2007

TR-2007002: Additive Preconditioning and Aggregation in Matrix Computations

Victor Y. Pan

Brian Murphy

Rhys Eric Rosholt

Dmitriy Ivolgin

Yuqing Tang

See next page for additional authors

Additive Preconditioning and Aggregation in Matrix Computations *

Victor Y. Pan[1,3,4],
Brian Murphy[1], Rhys Eric Rosholt[1],
Dmitriy Ivolgin[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. Program in Computer Science
The City University of New York
New York, NY 10036 USA
firstnameinitiallastname@gc.cuny.edu

[3] Ph.D. Programs in Mathematics and Computer Science
The City University of New York
New York, NY 10036 USA

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

Abstract

Multiplicative preconditioning is a popular SVD-based techniques for the solution of linear systems of equations, but our SVD-free additive preconditioners are more readily available and better preserve matrix structure. We combine additive preconditioning with aggregation and other relevant techniques to facilitate the solution of linear systems of equations and some other fundamental matrix computations. Our analysis and experiments show the power of our approach, guide us in selecting most effective policies of preconditioning and aggregation, and provide some new insights into these and related subjects of matrix computations.

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

Key words: Matrix computations, Additive preconditioning, Aggregation, MSAs.

1 Introduction

1.1 Background: multiