desired direction by using structured multipliers was made popular in [43], [60], where the more general class of Vandermonde multipliers proposed in [41] was narrowed to its effective special case of the Fourier transform multipliers. The resulting transform of the structures of the Toeplitz and Hankel types into the structures of the Cauchy and Vandermonde types supports pivoting and/or diagonal scaling. To yield structured Schur complements, however, we should move into the opposite direction,*

from the Cauchy and Vandermonde structures to the Toeplitz/Hankel structures. Here the Fourier multipliers are not generally sufficient, but one can apply the original Vandermonde multipliers from [41].

8 Computation of determinants and resultants

8.1 Computing determinants: the problem and two approaches to its solution

The sign of the determinant of a real $n \times n$ matrix A is required in some of the most fundamental geometric and algebraic computations, whereas the elimination methods for solving polynomial systems of equations rely on computing the determinants of the associated Newton's and resultant structured matrices (see [13]–[15], and the bibliography therein).

In the present day computing environment, numerical algorithms usually run faster than symbolic algorithms but can fail to produce correct output due to rounding errors. *Arithmetic filtering* combines both symbolic and numerical methods to perform the computations both faster and correctly. One begins with numerical algorithms, which rapidly compute the certified correct outputs on most of the input instances. In the rare cases the algorithms fail, and then one relays the task to symbolic methods, which are slower but reliable. Frequently the failure of numerical algorithms implies a smaller upper bound on $|\det A|$. Such a bound facilitates the application of symbolic methods to a matrix A filled with integers [13], [16].

We briefly review some symbolic methods for computing the determinants in Section 8.6, and we cover some numerical methods next.

8.2 Rounding errors and the certification of the output

Recall that we can readily compute the sign and the value of $\det A$ based on factorizations of the matrix A, in particular on its LU or QR factorizations (with or without pivoting) or its SVD $A = S\Sigma T^H$ because $\det A = (\det B) \det C$ if $A = BC$ and because the determinants of diagonal, triangular, and unitary matrices can be computed readily.

By employing a rational version of the modified Gram–Schmidt algorithm in [61], one can avoid computing the square roots and thus reduce the bad impact of rounding errors at the price of some moderate slowdown of the computations.

Whenever we compute the sign numerically, we can certify it if $\sigma_n = \sigma_n(A)$, the smallest singular value of the matrix, exceeds an upper estimate $e(A)$ for the norm of its factorization error [16]. We have a priori estimates $e(A) \le c\sigma_1\epsilon$ for such errors in LU and QR factorizations [3], [57]. Here ϵ is the unit roundoff and c represents small positive constants. A posteriori estimates are reduced to simpler bounds on the error norm for matrix-by-matrix or matrix-by-vector

multiplication [3, Section 2.1], [57, Section 3.5]. They involve smaller overhead constants and also cover computation of the determinants based on the SVDs.

Next we examine numerical computation of the determinants, both SVD-based (employing Theorem 3.2) and SVD-free (employing equations (7.1)–(7.9)). The SVD-based computation is vulnerable to the errors of computing the SVD, whereas our SVD-free algorithms are free also from this shortcoming.

8.3 SVD-based computation of the signs and the values of determinants

We first reuse the matrix factorizations in Theorem 3.2. Recall that $\det A = (\det B)\det C$ if $A = BC$ as well as if $A = \mathrm{diag}(A, B)$ and obtain that

$$\det A = \sigma^{g+h} \det F_l / ((\det \tilde{S}_{g,h}) \widehat{\prod}_j \sigma_j),$$

$$\det A = \sigma^{g+h} \det F'_r / ((\det \tilde{T}_{g,h}) \widehat{\prod}_j \sigma_j),$$

$$\det A = \sigma^{g+h} \det F_{l,r} / ((\det \tilde{S}_{g,h})(\det \tilde{T}_{g,h}) \widehat{\prod}_j \sigma_j).$$

Here the symbol $\widehat{\prod}_j$ stands for the product in j ranging from 1 to g and from $n-h+1$ to n, and we assume the definitions of Theorem 3.2 for $r = l = 0$ and $m = n = \rho$, so that the matrices $\tilde{S}_{g,h}$ and $\tilde{T}_{g,h}$ are unitary, $\sigma_{n-h} \le \sigma \le \sigma_g$, and the matrices $F_l = F_r$ and $F_{l,r}$ are better conditioned than a (g,h) matrix A. This enables us to simplify the computation of $\det A$ for a (g,h) matrix A given with the g-head and h-tail of its SVD.

The computation of the sign of the determinant can be simplified further. Namely, by combining our expressions for $\det A$ with equations (3.2)–(3.4) and noting that $\sigma_j > 0$ for $0 < j < \rho$, $\det \Sigma^{(g)} > 0$, and $\det \Sigma_h > 0$, we arrive at the equations

$$\mathrm{sign}\det A = (\mathrm{sign}\det \begin{pmatrix} T^{(g)H} \\ S^{(ORT)H}_{g,h} A \\ T^H_h \end{pmatrix}) \mathrm{sign}\det \tilde{S}_{g,h},$$

$$\mathrm{sign}\det A = (\mathrm{sign}\det(S^{(g)}, AT^{(ORT)}_{g,h}, S_h)) \mathrm{sign}\det \tilde{T}_{g,h},$$

$$\mathrm{sign}\det A = (\mathrm{sign}\det(S^{(ORT)H}_{g,h} AT^{(ORT)}_{g,h}))(\mathrm{sign}\det \tilde{S}_{g,h}) \mathrm{sign}\det \tilde{T}_{g,h}$$

where the unitary matrices $S^{(ORT)H}_{g,h}$ and $T^{(ORT)H}_{g,h}$ are defined in Section 3.2.

The three latter equations hold as long as the rounding errors of computing the matrices $\tilde{S}_{g,h}$, $\tilde{T}_{g,h}$, $\begin{pmatrix} T^{(g)H} \\ S^{(ORT)H}_{g,h} A \\ T^H_h \end{pmatrix}$), $(S^{(g)}, AT^{(ORT)}_{g,h}, S_h)$, and $S^{(ORT)H}_{g,h} AT^{(ORT)}_{g,h}$ do not exceed their smallest singular values. These values equal one for the unitary matrices $\tilde{S}_{g,h}$ and $\tilde{T}_{g,h}$ and equal $\sigma_{n-h}(A)$ for the three other matrices, whose norms equal $\sigma_g(A)$ provided $\sigma_{n-h}(A) \le 1 \le \sigma_g(A)$.

8.4 APC-based factorizations for the determinants

Based on the factorization $\det A = (\det C)\det(I_r - V^H C^{-1} U)$ in (2.3), we reduce the determinant computation to computing

a) an APC UV^H of a smaller rank r of the order of $\operatorname{nnul} A$ that defines a better conditioned matrix C,

b) the matrix $I_r - V^H C^{-} U$ with a high precision, and

c) the determinants of the matrices C and $I_r - V^H C^{-1} U$.

For a $(0, h)$ matrix A and for $h \le r < n$, task c) is simpler than the original task because the matrix C is better conditioned and the matrix $I_r - V^H C^{-1} U$ has a smaller size. If we need, we can recursively apply the same algorithm to the matrices C and $I_r - V^H C^{-1} U$.

For task a), we can either apply Algorithm 4.1 numerically (see Section 4.4) or employ the APPs of Examples 4.1–4.6. We can further refine the output APC UV^H according to the recipes to be given in Section 9.6. Furthermore, as we pointed out earlier, we can truncate the entries of the matrices U and V to represent them with a lower precision to compute the matrix C with no errors.

For task b), we can apply our variant of the iterative refinement and the floating-point subroutines from Section 12. If we seek the sign of the determinant we can safely stop wherever the rounding errors in computing the matrix are smaller than its smallest singular value.

If we are given (close approximations to) the h-tail of the SVD, then for task a) we can apply Algorithm 3.2 for $\sigma_{n-h} \le \sigma \le \sigma_1$ to compute an APC UV^H where the matrices U and V are defined by equations (3.8) and where we have $\operatorname{cond}_2 C \le \sigma_1/\sigma_{n-h}$ for $C = A + UV^H$ (cf. (3.9)), whereas for task c) we can employ the factorization

$$\det(I - V^H C^{-1} U) = \sigma^{-h} \prod_{j=1}^{n-h+1} \sigma_j,$$

implied by Theorem 7.1 for $m = n = \rho$ and $g = 0$. The computations are simpler than with the primal APPs of the same size.

Seeking $\det A$ for a nonsingular $(g, 0)$ matrix A, we can rely on the following dual version of factorization (2.3),

$$\det A = (\det H)\ \det((C_-)^{-1}) \tag{8.1}$$

where $H = I_r + U^H A V$ is the Schur complement of the nonsingular block $C_- = A^{-1} + VU^H$ in the block matrix $\begin{pmatrix} -C_- & V \\ U^H & I_r \end{pmatrix}$ and where $(C_-)^{-1} = A - AVH^{-1}U^H A$ due to equation (7.9). The computations are simpler than with the primal APPs of the same size.

Seeking the determinant of a (g, h) matrix, we can employ both primal and dual factorizations in (2.3) and (8.1).

8.5 Numerical tests for computing the signs of the determinants

We computed the signs of the determinants of nonsingular matrices $A = PML$ where P denoted permutation matrices, each swapping k random pairs of the rows of the matrix A, whereas L and M^T denoted random $n \times n$ lower triangular matrices with unit diagonal entries and with integer subdiagonal entries randomly sampled from the line intervals $[-\gamma, \gamma]$ for a fixed positive γ. It follows that $\det A = (-1)^k$. We generated such matrices for $k = 2n$ and $k = 2n - 1$ and for $\gamma \geq 5,000$.

We applied both Matlab's numerical subroutine and our algorithm based on factorization (2.3).

To generate random APPs in our algorithm, we first fixed a positive integer r, then generated two random $n \times r$ unitary matrices, truncated their entries to represent them with the precision of 20 bits, and thus arrived at two matrices $\tilde{U}$ and $\tilde{V}$. Then we computed the matrix $\tilde{U}\tilde{V}^H$ of rank r and finally applied the binary pseudo optimal scaling to yield an APP $UV^H = 2^d\tilde{U}\tilde{V}^H$ for an integer d such that $1/2 < \theta = ||UV^H||_2/||A||_2 \leq 2$ (cf. Section 5.3).

The Matlab's numerical subroutines performed poorly for the matrices of the selected class. They failed the competition in accuracy not only to the symbolic slower subroutines in MAPLE but also to our numerical tests. Already for $n = 4$ and $\gamma = 5,000$, the Matlab's 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 99% of the runs, whereas our algorithms produced correct output in over 99% of the runs for substantially larger n and γ.

We specify the results of these and further tests in our next papers.

8.6 Symbolic computation of determinants

Recently, substantial attention was paid by the researchers to estimating the randomized bit-operation (Boolean) complexity of computing matrix determinants. The smallest randomized bit-operation bounds were obtained based on symbolic algorithms that rely on solving linear systems and Hensel's lifting [62, Appendix], [63]–[65], on the block Lanczos–Wiedemann method [2, Section 9.2.6], [15], [66]–[74] (which has incorporated many advanced techniques during its long and intensive study by many researchers), and on high order Newton's lifting [75], [76].

These methods generate some random parameters and in addition perform $(n^d(\log ||A||_2)^{d_1})^{1+o(1)}$ bit operations for an $n \times n$ input matrix A filled with integers. The algorithm in [65] supports the bound $d_1 = 1.5$ versus $d_1 = 1$ in the other papers. The table below shows the exponents d, which can be decreased by incorporating asymptotically fast but practically infeasible algorithms for matrix multiplication and multivariate polynomial gcd (see some details in [72], [73], and the bibliography therein).

Table: the exponents of the bit-complexity

of computing matrix determinants

d	4	7/2	10/3	16/5	3
algorithm	classical	[65]	[70]	[71]	[76]

This table is our tribute to the challenge in the paper [70], which has focused on presenting its randomized exponents $d = 10/3$ and $d_1 = 1$ and devoted the main Theorem 2 essentially just to stating these exponents. These exponents are important but surely are not the only criteria and even are not the most important ones, however. Actually, the best use of the symbolic algorithms for the sign and/or the value of the determinants is within the cited arithmetic filtering approach, and various other symbolic algorithms (e.g., in [13], [14], and the references therein and in [16]) only support the (deterministic) exponent $d = 4$ but remain practically superior to the above symbolic algorithms for moderate and even moderately large n.

Let us, however, give a more detailed comparison of the three cited classes of asymptotically fast randomized symbolic algorithms with each other. The algorithm in [65] supports the best exponent $d = 3$ on the average input but, unlike the other algorithms, does not certify the output. The block Lanzcos–Wiedemann's algorithms in [70]–[73] are superior to the algorithm in [65] but inferior to the Storjohann's algorithm in exploiting the power of block matrix computations (see, e.g., [2, Sections 1.3 and 1.4] on this power). Unlike the other algorithms cited above, the block Lanzcos–Wiedemann's algorithms run faster by the factor of f for the (structured and/or sparse) matrices A that can be multiplied by a vector in $2n^2/f$ ops.

The latter advantage is practically important but of course only where the algorithms are practically feasible themselves. This requirement, however, is in conflict with the incorporation of the multivariate LKS half-gcd algorithm into the bottleneck stage of the determinant algorithm in [70], that is, the stage of the solution of an auxiliary block Toeplitz/Hankel linear system. The problem was resolved in [71] and [73] where the infeasible half-gcd step in [70] was replaced by the application of Hensel's lifting. The resulting determinant algorithm is feasible and supports the exponent 16/5.

One can further refine this algorithm by incorporating and refining generalized Hensel's lifting (proposed and elaborated upon in [77]–[80]) and by applying the lifting algorithms directly to block Toeplitz/Hankel linear systems, versus their auxiliary representation as Toeplitz/Hankel-like linear systems in [73].

8.7 Extension to the resultants and polynomial systems of equations

In Section 1.2 we recalled that the roots of a system of multivariate polynomial equations can be approximated via solving multi-level Toeplitz linear systems of equations, and we proposed to use APPs for the solution. As an alternative approach, we can approximate these roots as the roots of the determinant of the associated resultant or Newton's structured matrices. Then the problem

is reduced to solving a system of equations in fewer variables, possibly in a single variable [15, Sections 5–7]. With due caution to the numerical stability problems, we can combine the known iterative root-finders (e.g., Muller's or Newton's) with the above recipe for computing the resultant and with the partial derivative theorem. To support Newton's method we can apply the equation $\frac{d(A^{-1})}{dx} = -A^{-1}\frac{dA}{dx}A^{-1}$ to the resultant matrix $A = A(x)$.

The known rounding error estimates for computing determinants (based on the Hadamard's bound), however, are overly pessimistic, particularly where the determinants vanish. Numerically it can be more effective to annihilate the smallest singular value of the resultant or Newton's matrix or the absolutely smallest diagonal entry in the R factor in its QR factorization with pivoting.

PART III. NULL AGGREGATION, LINEAR SYSTEM SOLVING AND EIGENSYSTEMS

With our APPs in Part II we simplified the solution of ill conditioned linear systems of equations and the computation of determinants. We relied on matrix factorization and the Schur aggregation, which confined ill conditioning to the Schur complemements typically of smaller sizes. Now we are going to rely on computing null vectors and null bases and thus to confine ill conditioning to the search for the numerical nullity of the input matrix via a bounded number of condition estimations. We call these processes the *Null Aggregation* (cf. Remark 9.2). Besides the computation of null vectors in Sections 9.1–9.4 , we apply our algorithms to approximating the trailing singular subspaces of an ill conditioned matrix in Section 9.5, refining APCs in Section 9.6, and solving linear systems of equations in Section 10. Furthermore, we incorporate these algorithms into the inverse iteration for the algebraic eigenproblem in Section 11.

9 Computing null vectors and null bases with A-preconditioning and the Null Aggregation

One can obtain null matrix bases for an $m \times n$ matrix A of a rank ρ via computing its SVD, QRP, or PLU factorizations [2], [3], [12], [27] (cf., e.g., effective Algorithm 5.3.2 in [2]) or via computing a nonsingular and well conditioned $\rho \times \rho$ submatrix W of matrices A or MAN for some nonsingular multipliers M and N.

Randomized A-preconditioning gives us an alternative. Namely, for two matrices U of size $m \times r$ and V of size $n \times r$, suppose that an APP UV^H has rank $r = \operatorname{rnul} A$ and the matrix $C = A + UV^H$ has full rank. Then $C^{-}U$ is a null matrix basis for the matrix A (see Theorem 9.1). Furthermore, if A is a $(0, h)$ matrix and if the APP is random or pseudo random, well conditioned, consistently scaled, and has rank $h \geq \operatorname{nnul} A$, then the matrix C is likely to be well conditioned, according to our analysis in Section 5 and tests in Section 6.

Thus if $r = \operatorname{rnul} A$ is known, the computation of a null matrix basis essentially amounts to solving r well conditioned linear systems of equations. We specify the details in Algorithm 9.1.

We can apply Algorithm 4.1 for randomized computation of $\operatorname{nul} A$ and a desired APP (cf. Remark 4.1), and we can alternatively choose pseudo random ACs (see Examples 4.1–4.6).

Our algorithms in Section 9.1 compute some null matrix bases and, therefore, the rank and the nullity as by-products. In Sections 9.2 and 9.3 we cover computations in the left null space and the reduction to a square input, respectively. In Section 9.4 we simplify our algorithms where we seek a single null vector. In Sections 9.1–9.4 we assume error-free computation. In Section 9.5 we comment on an extension to numerical approximation of the trailing singular spaces of ill conditioned matrices. In Section 9.6 we apply our results on computing null bases to yield APCs of the optimal rank.

9.1 Null bases via A-preconditioning

We compute null bases based on a theorem that we next sketch and then state and prove formally. In our sketch we write $C = A + UV^H$, $C_1 = A + U_1V_1^H$, and $r = \operatorname{rank}(UV^H)$ and let "(nmb)" stand for "null matrix basis", "$\Longrightarrow$" for"implies", and "$\Longleftrightarrow$" for "if and only if". Assuming that A and C are $m \times n$ matrices and $\operatorname{rank} C = n \le m$, we have

$$N(A) \subseteq \operatorname{range}(C^-U),$$

$$\{r = \operatorname{nul} A\} \Longleftrightarrow \{N(A) = \operatorname{range}(C^-U)\} \Longleftrightarrow \{AC^-U = 0\},$$

$$\{X = (\text{nmb})(AC^-U)\} \Longrightarrow \{\operatorname{range}(C^-UX) = N(A)\}.$$

Theorem 9.1. *Suppose $m \ge n$ and for an $m \times n$ matrix A of a rank ρ and a pair of two matrices U of size $m \times r$ and V of size $n \times r$, the matrix $C = A + UV^H$ has full rank n. Then*

$$r \ge \operatorname{rank} U \ge n - \rho = \operatorname{rnul} A = \operatorname{nul} A, \tag{9.1}$$

$$N(A) \subseteq \operatorname{range}(C^-U). \tag{9.2}$$

Furthermore if

$$r = \operatorname{rank} U = n - \rho = \operatorname{rnul} A = \operatorname{nul} A, \tag{9.3}$$

then we have

$$N(A) = \operatorname{range}(C^-U), \tag{9.4}$$

$$V^HC^-U = I_r. \tag{9.5}$$

Proof. Bound (9.1) follows from Theorem 4.1a. If $\mathbf{y} \in N(A)$, then $C\mathbf{y} = (A + UV^H)\mathbf{y} = UV^H\mathbf{y}$, and therefore (cf. equation (2.1)),

$$\mathbf{y} = C^-U(V^H\mathbf{y}). \tag{9.6}$$

This proves (9.2).

(9.4) immediately follows from (9.2) and (9.3).

To prove (9.5), pre-multiply equation (9.6) by V^H, recall equation (9.4), and deduce that $(V^HC^{-}U - I_r)V^HC^{-}U = 0$. Now (9.5) follows unless the matrix $V^HC^{-}U$ is singular, but if it is, then $V^HC^{-}U\mathbf{z} = \mathbf{0}$ for some nonzero vector $\mathbf{z}$. Let us write $\mathbf{w} = C^{-}U\mathbf{z}$, so that $V^H\mathbf{w} = \mathbf{0}$ and $\mathbf{w} \in \text{range}(C^{-}U) = N(A)$. It follows that $A\mathbf{w} = \mathbf{0}$, and therefore, $C\mathbf{w} = A\mathbf{w} + UV^H\mathbf{w} = \mathbf{0}$. Since the matrix C has full rank, it follows that $\mathbf{w} = \mathbf{0}$. Consequently, $\mathbf{z} = \mathbf{0}$ because the matrix $C^{-}U$ has full rank. □

Theorem 9.1 implies that the range of the matrix $C^{-}U$ always contains the null space $N(A)$ (see (9.2)) and is equal to it if $r = \text{rank}(UV^H) = \text{nul}\, A$ (see (9.4)). Our algorithms for computing null bases in this subsection and their numerical versions in Section 9.5 rely on these observations.

The algorithms employ black box Subroutines LIN·SOLVE and **AC**.

Subroutines LIN·SOLVE are applied to an $m \times h$ matrix B and an $m \times n$ matrix A. They output FAILURE if the matrix A is rank deficient. Otherwise they compute an $n \times h$ matrix Y satisfying the matrix equation $AY = B$ or output INCONSISTENT if the equation has no solution.

Subroutines **AC** have an $m \times n$ matrix A and a nonnegative integer $r(+)$ as their input. They fail if $r(+) < \text{nul}\, A$. If $r(+) \geq \text{nul}\, A$, some of them never fail, whereas the other (randomized) algorithms may fail but with a low probability. Unless they fail, they output an AC of a rank r such that $\text{nul}\, A \leq r \leq r(+)$. Our Algorithms 3.2 and 4.1 can serve as Subroutines **AC**.

Correctness of our first two algorithms follows from equations (9.3) and (9.4). In the first algorithm we assume the value $\text{nul}\, A = \text{rnul}\, A$ known, generate an APP UV^H of rank $r = \text{nul}\, A$, and then immediately compute a null matrix basis $C^{-}U$ for $C = A + UV^H$. As we pointed out earlier, we can search for $r = \text{nul}\, A$ by computing APPs whose ranks recursively increase from zero and testing where C becomes a full rank matrix (cf. Algorithm 4.1 and Remark 4.1), but in our second algorithm we begin with an upper bound $r(+)$ on the nullity and recursively generate APPs UV^H whose rank r decreases from $r(+)$ to $\text{nul}\, A$. We recognize the moment when $r = \text{nul}\, A$ by observing that $AC^{-}U = 0$.

Algorithm 9.1. A null basis where the nullity is known.

INPUT: *two integers m and n such that $m \geq n > 0$, a normalized $m \times n$ matrix A, an integer $\nu = \text{rnul}\, A = \text{nul}\, A$, and Subroutines* **AC** *and LIN·SOLVE.*

OUTPUT: *FAILURE or an $n \times \nu$ unitary matrix basis Y for the null space $N(A)$.*

COMPUTATIONS:

1. *Apply Subroutine* **AC** *to the input pair of A and $r(+) = \nu$. Output FAILURE if the Subroutine* **AC** *has output FAILURE.*
2. *Otherwise apply Subroutine LIN·SOLVE to compute the matrix $C^{-}U$.*
3. *Compute and output the Q-factor Y of the matrix $C^{-}U$.*

Algorithm 9.2. A null basis with certification via annihilation.

INPUT: *two integers m and n such that $m \geq n > 0$, a normalized $m \times n$ matrix A, an integer $r(+) \geq \operatorname{rnul} A = \operatorname{nul} A$, and Subroutines* **AC** *and LIN·SOLVE.*

OUTPUT: *FAILURE or an integer $\nu = \operatorname{nul} A$ and an $n \times \nu$ unitary matrix basis Y for the null space $N(A)$.*

COMPUTATIONS:

1. Apply Subroutine **AC** *to the input pair of A and $r(+)$. Output FAILURE if so does the Subroutine* **AC**.

2. Otherwise apply Subroutine LIN·SOLVE to compute the matrix $C^{-}U$.

3. If $AC^{-}U = 0$, output the integer $\nu = r$. Compute and output the Q-factor Y of the matrix $C^{-}U$.

4. Otherwise $r(+) \leftarrow r - 1$ and reapply the algorithm.

We can apply the binary search for the value $r = \operatorname{nul} A$ because $r \geq \operatorname{nul} A$ if the matrix C has full rank, and if so, then $r > \operatorname{nul} A$ unless $AC^{-}U = 0$.

Next we propose three alternative algorithms. First we deduce from equations (9.1) and (9.2) that $N(A) = \operatorname{range}(C^{-}UX)$ provided X is a null matrix basis for the $m \times r$ matrix $AC^{-}U$. This implies correctness of the following algorithm. (see Remark

Algorithm 9.3. A null basis via aggregation *(see Remark 9.2).*

INPUT, OUTPUT, *and Stages 1 and 2 of* COMPUTATIONS *as in Algorithm 9.2.*

COMPUTATIONS:

3. Compute a unitary matrix basis X for the null space $N(AC^{-}U)$ (see Remark 9.2) and output the number ν of its columns.

4. Compute and output the Q-factor Y of the matrix $C^{-}UX$.

Our next algorithm relies on the fact that the ranges of the matrices $C^{-}U$ and $C_1^{-}U_1$ share the null space $N(A)$ for any pair of ACs UV^H and $U_1V_1^H$ (cf. (9.2)) and share nothing else for properly chosen ACs. We are guided by the following implication of Theorem 9.1,

$$\{\begin{pmatrix} X \\ Z \end{pmatrix} = (\text{nmb})(C^{-}U, C_1^{-}U_1) \text{ for random } U,\ V,\ U_1, \text{ and } V_1$$

$$\text{and for } X \text{ of full rank}\} \Longrightarrow$$

$$\{\operatorname{range}(C^{-}UX) = N(A) \text{ (likely)}\}.$$

Algorithm 9.4. A null basis via space intersection *(cf. Remarks 9.1 and 9.2).*

INPUT, OUTPUT, *and Stages 1–3 of* COMPUTATIONS *as in Algorithm 9.2.*

COMPUTATIONS:

4. Otherwise choose $r_1(+) \geq \operatorname{nul} A$ *such that* $r_1(+) + r \leq m$ *and apply Stages 1–3 of Algorithm 9.2 to the pair of* A *and* $r_1(+)$ *to output a positive integer* r_1 *such that* $\operatorname{nul} A \leq r_1 \leq r_1(+)$*, a pair of unitary matrices* U_1 *of size* $m \times r_1$ *and* V_1 *of size* $n \times r_1$*, and an* AC $U_1 V_1^H$ *of rank* r_1*. In this application, output FAILURE if* $r + r_1 > m$*. Otherwise choose the matrix* U_1 *such that*

$$(\operatorname{range} U_1) \bigcap \operatorname{range}(AC^- U) = \{\mathbf{0}\}. \tag{9.7}$$

(This equation is ensured if, say $U_1^H AC^- U = 0$*, and furthermore, is likely to hold for a random matrix* U_1 *if* $r + r_1 \leq m$ *because the ranges of the matrices* U_1 *and* $AC^- U$ *have dimensions* r_1 *and* r*, respectively.) Write* $C_1 = A + U_1 V_1^H$*.*

5. Compute an integer $q \geq \operatorname{nul} A$*, an* $r \times q$ *matrix* X *of full rank (e.g., unitary), and* $r_1 \times q$ *matrix* Z *such that the* $((r + r_1) \times q)$ *matrix* $\begin{pmatrix} X \\ Z \end{pmatrix}$ *is a null matrix basis for the matrix* $(C^- U, C_1^{-1} U_1)$ *(cf. Remarks 9.1 and 9.2).*

6. Compute the Q-factor Y *(of size* $m \times \nu$*) for the* $m \times q$ *matrix* $C^- UX$*. Output the matrix* Y *and the integer* ν*.*

To prove correctness of the algorithm, first observe that

$$L = \operatorname{range}(C^- UX) = \operatorname{range}(C^- U) \bigcap \operatorname{range}(C_1^- U_1).$$

Therefore, $N(A) \subseteq L$ (cf. (9.1) and (9.2)). Now let $\mathbf{y} \in L \subseteq \operatorname{range}(C_1^- U_1)$. Then $C_1 \mathbf{y} \in \operatorname{range} U_1$. Therefore, $A\mathbf{y} \in \operatorname{range} U_1$ since $C_1 \mathbf{y} = A\mathbf{y} + U_1 V_1^H \mathbf{y}$. Simultaneously we have $A\mathbf{y} \in AL \subseteq \operatorname{range}(AC^- U)$. Consequently, $A\mathbf{y} = \mathbf{0}$ due to assumption (9.7). Therefore, $N(A) = L = \operatorname{range}(C^- UX)$, and the correctness is proved.

The algorithm reduces the null basis computation for an $m \times n$ matrix A to the same problem for the $n \times (r + r_1)$ matrix $(C^- U, C_1^- U_1)$ and consequently, for the RP-factor in its QRP factorization.

For smaller integers $r(+)$ and general matrix A, the most costly stage in Algorithms 9.2–9.4 (in terms of ops involved) is the computation of the matrix $C^- U$ at Stage 2 (together with the equally costly computation of the matrix $C_1^- U_1$ at Stage 5). If $m = n$ we can compute the r columns of the matrix $C^{-1} U$ independently of each other by using $n^3 + O(rn^2)$ ops. If we first invert the matrix C and then multiply the inverse by the matrix U, then we need $2n^3 + O(rn^2)$ ops overall. In both cases the cubic terms cover the cost of solving linear systems with well conditioned matrices C, which supports the transition from the input matrix A to smaller size matrices.

We immediately observe that equation (9.7) is likely to hold even under the random choice of the matrix U_1 at Stage 4 of Algorithm 9.4. The unlikely case where the equation fails to hold can be detected indirectly, by testing whether

range$(C^-U) \bigcap$range$(C_1^-U_1) \neq N(A)$. If we detect this, we can generate a new APP $U_1V_1^H$ and repeat our computations. This leads us to the following modification of Algorithm 9.4.

Algorithm 9.5. A null basis via recursive space intersection *(cf. Remarks 9.1 and 9.2).*

INPUT, OUTPUT, *and Stages 1–5 of* COMPUTATIONS *as in Algorithm 9.4 except that the matrix U_1 is random, and equation (9.7) is not enforced at Stage 4 of the computations anymore.*

COMPUTATIONS:

6. Compute the $m \times q$ matrix C^-UX of a rank $s \leq q$. If $AC^-UX = 0$, then compute and output the Q-factor Y (of size $m \times \nu$) for the matrix C^-UX and output the integer ν.

7. Otherwise $C^-U \leftarrow C^-UX$ and reapply Stages 4–7.

Remark 9.1. *For $m = n$ the matrix C_1 is nonsingular, and so the matrices $(C^-U, C_1^-U_1)$ and (C_1C^-U, U_1) share their null space. Therefore, seeking matrix bases at Stage 4 of Algorithms 9.4 and 9.5, we can avoid the computation of the matrices C_1^- and $C_1^-U_1$ and thus save some ops.*

Remark 9.2. *Algorithm 9.3 reduces the computation of a null basis for an $m \times n$ matrix A to the same computation for the $m \times r$ matrix AC^-U. For $r < n$ this is an Aggregation Process, the matrix AC^-U is an aggregate of the input matrix A, and the computation of a null basis in Stage 2 is disaggregation (cf. [17]). For $r + r_1 < n$ Algorithm 9.4 is another aggregation process for the same problem reducing it to the case of the $n \times (r + r_1)$ input matrix $(C^-U, C_1^{-1}U_1)$, and we can similarly interpret Algorithm 9.5. These aggregation processes can be recursively applied to auxiliary matrices of smaller sizes. We call all these processes the* Null Aggregation *(cf. the Schur Aggregation in Section 7.2 and the Aggregation Processes in [17]).*

9.2 Left null bases via A-preconditioning

We can apply our previous study to the matrix A^H to compute its right nullity and a unitary right null basis, which gives us the integer lnul A and a left null basis for the matrix A. If $m = n$, the same factorization of the matrix C can be employed for computing both left and right null bases of the matrix A. Equation (9.5) remains valid. Here are the dual counter-parts of equations (9.1)–(9.2) and (9.3)–(9.4), respectively:

$$r \geq \text{rank}\, V \geq \text{lnul}\, A,\ LN(A) \subseteq \text{range}(V^HC^-), \tag{9.8}$$

$$r = \text{rank}\, V = \text{lnul}\, A,\ LN(A) = \text{range}(V^HC^-). \tag{9.9}$$

9.3 Transition from rectangular to square inputs

Computations with a rectangular $m \times n$ input matrix A can be readily reduced to the case of larger but square inputs.

If $m < n$ we can compute $n \times n$ matrices $\tilde{A} = MA$ for M equal to A^H or to an $n \times m$ matrix of full rank m, e.g., a fixed or random unitary matrix M. In both cases $N(A) = N(\tilde{A})$. If $M = A^H$, then $\operatorname{cond}_2 \tilde{A} = (\operatorname{cond}_2 A)^2$, but the matrices $\tilde{A} = MA = A^H A$ and $\tilde{C} = \tilde{A} + UU^H$ are Hermitian nonnegative definite, so that the matrix $\tilde{C}$ is positive definite if it is nonsingular.

For $M = \begin{pmatrix} 0 \\ I_m \end{pmatrix}$ and $M = \begin{pmatrix} I_m \\ 0 \end{pmatrix}$ we have $\tilde{A} = \begin{pmatrix} 0 \\ A \end{pmatrix}$ and $\tilde{A} = \begin{pmatrix} A \\ 0 \end{pmatrix}$, respectively. Here we lose symmetry but add no errors in the transition $A \leftarrow \tilde{A}$.

If $m > n$, then we can simply append the $m - n$ null column vectors to the matrix A to turn it into an $m \times m$ matrix $\tilde{A}$. Null vectors of the matrices A and $\tilde{A}$ are immediately recovered from each other.

Finally, we can apply 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 $m = n$.

9.4 Computing null vectors with A-preconditioning

If we only need to compute a normalized null vector $\mathbf{y}$ of a matrix A, we can simplify our Algorithms 9.1–9.5 as follows.

a) Change Stages 3 in Algorithm 9.1 as follows.

3. Compute and output the vector $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$ for a normalized vector $\mathbf{x}$.

b) Change Stages 3 and 4 in Algorithm 9.3 as follows.

3. Compute a normalized vector $\mathbf{x}$ satisfying the equation $AC^{-}U\mathbf{x} = \mathbf{0}$.
4. Compute and output the vector $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$.

c) Change Stage 3 in Algorithms 9.2, 9.4, and 9.5 as follows.

3. If there exists a normalized solution $\mathbf{x}$ to the vector equation $AC^{-}U\mathbf{x} = \mathbf{0}$, then compute such a solution and compute and output the vector $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$.

d) Change Stages 5 and 6 in Algorithm 9.4 as follows.

5. Compute a solution $\mathbf{w} = \begin{pmatrix} \mathbf{x} \\ \mathbf{z} \end{pmatrix}$ of dimension $r + r_1$ to the matrix equation $(C^{-}U, C_1^{-}U_1)\mathbf{w} = \mathbf{0}$ where the vectors $\mathbf{x}$ and $\mathbf{z}$ have dimensions r and r_1, respectively, and the vector $\mathbf{x}$ is normalized.
6. Compute and output the vector $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$.

e) Initialize Algorithm 9.5 with setting $i \leftarrow 1$ and change its Stages 4–7 as follows.

4. Otherwise choose $r_i(+) \geq \operatorname{nul} A$ such that $r_i(+) + r \leq m$ and apply Stages 1–3 of Algorithm 9.2 to the pair $(A, r_i(+))$. This computation produces a positive integer r_i such that $\operatorname{nul} A \leq r_i \leq r_i(+)$, two unitary matrices U_i of size $m \times r_i$ and V_i of size $n \times r_i$, and an AC $U_i V_i^H$ of rank r_i. Write $C_i = A + U_i V_i^H$.

5. Compute a normalized vector $\mathbf{x}$ and vectors $\mathbf{z}_1, \dots, \mathbf{z}_i$ such that

$$(C^- U, C_j^- U_j) \begin{pmatrix} \mathbf{x} \\ \mathbf{z}_j \end{pmatrix} = \mathbf{0}, \ j = 1, \dots, i.$$

6. If $AC^- U\mathbf{x} = \mathbf{0}$, then compute and output the vector $\mathbf{y} = C^- U\mathbf{x}/||C^- U\mathbf{x}||_2$.

7. Otherwise $i \leftarrow i + 1$ and reapply Stages 4–6.

Remark 9.3. *We can extend our modified algorithms to the computation of a null basis for an $m \times n$ matrix A without reverting back to the original algorithms. Compute a normalized (right) null vector $\mathbf{y}$, a normalized left null vector $\mathbf{z}$ for the matrix A, and $\sigma \approx \sigma_1(A)$, set $A \leftarrow A + \sigma \mathbf{y}\mathbf{z}^H$, and then reapply the modified algorithms. Repeat recursively. Stop where the current matrix A has full rank and thus has no left and/or right null vectors. The computed vectors $\mathbf{y}$ form a unitary null basis for the original matrix A. Their number equals the nullity.*

9.5 Numerical computations of null bases and null vectors

Numerical versions of our null basis and null vector algorithms are closely related to the approximation of trailing singular spaces of ill conditioned matrices. Assume that we are given an $(0, h)$ ill-conditioned matrix A of full rank ρ where $h = \operatorname{nnul} A$. Set to zero its h smallest singular values to arrive at a rank deficient matrix $A - E$ with $h = \operatorname{nul} A$ where the norm $\delta = ||E||_F^2 = \sum_{j=\rho-h+1}^{\rho} \sigma_j^2$ is small. The known perturbation bounds for singular spaces (cf. [22, Section 4.2.3], in particular [22, Theorem 4.2.13] for $M = 0$) imply that the null space $N(A - E)$ closely approximates the respective trailing singular space of the matrix A. One can readily refine such approximations by applying the inverse iteration (cf. Section 11).

Next, towards numerical implementation of our null basis and null vector algorithms, we first modify our sketches of Theorem 9.1 and its implication supporting Algorithm 9.4. We keep assuming that A and C are $m \times n$ matrices for $m \geq n$ and keep writing r for $\operatorname{rank}(UV^H)$, q for $n \leq m$, "(nmb)" for "null matrix basis", "$\Longrightarrow$" for "implies", and "$\Longleftrightarrow$" for "if and only if", but instead of the matrices A, $C = A + UV^H$, and $C_1 = A + U_1 V_1^H$, we deal with their approximations $\tilde{A} = A - E$, $\tilde{C} = C - E$, and $\tilde{C}_1 = C_1 - E$, respectively, and we write "nnul", "nrank", and "(τmb)" (for a fixed positive tolerance τ) instead of "nul", "rank", and "nmb", respectively. We say that one linear space approximates another if the largest canonical angle between these spaces is less than a fixed small positive θ (see the definition of the angle in [22, Section

4.2.1]). Now assume that $\operatorname{nrank} \tilde{C} = q$ and extend our sketches of Theorem 9.1 and its implication supporting Algorithm 9.4 as follows,

$$N(\tilde{A}) \approx \text{a subspace of } \operatorname{range}(\tilde{C}^- U),$$

$$\{r = \operatorname{nnul} \tilde{A}\} \Longleftrightarrow \{N(\tilde{A}) \approx \operatorname{range}(\tilde{C}^- U)\} \Longleftrightarrow \{||\tilde{A}\tilde{C}^- U||_2 \approx 0\},$$

$$\{\tilde{X} = (\tau\text{mb})(\tilde{A}\tilde{C}^- U)\} \Longrightarrow \{\operatorname{range}(\tilde{C}^- U X) \approx N(\tilde{A})\},$$

$$\{\begin{pmatrix} \tilde{X} \\ \tilde{Z} \end{pmatrix} = (\tau\text{mb})(\tilde{C}^- U, \tilde{C}_1^- U_1) \text{ for random unitary } U,\ V,\ U_1, \text{ and } V_1$$

$$\text{and for well conditioned } \tilde{X}\} \Longrightarrow$$

$$\{\operatorname{range}(\tilde{C}^- U\tilde{X}) \approx N(\tilde{A}) \text{ (likely)}\}.$$

To implement and to analyze our algorithms in Section 9.1 and the previous subsection numerically, one should elaborate upon these sketches. In particular, one should specify the tolerance values τ and θ, quantify the concepts "nnul", "nrank", and "likely", and incorporate the above modifications of Theorem 9.1 into the algorithms by applying the customary techniques such as the techniques for the least-squares solution of linear systems and for the norm and condition estimation [2], [3], [12], [27]. In the present paper we only initialize this study by stating three simple theorems.

The first of them extends the inclusion (9.2) numerically.

In the second theorem the matrix $(C - E)^- U$ is an $(\text{nmb})(A - E)$ and $\mathbf{y} = C^- U\mathbf{x}$ for a normalized vector $\mathbf{x}$. The theorem expresses the tolerance τ in terms of $\delta = ||E||_F$ and $\sigma_- = 1/||C^-||_2$ such that $\mathbf{y}$ is a (right) τ-*vector* for the matrix A, that is, such that $||A\mathbf{y}||_2 \le \tau ||A||_2 ||\mathbf{y}||_2$ for a fixed positive τ. (We can also define the left τ-vectors for a matrix A as the τ-vectors for the matrix A^H.) In this theorem we use $\delta = ||E||_F$ versus $\delta = ||E||_2$ in Theorem 5.4.

The third theorem bounds the leading coordinates in the linear expressions of τ-vectors via singular vectors.

Theorem 9.2. *For a matrix $C = A + UV^H$ of full rank and any vector $\mathbf{y}$ we have $\min_{\mathbf{x}} ||\mathbf{y} - C^- U\mathbf{x}||_2 \le ||C^- A\mathbf{y}||_2$.*

Proof. Recall that $C\mathbf{y} = A\mathbf{y} + UV^H\mathbf{y}$ and obtain that $\mathbf{y} - \mathbf{z} = C^- A\mathbf{y}$ where $\mathbf{z} = C^- UV^H\mathbf{y} \in \operatorname{range}(C^- U)$. □

In the proof of our next theorem we use Lemma 7.1 and the following lemma.

Lemma 9.1. *For two matrices C and E of the same size where the matrix C is nonsingular and $||C^{-1}E||_2 = \theta < 1$, we have $||I - (C - E)^{-1}C||_2 \le \frac{\theta}{1-\theta}$.*

Proof. See [3, Theorem 1.4.18] for $P = C^{-1}E$. □

Theorem 9.3. *For positive integers m, n, and r where $m \ge n$, a pair of $m \times n$ matrices A and E, and a pair of unitary matrices U of size $m \times r$ and V of size $n \times r$, assume that the matrices $C = A + UV^H$ and $C - E$ have full rank, $r = \operatorname{nul}(A - E)$, $||A||_2 = 1$, $\sigma_- = 1/||C^-||_2 > \delta = ||E||_F$, and $\mathbf{y} = C^- U\mathbf{x}$*

for a normalized vector $\mathbf{x}$. *Then* $\mathbf{y}$ *is a* τ*-vector of the matrix* A *where* $\tau \le (1+(4+4\delta)/((\sigma_- - \delta)^2\sigma_-))\delta$ *(for* $m \ge n$*) and* $\tau \le (1+(1+\delta)/(\sigma_- - \delta))\delta$ *for* $m = n$.

Proof. We have $(A-E)(C-E)^- U\mathbf{x} = \mathbf{0}$ in virtue of Theorem 9.1 (cf. (9.4)). Therefore, $A\mathbf{y} = AC^- U\mathbf{x} = EC^- U\mathbf{x} + \mathbf{z} = E\mathbf{y} + \mathbf{z}$ where $\mathbf{z} = (A-E)(C^- - (C-E)^-)U\mathbf{x}$, so that $||\mathbf{z}||_2 \le ||A-E||_2||(C^- - (C-E)^-)||_2 \le (1+\delta)||(C^- - (C-E)^-)||_2)$ (for $m \ge n$) and $||\mathbf{z}||_2 \le ||A-E||_2||(I-(C-E)^- C)||_2||\mathbf{y}||_2$ for $m = n$. Now the theorem follows from the bounds $||\mathbf{y}||_2 \ge 1/||C^-||_2 = \sigma_-$ (this bound holds because the $m \times n$ matrix C has full rank and because $m \ge n$), $||E||_2 \le ||E||_F = \delta$, and $||A-E||_2 \le 1+\delta$ and from Lemmas 7.1 and 9.1. □

Theorem 9.4. *Let* $A = S\Sigma T^H$ *be the SVD of a normalized* $m \times n$ *matrix. Then we have* $\sigma_g ||T^{(g)H}\mathbf{u}||_2 \le \tau$ *for a normalized* τ*-vector* $\mathbf{u}$.

Proof. Let $\mathbf{u} = \sum_{j=1}^n u_j \mathbf{t}_j$ be a normalized (right) τ-vector of the matrix A. Then we have $A\mathbf{u} = \sum_{j=1}^n u_j \sigma_j \mathbf{s}_j$, $T^{(g)H}\mathbf{u} = (u_j)_{j=1}^g$, and therefore, $||A\mathbf{u}||_2^2 = \sum_j |u_j^2|\sigma_j^2 \le \tau^2$, $\sigma_g^2 ||T^{(g)H}\mathbf{u}||_2^2 = \sigma_g^2 \sum_{j=1}^g |u_j|^2 \le \sum_{j=1}^g |u_j|^2 \sigma_j^2 \le \tau^2$. □

Remark 9.4. *Our study in Section 7.8 is readily extended to the Null Aggregation as long as we can choose matrices* U *and* V *whose structure is consistent with the structure of the input matrix* A. *Indeed the transition to the matrices* $C^- U$ *and* $V^H C^-$ *preserves the matrix structure. Thus we arrive at some structured null matrix bases for a structured rank deficient matrix* A. *For a structured ill conditioned matrix* A *of full rank, we arrive at structured matrices* $C^- U$ *and* $V^H C^-$ *that approximate matrix bases for the trailing singular spaces of the matrix* A. *Although these spaces may have no structured matrix bases, we obtain structured null matrix bases* $C^- U$ *and* $V^H C^-$ *for a nearby unstructured singular matrix, which itself we do not compute. The very existence of such matrix bases* $C^- U$ *and* $V^H C^-$ *for a structured ill conditioned matrix* A *can be of independent interest.*

9.6 Improving APCs based on the null basis computations

Let us next apply Theorem 9.1 to support policies (1.3) and (1.4) of computing ACs for singular square matrices and APCs for ill conditioned nonsingular matrices. Our argument can be extended to computing primal and dual ACs and APCs for rectangular matrices. By replacing the transposes with Hermitian transposes and the inverses with generalized inverses, we rewrite these policies as follows,

$$(U \leftarrow Q(C^- U), \quad V \leftarrow Q(C^{-H} V)), \tag{9.10}$$

$$(V \leftarrow Q((C_-)^- V), \quad U \leftarrow Q((C_-)^{-H} U)). \tag{9.11}$$

Here M^{-H} denotes $(M^H)^- = (M^-)^H$ and, as before, $Q(M)$ denotes the Q-factor in the QR factorization of a matrix M.

For simplicity we state the respective theorem for square matrices A.

Theorem 9.5. *Let A be a normalized $n \times n$ matrix of a rank $\rho < n$ and let U and V be a pair of $n \times r$ unitary matrices such that $r = n - \rho = \operatorname{nul} A$ and the matrix $C = A + UV^H$ is nonsingular. Let U_1 and V_1 denote the respective updates of the matrices U and V according to policy (9.10). Then the matrix $A + U_1 V_1^H$ is nonsingular and* $\operatorname{cond}_2(A + U_1 V_1^H) = \operatorname{cond}_2 A$.

Proof. Due to Theorem 9.1, the updated matrices U_1 and V_1 are the right and left null matrix bases of the matrix A, respectively. Let $A = \sum_{j=1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H$ be the SVD of the matrix A. Write $U_1 = (\mathbf{u}_j)_{j=1}^r$ and $V_1 = (\mathbf{v}_j)_{j=1}^r$ and obtain the SVD of the matrix $A + U_1 V_1^H = \sum_{j=1}^{r} \mathbf{u}_j \mathbf{v}_j^H + \sum_{j=1}^{\rho} \sigma_j \mathbf{s}_j \mathbf{t}_j^H$. Theorem 9.5 follows because $r = n - \rho$ and $\sigma_1 = 1$. □

Now we can extend Theorem 9.5 to support approach (9.10) for a nonsingular normalized ill conditioned $(0, r)$ matrix $\tilde{A}$. Obtain a singular well conditioned matrix A by truncating the $r = \operatorname{nnul} \tilde{A}$ smallest singular values of the matrix $\tilde{A}$ and then compute an AC UV^H for the matrix A based on policy (9.10) and Theorem 9.5. Clearly, this AC can serve as an APC for the matrix $\tilde{A}$ because the truncated r-tail of the SVD of the matrix $\tilde{A}$ has a small norm since $r = \operatorname{nnul} \tilde{A}$, and so shifting back to the matrix $\tilde{A} + UV^H$ by adding this tail to the well conditioned matrix $A + UV^H$ little changes its singular values.

A similar policy by X. Wang in [18] produces the APC $U_1 V_1^H = ||A||_2 VQ(UP)^H$ of rank h where $P\Sigma_1 P^H$ and $Q\Sigma_2 Q^H$ are the h-heads of the SVDs of the matrices $(AU)^H AU$ and $(A^H V)A^H V$, respectively. X. Wang successfully tested this policy for 10×10 Hilbert matrices $(\frac{1}{i+j-1})_{i,j=1}^{10}$ and various choices of h and $r < 2h$ (see [18]).

Random or pseudo random APPs UV^H whose rank exceeds $\operatorname{nnul} A$ is a safe initial choice for obtaining a well conditioned matrix $C = A + UV^H$ according to our extensive tests. The above recipes complement this choice by yielding APCs of the optimal rank, thus supporting our construction in Section 7.5.

Similar comments apply to our policy (9.11) for dual APCs VU^H and its following combination with policy (9.10),

$$\tilde{U}_h \leftarrow Q(C^- \tilde{U}_h),\ \tilde{V}_h^H \leftarrow Q(\tilde{V}^H C^-),$$

$$V_g \leftarrow Q((\tilde{C}_-)^- V_g),\ U_g^H \leftarrow Q(U_g^H (\tilde{C}_-)^-),$$

for a pair of primal/dual APPs $\tilde{U}_h \tilde{V}_h^H$ and $V_g U_g^H$ and the output matrix $\widehat{C}_- = \tilde{C}^- + V_g U_g^H$ where $\tilde{C} = A + \tilde{U}_h \tilde{V}_h^H$.

10 Alternative methods for linear systems

10.1 Reduction to null vector computations

In Section 7 we employed the Schur aggregation to facilitate matrix factorization and the solution of a matrix equation $AY = B$, for an $m \times n$ matrix A and $m \times h$ matrix B. In our alternative approach in this subsection we reduce the solution

to null space computations, namely, to the equivalent task of computing an $(n+h)\times h$ matrix $Z=\begin{pmatrix} I_h \\ Y \end{pmatrix}\in N(\widehat{A})$ where $\widehat{A}=(-B,A)$ is an $m\times(n+h)$ matrix. (Conversely, the null space computation is a special case of solving the matrix equation $AY=B$ where $B=0$.) For this task we can adjust our algorithms in Sections 9.1 and 9.4, preceded by application of the recipes in Section 9.3 if $(n+h)>m$.

Let us specify the solution assuming that the input includes Subroutines LIN·SOLVE, defined in Section 9.1, and NULL·VECTORS that have an $m\times n$ matrix A and an integer $r(+)\ge \operatorname{nul} A$ as the input and that output a unitary null matrix basis Y for the matrix A. We devised such subroutines in the previous section.

Algorithm 10.1. Solutions to linear systems as null vectors.

INPUT: *an $m\times h$ normalized matrix B, an $m\times n$ normalized matrix A, and black box Subroutines* **AP**, *LIN·SOLVE, and NULL·VECTORS.*

OUTPUT: *an $n\times h$ matrix Y satisfying the matrix equation $AY=B$ or INCONSISTENT if the equation has no solution.*

COMPUTATIONS:

1. Apply Subroutine LIN·SOLVE to compute and output the matrix Y. If this works, stop. Otherwise apply the Subroutine NULL·VECTORS to the pair $(r(+),\widehat{A})$ of a fixed integer $r(+)\ge\nu=\operatorname{nul}\widehat{A}\ge h$ and the $m\times(n+h)$ matrix $\widehat{A}=(-B,A)$ (rather than A) to compute an $(n+h)\times h$ unitary null matrix basis $\widehat{Y}$ for the matrix A. (We can choose any integer $r(+)$ which would support generating a pair of unitary matrices $\widehat{U}$ of size $m\times r(+)$ and $\widehat{V}$ of size $(n+h)\times r(+)$ such that the matrix $\widehat{C}=\widehat{A}+\widehat{U}\widehat{V}^H$ has full rank.)

2. Write $\widehat{Y}=\begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}$ where Y_0 and Y_1 are $h\times\nu$ and $n\times\nu$ matrices, respectively. If the matrix Y_0 is rank deficient, output INCONSISTENT. Otherwise apply Subroutine LIN·SOLVE to compute and output the $n\times h$ matrix $Y=Y_1Y_0^-$ satisfying the matrix equation $Y_0Y_0^-=(I_h,0)$.

To prove correctness of the algorithm, first note that the consistency of the matrix equation $AY=B$ is equivalent to the inclusion $\operatorname{range}Z\subseteq\operatorname{range}\widehat{Y}$ for some matrix $Z=\begin{pmatrix} I_h \\ Y \end{pmatrix}\in N(\widehat{A})$. Therefore, the equation $AY=B$ is consistent if and only if the matrix Y_0 has full rank h, and in this case we have

$$\operatorname{range}Y_0=\operatorname{range}I_h. \tag{10.1}$$

It remains to show that $AY=B$ for $Y=Y_1Y_0^-$. From the equation $(-B,A)\widehat{Y}=0$ deduce that $(-B,A)\widehat{Y}Y_0^-=0$, whereas $\widehat{Y}Y_0^-=\begin{pmatrix} I_h \\ Y_1Y_0^- \end{pmatrix}$. Therefore, $AY=B$ for $Y=Y_1Y_0^-$, and this completes the correctness proof.

For smaller h the complexity of Algorithm 10.1 is dominated by application of the Subroutine NULL·VECTORS at Stage 1. For $h = 1$ the matrix equation $AY = B$ turns into a single linear system of n equations with n unknowns. For any $n \times h$ matrix B we can reduce the computation to solving h linear systems $A\mathbf{y}_i = B\mathbf{e}_i$ for $i = 1, \dots, h$, so that $Y = (\mathbf{y}_i)_{i=1}^{h}$.

Remark 10.1. *In its numerical implementation Algorithm 10.1 can fail at Stage 2 if the matrix Y_0 is nonsingular but ill conditioned. This cannot occur for $h = 1$ because in this case Y_0 is a scalar. For a larger h we can solve separately the h linear systems $A\mathbf{y}_i = \mathbf{b}_i$ for $i = 1, \dots, h$ where $Y = (\mathbf{y}_i)_{i=1}^{h}$ and $B = (\mathbf{b}_i)_{i=1}^{h}$. Alternatively we can move the problem to Stage 1 by requiring at that stage that $\widehat{Y} = \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}$ where $Y_0 = (Y_{0,0}, Y_{0,1})$ and $Y_{0,0}$ is an $h \times h$ unitary matrix. Then at Stage 2 we would readily compute Y_0^{-} based on equation (2.2).*

10.2 The primal and dual Null SMW expressions for solving linear systems

By applying the SMW formula for the inverse of the matrix $A = C - UV^H$, express the solution to a nonsingular linear system $A\mathbf{y} = \mathbf{b}$ as

$$\mathbf{y} = C^{-1}(\mathbf{b} + U\mathbf{x}) \tag{10.2}$$

where UV^T is an APC of our choice such that the matrix $C = A + UV^H$ is nonsingular and well conditioned and $\mathbf{x}$ is a vector satisfying the linear system

$$AC^{-1}(\mathbf{b} + U\mathbf{x}) = \mathbf{b}. \tag{10.3}$$

We call the latter equations the *primal Null SMW expression*. The columns of the matrix $AC^{-1}U$ are the aggregates of the input matrix A. If $\operatorname{rank}(UV^H) = \operatorname{nnul} A$, then the matrix $AC^{-1}U$ has a small norm and must be computed with a high precision to support the computation of the vector $\mathbf{x}$. At this stage we can again employ the variant of the iterative refinement in Section 7.5 and the algorithms from Section 12 to compute the latter aggregates with a small relative error norms. Then we obtain the vectors $\mathbf{x}$ and $\mathbf{y}$ from equations (10.2) and (10.3).

The analysis and the resulting ops count for these single or double precision numerical computations are similar to the case of the primal Schur Aggregation in Section 7.5. Versus Section 7.5, we have greater need for the segmentation techniques from the end of Section 12 for reducing pre-multiplication by the input matrix A to low precision multiplications and summation. Indeed, in Section 7.5, at the respective stage of pre-multiplication by the matrix V^H, we had the entries of this matrix alreday represented with a shorter precision, but now we ought to apply segmentation to the entries of the matrix A to achieve this effect.

By applying the dual SMW formula, we obtain the following *dual Null SMW expression*, which complements the expressions (10.2) and (10.3): a vector $\mathbf{y} =$

$\mathbf{z} - VU^H\mathbf{b}$ satisfies the linear system $A\mathbf{y} = \mathbf{b}$ if $(C_-)^{-1}\mathbf{z} = \mathbf{b}$ for the matrix $(C_-)^{-1}$ in (7.9). This expression is proposed for ill-conditioned linear systems $A\mathbf{y} = \mathbf{b}$ where the matrix $A^{-1} + VU^H$ is well conditioned and two matrices U and V have smaller sizes or have structure consistent with the structure of the matrix A.

To alleviate the size problem, one can apply the dual Null Aggregation recursively, choosing a pair of random or pseudo random vectors as the matrices U and V in each recursive step and stopping where this process produces a well conditioned matrix $(C_-)^{-1}$.

Finally, one can readily combine the primal and dual Null Aggregation for a linear system with a (g, h) matrix. This seems to be particularly appropriate for smaller positive g and h.

11 APPs, eigenspaces, and the inverse iteration

We begin with definitions in the next subsection and brief theoretical analysis of the impact of A-preprocessing on the eigensystems in Sections 11.2 and 11.3. Then in Sections 11.4–11.9 we apply A-preconditioning and the Null Aggregation to the inverse iteration.

11.1 Matrix polynomials and the algebraic eigenproblem

The relevant definitions from Section 2 can be extended from matrices to matrix polynomials $A(\lambda) = \sum_{i=0}^{m} A_i\lambda^i$ where $A_0, \dots, A_m$ are matrices of the same size.

The eigenvalues of a matrix polynomial $A(\lambda)$ of a positive degree m are the roots of the characteristic polynomial $c_A(\lambda) = \det A(\lambda)$. The eigenvalues of a scalar matrix A are the eigenvalues of the linear matrix polynomial $A(\lambda) = \lambda I - A$. For simplicity the reader may wish to deal just with this classical case.

The (algebraic) multiplicity $m(\mu)$ of an eigenvalue μ of $A(\lambda)$ is the multiplicity of the root μ of the polynomial $c_A(\lambda)$.

An eigenvalue μ of $A(\lambda)$ is associated with the left and right eigenspaces $LN(A(\mu))$ and $N(A(\mu))$ made up by its associated left and right eigenvectors, respectively. It has left and right geometric multiplicities $l.g.m._A(\mu) = \mathrm{lnul}\, A(\mu)$ and $r.g.m._A(\mu) = \mathrm{rnul}\, A(\mu)$, respectively.

To a fixed set $\{\lambda_1, \dots, \lambda_h\}$ of the eigenvalues of $A(\lambda)$ we associate the left and right invariant eigenspaces $LN(A)$ and $N(A)$ of the matrix $A = \prod_{i=1}^{h} A(\lambda_i)$.

11.2 The affect of A-preprocessing on the eigensystem

Let us examine some impacts of A-preprocessing on the eigensystem.

Theorem 9.1 implies some rational characteristic equations for the eigenvalues of a matrix polynomial $A = A(\lambda)$. Suppose that $g.m._A(\lambda) = r$ for a fixed value of λ, U and V are $n \times r$ matrix polynomials in λ, and $C = A + UV^H$. Then matrix equation (9.5) turns into the system of r^2 rational equations

$$F(\lambda) = I_r - V^H C^{-1} U = 0_r \tag{11.1}$$

satisfied by the eigenvalues λ with $g.m._A(\lambda) = r$. By pre- and post-multiplying matrix equation (11.1) by vectors $\mathbf{s}^H$ and $\mathbf{t}$ of dimension r, respectively, we obtain a single scalar equation in λ,

$$f(\lambda) = \mathbf{s}^H F(\lambda)\mathbf{t} = \mathbf{s}^H\mathbf{t} - \mathbf{s}^H V^H C^{-1} U\mathbf{t} = 0.$$

By applying equation (2.3) for $A = A(\lambda)$ and $C = C(\lambda)$, so that $\det A = \det A(\lambda) = c_A(\lambda)$ and $\det C = \det C(\lambda) = c_C(\lambda)$, we obtain the following equations in λ independent of the multiplicity $g.m._A(\lambda)$.

$$c_A(\lambda) = c_C(\lambda)\det(I_r - V^H C^{-1} U).$$

Based on this equation, one can apply rank-one or small-rank modifications of a tridiagonal Hermitian matrix A to devise effective divide-and-conquer algorithms for approximating its eigenvalues and eigenvectors [2, Section 8.5.4], [22, Section 3.2]. We refer the reader to [81] on some serious difficulties with the extension of this approach to the non-Hermitian eigenproblem.

Remark 11.1. *For $C = \lambda I_r - M$ the matrix in equation (11.1) is a special case of the expressions $B - V^H(\lambda I - M)^{-1}U$, called the* realizations of rational matrix functions*, naturally interpreted as Schur complements and extensively used in linear systems and control [82].*

Finally let us estimate the impact of random A-preprocessing on the geometric multiplicity of the eigenvalues.

Theorem 11.1. *Suppose that $A = A(\lambda)$, $U = U(\lambda)$, and $V = V(\lambda)$ denote three matrix polynomials of sizes $n \times n$, $n \times r$, and $n \times r$, respectively. Fix a scalar λ, write $C = A + UV^H$ and $h = g.m._A(\lambda)$ so that $h \le n$, and suppose that $r \le h$. Then*

a) $g.m._C(\lambda) \ge h - r$ and

b) $g.m._C(\lambda) = h - r$ with a probability of at least $1 - 2r/|\Sigma|$ if the $(m+n)r$ entries of the matrices U and V have been randomly sampled from a set Σ of cardinality $|\Sigma|$.

Proof. Part a) is immediate. Now write $\rho = \operatorname{rank} A$ and $q = \rho + r$ and suppose that $q < n$ and A_q is a $q \times q$ submatrix of the matrix A such that $\operatorname{rank} A_q = \operatorname{rank} A = \rho$. Clearly, we can readily choose the matrices U and V such that the respective $q \times q$ submatrix $C_q = A + UV^H$ of the matrix C is nonsingular. Part b) follows from Lemma 2.1 because $\det C_q$ is a nonzero polynomial of degree of at most $2r$ in the entries of the matrices U and V. □

It follows that randomized A-preprocessing of a rank r is likely to decrease the geometric multiplicity of a multiple eigenvalue λ by $\min\{r,\ g.m._A(\lambda)\}$, and we should expect similar impact on the clusters of the eigenvalues. For Hermitian matrices the eigenvalues are also the singular values, and so random APPs are likely to decompress a compressed singular spectrum.

It is also likely, however, that the approximation of an eigenvalue λ of multiplicity $h > 1$ for a nonderogatory matrix A can be simplified if we apply a

random APP UV^H of rank $r = h - 1$ to obtain the matrix $C = A + UV^H$. Indeed, in virtue of Theorem 11.1, we can expect that the value $g.m._C(\lambda)$ is equal to one.

11.3 APPs and the eigenvectors: preliminary observations

The eigenspaces of a matrix polynomial $A(\lambda)$ associated with its eigenvalue $\lambda = \mu$ are precisely the null spaces $LN(A(\mu))$ and $RN(A(\mu))$. Therefore, the algorithms in Section 9 can be applied to the respective eigen-computations, and Theorem 9.1 enables us to express the eigenvectors associated with the eigenvalue μ through linear systems of equations with the matrices $C(\mu) = A(\mu) + U(\mu)V(\mu)^H$. Let us specify these expressions in the simpler case of a rank-one modification of a diagonalizable matrix polynomial $A(\lambda)$.

Theorem 11.2. *Suppose that* $A(\lambda) = G_A D_A(\lambda) G_A^{-1}$, $D_A(\lambda) = \mathrm{diag}(p_{A,i}(\lambda)(\lambda - \lambda_{A,i}))_{i=1}^n$, *$G_A$ is a nonsingular $n \times n$ matrix, $p_{A,i}(\lambda)$, for $i = 1, \dots, n$, are scalar polynomials in λ, $\mathbf{u}(\lambda)$ and $\mathbf{v}(\lambda)$ are a pair of n-dimensional vectors or vector polynomials in λ, and $C(\lambda) = A(\lambda) + \mathbf{u}(\lambda)\mathbf{v}^H(\lambda)$ such that the matrices $C = C(\lambda_{A,i})$, for $i = 1, \dots, n$, are nonsingular. Write $\mathbf{u} = \mathbf{u}(\lambda_{A,i})$ and $\mathbf{v} = \mathbf{v}(\lambda_{A,i})$, for $i = 1, \dots, n$. Then $\lambda = \lambda_{A,i}$, for $i = 1, \dots, n$, are the eigenvalues of the matrix polynomial $A(\lambda)$ and the vectors $G_A\mathbf{e}_i = (\mathbf{v}^H G_A \mathbf{e}_i) C^{-1}\mathbf{u}$ and $\mathbf{e}_i^H G_A^{-1} = (\mathbf{e}_i^H G_A^{-1}\mathbf{u})\mathbf{v}^H C^{-1}$ are their associated eigenvectors.*

Proof. The first equation is just equation (9.6) for $\mathbf{y} = G_A\mathbf{e}_i$, $U = \mathbf{u}$ and $V = \mathbf{v}$. The second equation is obtained similarly. □

The eigenvectors associated with a fixed eigenvalue of a matrix or a matrix polynomial are the solutions of some homogeneous singular linear systems of equations. Theorem 11.2 and the algorithms in Section 9 enable us to compute these vectors by solving nonsingular and sufficiently well conditioned linear systems of equations. In the next subsections we incorporate this computation to refine some popular eigen-solvers.

11.4 Inverse iteration and our modifications: an overview

The solution of an ill conditioned linear system of equations is the basic operation in some popular eigen-solvers such as the inverse power iteration, the Jacobi–Davidson algorithm, and the Arnoldi and Lanczos algorithms with the shift-and-invert enhancements. The same operation is encountered at the deflation stage of the QR algorithm. As we could have seen already, scaled random small-rank modifications are likely to improve the conditioning of such linear systems. Next we exemplify and analyze this approach for the inverse power iteration, which is a classical tool for the refinement of a crude solution to the algebraic eigenproblem [2], [22], [28], [83]–[85] and has a more recent block version,

called the inverse orthogonal iteration [2, page 339] and the inverse Rayleigh–Ritz subspace iteration [22, Section 6.1]. We use the respective abbreviations IPI and IR–RI.

Somewhat counter-intuitively, the IPI produces an accurate eigenvector as the solution of an ill conditioned linear systems of equations. This is not completely painless, however. In [28] the exposition of the inverse power iteration is concluded with the following sentence: "... inverse iteration does require a factorization of the matrix $A - \delta I$, making it less attractive when this factorization is expensive." Our next goal is to counter this deficiency by applying A-preconditioning (and involving neither M-preconditioning nor the Schur aggregation).

Recall that for a matrix polynomial $A(\lambda) = \sum_{i=0}^{m} A_i \lambda^i$, we define its eigenpairs (λ, Y) such that λ is a scalar, $\det A(\lambda) = 0$, and Y is a unitary matrix basis for the null space $N(A(\lambda))$.

Given a close approximation $\tilde{\lambda}$ to a geometrically simple eigenvalue λ of $A(\lambda)$ and a generally crude normalized approximation $\tilde{\mathbf{y}}$ to an associated eigenvector $\mathbf{y}$, the IPI recursively alternates updatings of the scalar $\tilde{\lambda}$ and the vector $\tilde{\mathbf{y}}$ according to the mappings $\tilde{\lambda} \leftarrow$ (the Rayleigh quotient $\tilde{\mathbf{y}} A(\tilde{\lambda}) \tilde{\mathbf{y}}$) and $\tilde{\mathbf{y}} \leftarrow A^{-1}(\tilde{\lambda})\tilde{\mathbf{y}} / ||A^{-1}(\tilde{\lambda})\tilde{\mathbf{y}}||_2$. The process stops where a fixed tolerance value exceeds the Rayleigh quotient.

For $\tilde{\lambda} \approx \lambda$ the matrix $A(\tilde{\lambda})$ is ill conditioned, but we can reduce updating the vector $\tilde{\mathbf{y}}$ to solving a linear system $C(\tilde{\lambda})\tilde{\mathbf{y}} = \mathbf{u}$ with a preconditioned coefficient matrix $C(\tilde{\lambda}) = A(\tilde{\lambda}) + \mathbf{u}\mathbf{v}^H$. Here the APP is given by a pair of random normalized vectors $\mathbf{u}$ and $\mathbf{v}$ (or of pseudo random vectors chosen, e.g., based on Examples 4.1–4.6), and due to our study in Section 5 and test results in Section 6, we expect that the matrix polynomial $C(\tilde{\lambda})$ is better conditioned than $A(\tilde{\lambda})$. Otherwise we stay with essentially the same iterative process and readily extend our study of their convergence and arithmetic cost. Indeed, if $\tilde{\lambda} \approx \lambda$, then the vector $C^{-1}(\tilde{\lambda})\mathbf{u}$ is close to a vector $\mathbf{y} \in N(A)$. This can be deduced from equation (9.4) or from the the equation $C(\lambda)\mathbf{y} = b\mathbf{u}$ for $\mathbf{y} \in N(A)$ and $b = \mathbf{v}^H\mathbf{y}$.

Describing the resulting algorithm below, we write $||\cdot||_q$ for $q = 2$ or $q = F$ to denote the 2-norm or the Frobenius norm of a matrix, write $\mathbf{y}$ instead of $\mathbf{u}$, and recursively update the vector $\mathbf{y}$ by over-writing it with the vector $C^{-1}(\tilde{\lambda})\mathbf{y}$ where $C(\tilde{\lambda}) = A(\tilde{\lambda})$ if the matrix $A(\tilde{\lambda})$ is well conditioned and $C(\tilde{\lambda}) = A(\tilde{\lambda}) + \mathbf{y}\mathbf{v}^H$ otherwise.

11.5 Inverse iteration with APPs of rank one

Algorithm 11.1. Inverse iteration with APPs of rank one.

INPUT: *a matrix polynomial $A(\lambda)$, an approximation $\tilde{\lambda}$ to its eigenvalue λ, two positive values τ and k, $q = 2$ or $q = F$, and Subroutine LIN·SOLVE for solving a nonsingular linear system of equations.*

OUTPUT: *either FAILURE, or PROBABLY G·MULTIPLE, or an approximation $(\lambda_{final}, \mathbf{y}_{final})$ to an eigenpair of $A(\lambda)$ such that $||A(\lambda_{final})\mathbf{y}_{final}||_2 \leq$*

$\tau||A(\lambda_{final})||_q$.

INITIALIZATION: *Set* $COUNTER \longleftarrow 0$, $\sigma \longleftarrow ||A(\tilde{\lambda})||_q$, *and* $(\mathbf{v}, \mathbf{y}) \longleftarrow$ *a normalized pair of random or pseudo random vectors (satisfying* $||\mathbf{v}||_2 = ||\mathbf{y}||_2 = 1$*). (The vector* $\mathbf{v}$ *is needed only if the matrix* $A(\tilde{\lambda})$ *is ill conditioned for the input or updated value of* $\tilde{\lambda}$*.)*

COMPUTATIONS:

1. *If* $COUNTER > k$, *output FAILURE and stop. Otherwise set* $C(\tilde{\lambda}) \longleftarrow A(\tilde{\lambda})$ *and apply LIN·SOLVE to compute vector* $C^{-1}(\tilde{\lambda})\mathbf{y}$.
2. *If this application fails (that is, if the matrix* $C(\tilde{\lambda})$ *is singular), then* $C(\tilde{\lambda}) \longleftarrow \frac{1}{\sigma}C(\tilde{\lambda}) + \mathbf{y}\mathbf{v}^H$. *Apply LIN·SOLVE to compute the vector* $C^{-1}(\tilde{\lambda})\mathbf{y}$. *If this application fails (that is, if the matrix* $C(\tilde{\lambda})$ *is still singular), then output PROBABLY G·MULTIPLE and stop.*
3. *Otherwise* $\mathbf{y} \longleftarrow \frac{C^{-1}\mathbf{y}}{||C^{-1}\mathbf{y}||_2}$ *(compute or update a normalized approximate eigenvector).*
4. $\tilde{\lambda} \longleftarrow$ *a root* $\tilde{\lambda}$ *of the equation* $\mathbf{y}^H A(\tilde{\lambda})\mathbf{y} = 0$ *which minimizes the residual norm* $||A(\tilde{\lambda})\mathbf{y}||_2$. *(*$\tilde{\lambda} = \frac{\mathbf{y}^H M\mathbf{y}}{\mathbf{y}^H N\mathbf{y}}$ *if* $A = \lambda N - M$*.) Update the matrix* $A(\tilde{\lambda})$ *for the updated value of* $\tilde{\lambda}$.
5. $\sigma \longleftarrow ||A(\tilde{\lambda})||_q$. *(*$\sigma \longleftarrow \sigma + \tilde{\lambda}_{new} - \tilde{\lambda}_{old}$ *if* $q = 2$ *and* $A = \lambda I - M$*.) If* $||A(\tilde{\lambda})\mathbf{y}||_2 \leq \sigma\tau$ *(that is, if the residual norm is small enough), output* $\lambda_{final} = \tilde{\lambda}$, $\mathbf{y}_{final} = \mathbf{y}$ *and stop. Otherwise set* $COUNTER \longleftarrow COUNTER + 1$ *and go to Stage 1.*

11.6 Inverse iteration with APPs of small ranks

Algorithm 11.1 outputs PROBABLY G·MULTIPLE if LIN·SOLVE fails for both coefficient matrices $A(\tilde{\lambda})$ and $C(\tilde{\lambda})$. According to our study in Section 5, this can occur either because the vectors $\mathbf{v}$ and/or $\mathbf{y}$ lie in or near the ranges of the matrices $A(\lambda)^H$ and/or $A(\lambda)$, respectively (although such a case is unlikely for random vectors $\mathbf{v}$ and $\mathbf{y}$ and singular matrices $A(\lambda)$), or because λ is a geometrically multiple eigenvalue of the matrix polynomial $A(\lambda)$ or lies near another eigenvalue. We can modify Algorithm 11.1 to approximate such an eigenvalue λ as well. We just need to keep adding the outer products $\mathbf{y}\mathbf{v}^H$ of pairs of random or pseudo random vectors $\mathbf{y}$ and $\mathbf{v}^H$ to the matrix $C(\tilde{\lambda})$ until it becomes well conditioned (cf. numerical version of Algorithm 9.2). The resulting algorithm can be viewed as the IPI/IR–RI that for $\tilde{\lambda} \approx \lambda_{A,i}$ employs APPs of small ranks.

Algorithm 11.2. Inverse iteration with APPs of small ranks.

INPUT: *as in Algorithm 11.1.*

OUTPUT: *either FAILURE or an approximation* $(\lambda_{final}, Y_{final})$ *to an eigenpair of* $A(\lambda)$ *such that* $||A(\lambda_{final})Y_{final}||_2 \leq \tau||A(\lambda_{final})||_q$.

INITIALIZATION: $COUNTER \longleftarrow 0$, $i \longleftarrow 0$, $\sigma \longleftarrow ||A(\tilde{\lambda})||_q$, $V_0 \longleftarrow ()$, $Y_0 \longleftarrow ()$ *where* $()$ *denotes the* $n \times 0$ *empty matrix,* $Y_1 \longleftarrow$ *an* $n \times 1$ *random unitary matrix.*

COMPUTATIONS:

1. *If* $COUNTER > k$, *output FAILURE and stop. Otherwise* $C(\tilde{\lambda}) \longleftarrow \frac{1}{\sigma}A(\tilde{\lambda}) + Y_i V_i^H$, *apply LIN·SOLVE to compute the matrix* $C^{-1}(\tilde{\lambda})Y_{\mu(i)}$ *where* $\mu(i) = 1$ *if* $i = 0$, $\mu(i) = i$ *otherwise.*
2. *If this application fails (that is, if the matrix* $C(\tilde{\lambda})$ *is singular), then set* $(\mathbf{v}, \mathbf{y}) \longleftarrow$ *a pair of normalized random or pseudo random vectors such that (*$||\mathbf{v}||_2 = ||\mathbf{y}||_2 = 1$ *and)* $\mathbf{v}^H V_i = \mathbf{0}$ *if* $i > 0$, $V_{i+1} \longleftarrow (V_i, \mathbf{v})$, $Y_{i+1} \longleftarrow (Y_i, \mathbf{y})$ *(so that* $Y_{i+1}V_{i+1}^H = Y_i V_i^H + \mathbf{y}\mathbf{v}^H$*),* $i \longleftarrow i + 1$, $COUNTER \longleftarrow COUNTER + 1$, *and go to Stage 1.*
3. $Y \longleftarrow$ *the Q-factor in the QR factorization of the matrix* $C^{-1}(\tilde{\lambda})Y_{\mu(i)}$ *or, numerically, a properly truncated Q-factor in the rank revealing pivoted QR factorization of the matrix* $C^{-1}(\tilde{\lambda})Y_{\mu(i)}$ *[2, Sections 5.2–5.4], [3, Chapters 4 and 5].*
4. $\tilde{\lambda} \longleftarrow$ *a root* $\tilde{\lambda}$ *of the equation* $\mathrm{trace}(Y^H A(\tilde{\lambda})Y) = 0$ *which minimizes the residual norm* $||A(\tilde{\lambda})Y||_2$. *(*$\tilde{\lambda} = \mathrm{trace}\,\frac{Y^H MY}{Y^H NY}$ *if* $A = \lambda N - M$*.) Update the matrix* $A(\tilde{\lambda})$ *for the updated value of* $\tilde{\lambda}$.
5. $\sigma \longleftarrow ||A(\tilde{\lambda})||_q$. *(*$\sigma \longleftarrow \sigma + \tilde{\lambda}_{new} - \tilde{\lambda}_{old}$ *if* $q = 2$ *and* $A = \lambda I - M$*.) If* $||A(\tilde{\lambda})Y||_2 \le \sigma\tau$ *(that is, if the residual norm is small enough), output* $\lambda_{final} = \tilde{\lambda}$, $Y_{final} = Y$ *and stop. Otherwise set* $COUNTER \longleftarrow COUNTER + 1$, $Y_{\mu(i)} \longleftarrow Y$, *and go to Stage 1.*

Remark 11.2. *By applying Algorithms 11.1 and/or 11.2 to the matrix polynomial* $A^H(\lambda)$, *we approximate its right eigenvectors, which are the left eigenvectors of the matrix polynomial* $A(\lambda)$ *associated with the same eigenvalues.*

Remark 11.3. *Proper selection of the vectors* $\mathbf{v}$ *in Algorithms 11.1 and 11.2 can sometimes simplify the computations. For example, if* $A(\lambda) = R(\lambda) + \mathbf{w}\mathbf{z}^H$ *for a triangular matrix polynomial* $R(\lambda)$ *and two vectors* $\mathbf{w}$ *and* $\mathbf{z}$, *then the choice* $\mathbf{v} = \mathbf{z}$ *can simplify the computation. If we seek no such benefits, we can choose random or pseudo random vectors* $\mathbf{v}$ *or we can let them equal to* $\mathbf{y}$. *The latter choice is natural where* $A(\lambda)$ *is a Hermitian matrix or matrix polynomial.*

Remark 11.4. *For some bad choices of the vectors* $\mathbf{v}$ *and* $\mathbf{y}$ *in Algorithm 11.2, the parameter* i *can exceed the rank of the matrix* $Y_i V_i^H$. *For random vectors* $\mathbf{v}$ *and* $\mathbf{y}$, *this degeneration occurs rarely; moreover, the QR factorizations at Stage 3 can fix it. As a more costly additional fix, at Stage 1 we can apply LIN·SOLVE not to the matrix* $\frac{1}{\sigma}A(\tilde{\lambda}) + Y_i V_i^H$ *but recursively to the matrices* $\frac{1}{\sigma}A(\tilde{\lambda}) + Y_j V_j^H$ *for* $j = 0, 1, \ldots$ *where the matrices* Y_j *and* V_j *are made up of the first* j *columns of the matrices* Y_i *and* V_i, *respectively; here we increment* j *as long as LIN·SOLVE fails and* $j < i$. *By applying the binary search, we can do with at most* $\lceil \log_2 i \rceil$ *applications of LIN·SOLVE.*

11.7 Perturbations and errors in the modified inverse iteration

For $\tilde{\lambda} \approx \lambda$ Algorithms 11.1 and 11.2 compute nearly the same approximations to the eigenspaces of a matrix polynomial $A(\lambda)$ as the IPI and the IR–RI do, and so we can extend the extensive analysis of the latter iterations from [2], [22], [28], [83]–[85], and the bibliography therein. Moreover, we can simplify this analysis because we can involve the matrix $C^{-1}(\mu)$ even where $\lambda = \mu$ is an eigenvalue of the matrix polynomial $A(\lambda)$. Let us estimate the errors to show local quadratic convergence of Algorithms 11.1 and 11.2 for the classical algebraic eigenproblem, where

$$A = A(\lambda) = \lambda I - M,\ \tilde{A} = A(\tilde{\lambda}) = \tilde{\lambda} I - M, \tag{11.2}$$

and the algorithms recursively refine approximations $\tilde{\lambda}$ to an eigenvalue λ and $\tilde{Y}$ to a matrix basis Y for the associated eigenspace.

We first express the errors in the Rayleigh quotients via the eigenvectors errors (without assuming equations (11.2)).

Theorem 11.3. *Let $\tilde{Y}$ and Y be $n \times k$ matrices and write $\Delta = \tilde{Y} - Y$. Then for an $n \times n$ matrix A we have $\tilde{Y}^H A \tilde{Y} - Y^H A Y = \Delta^H A Y + Y^H A \Delta + \Delta^H A \Delta$.*

Next we express the residual $\tilde{C}^{-1}\tilde{Y}$ via the input errors.

Theorem 11.4. *Let Y be a unitary $n \times k$ matrix basis for the null space $N(A)$ of an $n \times n$ matrix A. Let a pair of matrices $\tilde{A}$, $\tilde{Y}$ approximate the pair A, Y. Write $C = A + \tilde{Y}V^H$, $\tilde{C} = \tilde{A} + \tilde{Y}V^H$, $E = \tilde{C} - C = \tilde{A} - A$, $\Delta = \tilde{Y} - Y$ for an $n \times k$ matrix V such that the matrices $B = V^H Y$ and $\tilde{C}$ are nonsingular, so that $B = I_k$ if $V = Y$. Then we have*

a) $\tilde{C}^{-1}\tilde{Y} = YB^{-1} - \tilde{C}^{-1}EYB^{-1}$.

b) Furthermore, suppose that

$$\text{range}(EY) \subseteq \text{range}\, Y = N(A) \tag{11.3}$$

and define a matrix F such that $EYB^{-1} = YF$. Then $\tilde{C}^{-1}\tilde{Y} = YB^{-1}(I - F) + \tilde{C}^{-1}YF^2 + \tilde{C}^{-1}\Delta F$.

Proof. First assume that the matrix C is nonsingular.

Observe that $\tilde{C}^{-1} = (I - \tilde{C}^{-1}E)C^{-1}$. Recall that $AY = 0$, and so $CY = (A + \tilde{Y}V^H)Y = \tilde{Y}(V^H Y) = \tilde{Y}B$, $C^{-1}\tilde{Y} = YB^{-1}$. Therefore,

$$\tilde{C}^{-1}\tilde{Y} = (I - \tilde{C}^{-1}E)C^{-1}\tilde{Y} = YB^{-1} - \tilde{C}^{-1}EYB^{-1}.$$

This proves part a).

Substitute the equation $EYB^{-1} = YF$ into the equation of part a) and obtain that $\tilde{C}^{-1}\tilde{Y} = YB^{-1} - \tilde{C}^{-1}YF$.

Substitute

$$\tilde{C}^{-1}Y = \tilde{C}^{-1}\tilde{Y} - \tilde{C}^{-1}\Delta = YB^{-1} - \tilde{C}^{-1}YF - \tilde{C}^{-1}\Delta$$

on the right-hand side and obtain that

$$\tilde{C}^{-1}\tilde{Y} = YB^{-1}(I - F) + \tilde{C}^{-1}YF^2 + \tilde{C}^{-1}\Delta F.$$

This proves part b).

Relax the assumption that the matrix C is nonsingular by applying infinitesimal perturbations of the matrix A. □

Remark 11.5 and the following lemma support the assumptions in part b).

Lemma 11.1. *Under (11.2), we have*

$$E = (\tilde{\lambda} - \lambda)I, \quad F = (\tilde{\lambda} - \lambda)B^{-1}, \tag{11.4}$$

and assumption (11.3) in part b) holds.

Theorem 11.4 implies the following estimates for the residual norm.

Corollary 11.1. *The norm $||C^{-1}\tilde{Y} - YB^{-1}||$ is in $O((|E||)$ under the assumptions of Theorem 11.4 a) and in $O((||\Delta|| + ||F||)||F||)$ under the assumptions of Theorem 11.4 b).*

Combining Theorem 11.3, Lemma 11.1, and Corollary 11.1 immediately implies quadratic convergence of Algorithms 11.1 and 11.2 to the eigenvalue/eigenspace pair assuming (11.2), the choice of $V = \tilde{Y}$, and a close initial approximation to the eigenvalue λ (but not necessarily to the associated eigenspace).

Remark 11.5. *In Theorem 11.4 b) we require that the matrix B be nonsingular. This property is expected to hold under random variation of the matrices $\tilde{Y}$ and V. The above estimate for the residual norm does not depends on the norm $||B^{-1}||_2$, which we estimate below only for the sake of completeness.*

Lemma 11.2. *Let $V = \tilde{Y}$ be a unitary matrix and let $||\Delta||_2 < 1$. Then the matrix B is nonsingular and $||B^{-1}||_2 \le \frac{1}{1-||\Delta||_2}$.*

Proof. Under the assumptions of the lemma, we have $B = I_k + \Delta^H Y$ and $B^{-1} = I_k + \sum_{i=1}^{\infty}(-\Delta^H Y)^i$, and the lemma follows. □

11.8 Experimental iteration count for the IPI and Algorithm 11.1

In Tables 11.1 and 11.2 we show the numbers of iterations required for the convergence of the IPI and Algorithm 11.1. We display the average (mean) values and standard deviations in 200 tests with $n \times n$ matrices $A = \lambda I - M$ for $M = G^{-1}TG$, $n = 64$ and $n = 100$, G being either a random matrix or the Q-factor in QR factorization of a random matrix, and T from one of the following four matrix classes.

1. $T = D_r$ is a real diagonal matrix with random entries in the closed line interval $[0, 10]$.

2. $T = D_c$ is a complex diagonal matrix whose entries have random absolute values in the line segment $[0, 10]$ and random arguments in the closed line interval $[0, 2\pi)$.

3. $T = D_r + \mathbf{e}_1\mathbf{v}^T + \mathbf{u}\mathbf{e}_n^T$ is an arrow-head matrix, D_r is a matrix of class 1, and the vectors $\mathbf{u}$ and $\mathbf{v}$ have random entries in the line segment $[0, 10]$.

4. $T = D_r + \mathbf{u}\mathbf{v}^T$, D_r and $\mathbf{v}$ are as in matrix class 3, and the vector $\mathbf{u}$ has random coordinates in the closed line segment $[0, 1]$.

The results of our extensive tests reported in Tables 11.1 and 11.2 confirm that the IPI and Algorithm 11.1 converge with about the same rate, even though Algorithm 11.1 deals with better conditioned matrices.

Table 11.1: Iteration count for IPI and Algorithm 11.1 with unitary matrix G

Matrix		Algorithm 11.1		IPI	
Classes	n	iter	std dev	iter	std dev
$T = D_r$	64	4.74	1.145	4.93	1.242
	100	4.71	1.277	4.88	1.299
$T = D_c$	64	5.67	1.415	5.61	1.396
	100	5.67	1.461	5.62	1.321
$T = D_r + \mathbf{e}_1\mathbf{v}^T + \mathbf{u}\mathbf{e}_n^T$	64	4.94	1.230	5.01	1.341
	100	4.75	1.176	4.75	1.260
$T = D_r + \mathbf{u}\mathbf{v}^T$	64	5.77	1.668	5.95	1.808
	100	5.54	1.445	5.67	1.553

Table 11.2: Iteration count for IPI and Algorithm 11.1 with random matrices G

Matrix		Algorithm 11.1		IPI	
Classes	n	iter	std dev	iter	std dev
$T = D_r$	64	5.36	2.532	5.36	2.520
	100	4.88	2.509	4.86	2.452
$T = D_c$	64	5.76	1.716	5.71	1.516
	100	5.59	1.401	5.64	1.497
$T = D_r + \mathbf{e}_1\mathbf{v}^T + \mathbf{u}\mathbf{e}_n^T$	64	5.09	1.621	5.03	1.605
	100	4.72	1.473	4.67	1.467
$T = D_r + \mathbf{u}\mathbf{v}^T$	64	5.550	1.907	5.550	1.872
	100	5.660	2.118	5.555	1.992

11.9 Further extensions

We can extend Algorithm 11.2 to simultaneous approximation of more than one eigenvalue of a matrix polynomial $A(\lambda)$ (e.g., a cluster of h eigenvalues or a pair of complex conjugate eigenvalues of a real matrix polynomial M) by modifying its Stage 4 as follows:

4. $(\tilde{\lambda}_i, \tilde{Y}_i) \longleftarrow$ *the eigenpairs of the* $k \times k$ *matrix polynomial* $B(\lambda) = Y^H A(\tilde{\lambda}) Y$; $Y_i \longleftarrow Y\tilde{Y}_i$, $i = 1, \ldots, h$.

In this case we should continue the computations with h applications of Algorithm 11.2 initialized with approximate eigenpairs $(\tilde{\lambda}_i, Y_i)$.

Especially, suppose we have a pair of the initial complex conjugate approximations $\tilde{\lambda}_1 \approx \lambda_1$ and $\tilde{\lambda}_2 \approx \lambda_2$ to a pair (λ_1, λ_2) of complex conjugate eigenvalues of a matrix $\lambda I - M$ where M is a real matrix. In this case the right invariant subspace of M associated with λ_1 and λ_2 equals $N(A_{1,2})$, we have $N(W_{1,2}) \approx N(A_{1,2})$ where $A_{1,2} = (\lambda_1 I - M)(\lambda_2 I - M)$ and $W_{1,2} = (\tilde{\lambda}_1 I - M)(\tilde{\lambda}_2 I - M)$ are real matrices, $W_{1,2} \approx A_{1,2}$, and the real matrix $\tilde{C}_{1,2} = \frac{1}{\sigma} W_{1,2} + Y_i V_i^H$ replaces the matrix $C(\tilde{\lambda})$ in Algorithm 11.2.

Similarly we can refine approximations $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_h$ to complex eigenvalues $\lambda_1, \ldots, \lambda_h$ of $A(\lambda)$ where the matrices $A_h = \prod_{i=1}^{h} (\lambda_i I - M)$ and $W_h = \prod_{i=1}^{h} (\tilde{\lambda}_i I - M)$ play the roles of the matrices $A_{1,2}$ and $W_{1,2}$, respectively.

Finally, one can extend A-preconditioning to any eigen-solver involving ill conditioned linear system of equations. As we mentioned, this includes the IRRI iteration, the Jacobi–Davidson algorithm, the shift-and-invert enhancements of the Arnoldi and Lanczos algorithms [22], and the deflation stage of the QR algorithm.

12 Floating-point Summation and Multiplication

In Sections 7.5, 7.6, and 10.2 we needed effective algorithms for floating-point summation and multiplication to counter the cancellation of the leading significant bits in the representation of the output values. For an extensive bibliography on these floating-point operations, see [57]–[59], [86]–[90], and the references therein. The study goes back over three decades to the Kahan–Babushka's and Dekkert's classical algorithm for the compensated summation of two floating-point numbers, which we reproduce next from [89, Algorithm 1.1].

Algorithm 12.1. Compensated floating-point summation of two numbers.

function$[x, y]$=FastTwoSum(a, b)
x =fl$(a + b)$
y =fl$((a - x) + b)$.

It can be shown that this is the error-free floating-point summation if $|a| \geq |b|$, that is, in this case $x + y = a + b$.

Substantial further progress has resulted in a large number of effective advanced algorithms for floating-point summation, covered in the cited bibliography. The selection of the most appropriate among the them is a nontrivial task beyond the scope of our paper. For completeness of our presentation, however, we next describe some rudimentary and elementary techniques, which should be sufficient for our applications.

Problem 12.1. Floating point summation.

INPUT: *k IEEE standard floating-point numbers $s_1, \dots, s_k$, $s_i = \sigma_i 2^{e_i} f_i$ where all σ_i are equal to -1 or one, all e_i are integers in a fixed range $[1-r, r]$, all f_i are either zeros or binary numbers in the range $[1, 2)$ represented with $p+1$ bits, including the leading (that is, leftmost and most significant) bit one. Here $r = 127$ and $p = 23$ for the IEEE single precision numbers and $r = 1023$ and $p = 52$ for the IEEE double precision numbers. (The actual IEEE representation uses the p-bit fractions $f_i - 1$, lying in the range $[0, 1)$, but we use the nomenclature "fractions" for f_i and their segments. Furthermore, we say that the terms s_i are represented with the $(p+1)$-bit precision and call them $(p+1)$-bit numbers, for short. For simplicity we assume having no vanishing summands $s_i = 0$.)*

OUTPUT: *a floating point sum $s = \mathrm{fl}(\sum_{i=1}^{k} s_i) = \sigma 2^e f$ where σ is equal to -1 or one, e is an integer in the range $[1-r, r]$, and f is either zero or a binary number in the range $[1, 2)$ represented with $p+1$ bits, including the leading bit one.*

The straightforward solution algorithm can lead to catastrophic cancellation of the leading bits in the exact sum.

Example 12.1. *Let $k = 6$, $s_1 = 2^{p+2} f_1$, $s_2 = -f_2$, $s_3 = 2^{-p-2} f_3$, $s_4 = f_2 = -f_2$, $s_5 = -s_1 = -2^{p+2} f_1$, and $s_6 = s_3 = 2^{-p-2} f_3$. The exact sum is equal to $s_1 + \cdots + s_6 = s_3 + s_6 = 2^{-p-1} f_3$, but due to the rounding errors, we output $s_6 = \mathrm{fl}(s_1 + \cdots + s_6)$, thus losing the prefix of the $p+1$ leading bits.*

To counter such a calamity, examine the floating point subtraction $s_i + s_j$ of two IEEE numbers where $s_i s_j < 0$ and $|s_j| \le |s_i|$. The absolute error is minimized for a fixed i or j if $|e_i - e_j|$ is minimized. Further observe that the leading bits are quite well preserved in the summation of only positive or only negative numbers. This suggests the following simple algorithm.

Algorithm 12.2. Summation with ordered subtractions. *Make up two sorted lists of all positive and of all negative summands s_i, respectively. If both lists are nonempty, compute the sum $s_i + s_j$ where $s_i s_j < 0$ and the integer $|e_i - e_j|$ is minimum. (If there are ties, maximize at first $\max\{e_i, e_j\}$ and if there still remain ties, then maximize $\min\{e_i, e_j\}$.) Update the lists by including the computed sum and removing the terms s_i and s_j. Repeat the computations until one list becomes empty. Then output the sum of all values s_h that are currently in the other list and stop.*

The algorithm computes the sum in Example 12.1 with no error but works poorly in the more rare cases represented by the following example.

Example 12.2. *Let* $k = 6$, $s_1 = 2^{p+2}f_1$, $s_2 = s_3 = s_4 = s_5 = -f_1$, $s_6 = 2^{-p-2}f_6$ *where the binary representation of the fraction* f_1 *ends with the suffix string* $\{01\}$. *The exact sum* $s_1 + \cdots + s_6$ *equals* $s_6 = 2^{-p-2}f_6$, *but due to the rounding errors in the summation in Algorithm 12.2, we obtain* $\mathrm{fl}(s_1+\cdots+s_5) = 2^4$ *and* $\mathrm{fl}(2^4 + s_6) = 2^4 >> s_6$.

We propose two algorithms for repairing such damage.

Algorithm 12.3. Summation with ordered subtractions and segmentation. *Apply Algorithm 12.2, but wherever it computes* $\mathrm{fl}(s_i + s_j)$ *for* $s_i s_j < 0$ *and* $0 < e_i - e_j \le p$, *replace the summand* s_j *by the pair of summands* s'_j *and* s''_j *such that* $s_j = s'_j + s''_j$, $s'_j = \sigma_j 2^{e_j} f'_j$, $s''_j = \sigma_j 2^{e_i} f''_j$. *(The trailing bit of the summand* s_i *partitions the "fraction"* $f_j = f'_j + 2^{e_i - e_j} f''_j$ *into its* $(e_i - e_j)$*-bit prefix* f'_j *and its suffix* f''_j.*) Then replace the summand* s_i *in its list with* $\mathrm{fl}(s_i + s'_j) = s_i + s'_j$ *and resume the computations.*

The algorithm partitions the "fraction" f_j to yield the error-free subtraction $s_i + s'_j$ and therefore enforces the error-free summation until all subtractions $s_i + s_j$ with $|e_i - e_j| \le p$ are performed. This must occur in a finite number of steps because every subtraction step strictly decreases the sum of the absolute values of all positive and negative summands. The remaining $h - 1 < k$ operations of subtraction and addition cause only a negligible relative error of at most $(h-1)/2^{p+1}$.

Our third algorithm was proposed in 1992 in [58] and also published in [59]. It can be combined with Algorithm 12.2 or applied independently.

In its basic version, the algorithm computes the floating point sum $s = \sigma 2^e f$ and a positive upper bound δ on its absolute error. If the ratio $\delta/|s|$ is small enough, the algorithm outputs s and stops. Otherwise it computes the integer $d = 1 + \lceil \log_2(|s| + \delta) \rceil$ and then recomputes the sum modulo 2^d. The modular reduction does not change the exact sum of all input terms because this sum does not exceed 2^{d-1}. The error bound δ, the exponent d, and the sum modulo 2^d are recursively recomputed and decrease until the ratio $\delta/|s|$ becomes small enough. Then the current sum is output.

We choose $\delta = (h-1)2^{e_+ - p - 1}$ where $e_+ = \max_i e_i$, the maximum is over all h summands s_i that are currently in the list (some summands can vanish in the reduction modulo 2^d), and e_i are the exponents in the IEEE representation of the summands s_i.

Below, we slightly modify this algorithm by applying the modular reduction only to the summands and by summing the results with double precision. Here is the formal description of the algorithm.

Algorithm 12.4. Summation with backward segmentation.

INITIALIZATION: *fix a positive tolerance* t, *set the counter of the summands* $h \leftarrow k$, *and make the list of all* h *summands.*

COMPUTATIONS:

1. Compute the floating point sum $s = \mathrm{fl}(s_1 + \cdots + s_h)$*, the maximum exponent* $e_+ = \max_{i=1}^{h} e_i$ *over all* h *summands* $s_i = \sigma_i 2^{e_i} f_i$ *currently in the list, and the upper bound* $\delta = (h-1)2^{e_+ - p - 1}$ *on the overall rounding error. If* t *exceeds the ratio* $\delta/|s|$*, output* s *and stop.*

2. Otherwise reduce all summands s_i *modulo* 2^d *for* $d = 1 + \lceil \log_2(s + \delta) \rceil$*, that is, set to zero all their bits representing the powers* 2^g *for* $g \geq d$*. Remove all vanished summands from the list, decrease the counter* h *respectively, reenumerate the summands* s_i *from 1 to* h*, and go to Stage 1.*

The algorithm recursively decreases the maximum exponent e_+ and the error bound δ, and thus, in a finite number of loops, approximates the exact sum within a relative error of at most t. In particular for $t = 0$ the algorithm outputs the error-free sums in both Examples 12.1 and 12.2.

Next we extend our summation algorithms.

Numerical stabilization of bilinear computations and further extensions

Floating-point multiplication outputs the correct prefix of the $p+1$ leading bits of the exact product but loses the remaining bits. This loss is irreparable if the leading bits are cancelled in the subsequent summation, but we can avoid the loss with a proper segmentation of the multiplicands.

Seeking the product of two positive IEEE numbers $f_1 2^{e_1}$ and $f_2 2^{e_2}$, first represent their $(p+1)$-bit "fractions" as the sums $f_1 = f_{10} + 2^{-e_{11}} f_{11}$ and $f_2 = f_{20} + 2^{-e_{21}} f_{21}$ where $e_{11} \geq q$, $e_{21} \geq q$, and the "fractions" f_{10} and f_{20} are q-bit numbers, whereas f_{11} and f_{21} are at most $(p-q)$-bit numbers, $q = \lfloor p/2 \rfloor$, and all nonvanishing "fractions" lie in the range $[1, 2)$.

If p is an odd integer, then $q = p - q$, and we represent the product $(f_1 2^{e_1})(f_2 2^{e_2}) = f_1 f_2 2^{e_1+e_2}$ as the exact sum of four IEEE numbers, $f_{10} f_{20} 2^{e_1+e_2}$, $f_{10} f_{21} 2^{e_1+e_2-e_{21}}$, $f_{11} f_{20} 2^{e_1+e_2-e_{11}}$, and $f_{11} f_{21} 2^{e_1+e_2-e_{11}-e_{21}}$, each of at most $p+1$ bits.

If p is an even integer, then $q = p - q - 1$, and the first three products are at most $(p+1)$-bit numbers, whereas the exact representation of the product $f_{11} f_{21}$ may require one extra (trailing) bit. To avoid losing this bit, we can represent the product as the exact sum of two numbers with at most $(p+1)$-bit precision and thus represent the original product as the exact sum of at most five IEEE numbers.

With this technique, given a k-term bilinear expression β, we can first apply the above algorithm k times to compute $4k$ or $5k$ IEEE numbers (each represented with at most $p+1$ bits) whose exact sum equals β. Then it remains to apply Algorithms 12.2–12.4 and to output the $(p+1)$-bit prefix of this sum with a negligible error. If we sum l bilinear expressions (e.g., the (i,j)th entries of l matrices $A_1, \ldots, A_l$ for a fixed pair of i and j), we can first compute all kl products exactly and then apply our summation algorithms only once, to sum all these kl terms.

These techniques cover many matrix computations, including various iterative processes for linear systems of equations [58], [59], and can be readily extended to the exact computation of multivariate polynomial expressions and to high precision approximation of rational and algebraic expressions. Indeed, for two positive integers k and p, we can represent the $((p+1)k)$-bit prefix of an infinite precision binary number as the exact sum of k terms, each being a $(p+1)$-bit IEEE number.

PART IV. CONCLUSION

We proposed primal and dual A-preprocessing and analyzed its basic properties and the benefits of using it. We linked it to Aggregation Processes, iterative refinement, M-preconditioning, factorization, and the SVD and null space computations. We generated random and structured pseudo random APCs and studied their affect on the conditioning of an input matrix, both theoretically and experimentally. We proposed various techniques for the applications of APCs to solving linear systems of equations and other fundamental matrix computations, demonstrated the power and promise of our approach, and thus, hopefully, motivated its further study.

Our extensive analysis of A-preprocessing can have some independent interest and applications.

Our plans include further theoretical and experimental study of the A-preconditioning, with special attention to the case where the input matrix has nested clusters of small singular values, to comparison of our five approaches to solving linear systems of equations (based on the Null Aggregation in Section 10.1, on the primal and dual SMW formulae, and on the primal and dual Null SMW formulae), and to the links of the Schur and the Null Aggregation with the Aggregation Processes in [17]. We plan to test the power of all our algorithms in Sections 7.7 and 10 for the APC-based solution of the linear systems of equations. We shall compare the algorithms with each other and with the SVD-based algorithms.

A specific subject for our experiments is the affect of our APCs on the accuracy of the direct methods applied to various poorly conditioned linear systems of equations and on the speed of the convergence of the CG, GMRES, and other iterative algorithms applied to such systems.

Testing the efficiency of the APPs in Examples 4.1–4.6 and other structured APPs is another important subject. We plan to begin such tests with A-preconditioning of structured linear systems of equations. Then we shall approximate the inverse of an input matrix by applying the CG or GMRES algorithms and finally refine the computed approximation by applying structured versions of Newton's iteration [6, Chapter 6], [7], [8].

References

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

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

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

[4] B. Mourrain, V. Y. Pan, Multivariate Polynomials, Duality and Structured Matrices, *J. of Complexity*, **16**, **1**, 110–180, 2000.

[5] D. Bondyfalat, B. Mourrain, V. Y. Pan, Computation of a Specified Root of a Polynomial System of Equations Using Eigenvectors, *Linear Algebra and Its Applications*, **319**, 193–209, 2000.

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

[7] V. Y. Pan, M. Van Barel, X. Wang, G. Codevico, Iterative Inversion of Structured Matrices, *Theoretical Computer Science*, Special Issue on Algebraic and Numerical Algorithms (I. Z. Emiris, B. Mourrain, and V. Y. Pan editors), **315**, **2**–**3**, 581–592, 2004.

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

[9] V. Y. Pan, Parallel Solution of Toeplitz-like Linear Systems, *J. of Complexity*, **8**, 1–21, 1992.

[10] V. Y. Pan, Decreasing the Displacement Rank of a Matrix, *SIAM J. on Matrix Analysis*, **14, 1**, 118–121, 1993.

[11] V. Y. Pan, Concurrent Iterative Algorithm for Toeplitz-like Linear Systems, *IEEE Trans. on Parallel and Distributed Systems*, **4, 5**, 592–600, 1993.

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

[13] H. Brönnimann, I. Z. Emiris, V. Y. Pan, S. Pion, Sign Determination in Residue Number Systems, *Theoretical Computer Science*, **210**, **1**, 173–197, 1999. Proceedings Version in *Proc. 13th Ann. ACM Symp. on Computational Geometry*, 174–182, ACM Press, New York, 1997.

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

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

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

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

[18] X. Wang, Affect of Small Rank Modification on the Condition Number of a Matrix, preprint, 2006.

[19] M. T. Chu, R. E. Funderlic, G. H. Golub, A Rank-One Reduction Formula and Its Applications to Matrix Factorizations, *SIAM Review*, **37**, **4**, 512–530, 1995.

[20] L. Hubert, J. Meulman, W. Heiser, Two Purposes for Matrix Factorization: A Historical Appraisal, *SIAM Review*, **42**, **1**, 68–82, 2000.

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

[22] G. W. Stewart, *Matrix Algorithms, Vol II: Eigensystems*, SIAM, Philadelphia, 1998.

[23] G. Peters, J. H. Wilkinson, Computing the Generalized Singular Value Decompositions, *SIAM J. on Sientific and Statistical Computing*, **4**, 1112–1146, 1986.

[24] D. A. Bini, L. Gemignani, V. Y. Pan, Inverse Power and Durand/Kerner Iteration for Univariate Polynomial Root-finding, *Computers and Mathematics (with Applications)*, **47**, **2/3**, 447–459, 2004. (Also Technical Reports TR 2002 003 and 2002 020, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York,* 2002.)

[25] D. A. Bini, L. Gemignani, V. Y. Pan, Fast and Stable QR Eigenvalue Algorithms for Generalized Companion Matrices and Secular Equation, *Numerische Math.*, **3**, 373–408, 2005. (Also Technical Report 1470, *Department of Math., University of Pisa*, Pisa, Italy, July 2003.)

[26] D. A. Bini, L. Gemignani, V. Y. Pan, Improved Initialization of the Accelerated and Robust QR-like Polynomial Root-finding, *Electronic Transactions on Numerical Analysis*, **17**, 195–205, 2004.

[27] L. N. Trefethen, D. Bau, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[28] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[29] R. A. Demillo, R. J. Lipton, A Probabilistic Remark on Algebraic Program Testing, *Information Processing Letters*, **7**, **4**, 193–195, 1978.

[30] R. E. Zippel, Probabilistic Algorithms for Sparse Polynomials, *Proceedings of EUROSAM'79, Lecture Notes in Computer Science*, **72**, 216–226, Springer, Berlin, 1979.

[31] J. T. Schwartz, Fast Probabilistic Algorithms for Verification of Polynomial Identities, *Journal of ACM*, **27**, **4**, 701–717, 1980.

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

[33] T. W. Li, Z. Zeng, A Rank Revealing Method with Updating, Downdating and Applications, *SIAM J. on Matrix Analysis and Applications*, **26**, 918–946, 2005.

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

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

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

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

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

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

[40] T. Kailath, S. Y. Kung, M. Morf, Displacement Rank of Matrices and Linear Equations, *J. of Math. Analysis and Applications*, **68, 2**, 395–407, 1979.

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

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

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

[44] T. Kailath, A. H. Sayed (editors), *Fast Reliable Algorithms for Matrices with Structure*, SIAM Publications, Philadelphia, PA, 1999.

[45] Y. Eidelman, I. Gohberg, Fast Inversion Algorithms for a Class of Structured Operator Matrices, *Linear Algebra and Its Applications*, **371**, 153–190, 2003.

[46] R. Vandebril, Semiseparable Matrices and the Symmetric Eigenvalue Problem, PhD Thesis, *Katholieke Universiteit Leuven, Departement Computerwetenschappen*, Leuven, Belgium, 2004.

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

[48] 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.

[49] D. Bini, V. Y. Pan, Parallel Complexity of Tridiagonal Symmetric Eigenvalue Problem, *Proc. 2nd Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA'91)*, 384–393, ACM Press, New York, and SIAM Publications, Philadelphia, 1991.

[50] D. Bini, V. Y. Pan, Computing Matrix Eigenvalues and Polynomial Zeros Where the Output Is Real, *SIAM J. on Computing*, **27, 4**, 1099–1115, 1998.

[51] M. R. Hestenes, E. Stiefel, Methods of Conjugate Gradient for Solving Linear Systems, *J. of Research of the National Bureau of Standards*, **49**, 409–436, 1952.

[52] O. Axelsson, A survey of Preconditioned Iterative Methods for Linear Systems of Equations, *BIT*, **25**, 166–187, 1985.

[53] Y. Saad, M. N. Schultz, GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems, *SIAM J. on Sientific and Statistical Computing*, **7**, 856–869, 1986.

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

[55] Y. Saad, H. A. van der Vorst, Iterative Solution of Linear Systems in the 20th Century, *J. of Computational and Applied Math.*, **123**, 1–33, 2000.

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

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

[58] 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.

[59] 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.

[60] 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.

[61] K. L. Clarkson, Safe and Effective Determinant Evaluation, *Proc. 33rd Ann. IEEE Symp. on Foundations of Computer Science*, 387–395, IEEE Computer Society Press, Los Alamitos, California, 1992.

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

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

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

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

[66] C. Lanzcos, An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators, *J. of Research of the National Bureau of Standards*, **45**, 255–280, 1950.

[67] J. Cullum, W. E. Donath, The Block Lanczos Algorithm for Computing the q Algebraically Largest Eigenvalues and a Corresponding Eigenspace of Large Sparse Real Symmetric Matrices, *Proc. of IEEE Conf. on Decision and Control*, 505–509, Phoenix, Arizona, 1974.

[68] D. Wiedemann, Solving Sparse Linear Equations over Finite Fields, *IEEE Trans. Inf. Theory* IT–**32**, 54–62, 1986.

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

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

[71] V. Y. Pan, Randomized Acceleration of Fundamental Matrix Computations, *Proc. Symp. on Theoretical Aspects of Computer Science (STACS)*, *Lect. Notes in Comput. Sci.*, **2285**, 215–226, Springer, Heidelberg, Germany, 2002.

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

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

[74] E. Kaltofen, G. Villard, Complexity of Computing Determinants, *J. Computational Complexlty*, **13, 3–4**, 91–130, 2004.

[75] A. Storjohann, High Order Lifting and Integrality Certification, *J. of Symbolic Computation,* **36**, **3–4,** 613–648, 2003. (Proc.

version in *Proc. of Intern. Symposium on Symbolic and Algebraic Computation*, 246–254, ACM Press, New York, 2002.)

[76] A. Storjohann, The Shifted Number System for Fast Linear Algebra on Integer Matrices, Technical Report TR CS 2004 18, *School of Computer Science, University of Waterloo*, Canada, April 2004.

[77] V. Y. Pan, X. Wang, Acceleration of Euclidean Algorithm and Extensions, *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'02)*, (Teo Mora editor), 207–213, ACM Press, New York, 2002.

[78] V. Y. Pan, Can We Optimize Toeplitz/Hankel Computations? *Proc. of the 5th Intern. Workshop on Computer ALgebra in Scientific Computing (CACS'03)*, (E. W. Mayr, V. G. Ganzha, and E. V. Vorozhtzov Editors), 253–264, Technicshe Univ. München, Germany, 2002.

[79] V. Y. Pan, Nearly Optimal Toeplitz/Hankel Computations, Technical Reports 2002 001 and 2002 017, *Ph.D. Program in Computer Science, The Graduate Center, CUNY*, New York, 2002.

[80] V. Y. Pan, B. Murphy, R. E. Rosholt, X. Wang, Toeplitz and Hankel Meet Hensel and Newton: Nearly Optimal Algorithms and Their Practical Acceleration with Saturated Initialization. Technical Report 2004 013, *PhD Program in Computer Science, The Graduate Center, CUNY*, New York, 2004.

[81] E. R. Jessup, A Case Against a Divide and Conquer Approach to the Nonsymmetric Eigenvalue Problem, *Appl. Numer. Math.,* **12**, 403–420, 1993.

[82] T. Kailath, *Linear Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.

[83] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

[84] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980, and SIAM, Philadelphia, 1998.

[85] I. C. F. Ipsen, Computing an Eigenvector with Inverse Iteration, *SIAM Review*, **39**, 354–391, 1998.

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

[87] N. J. Higham, The Accuracy of Floating Point Summation, *SIAM J. on Scientific Computing*, **14**, 783–799, 1993.

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

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

[90] S. M. Rump, T. Ogita, and S. Oishi, Accurate Floating-Point Summation, Tech. Report 05.12, *Faculty for Information and Communication Sciences, Hamburg University of Technology*, November 2005, 41 pages, submitted for publication, 2006.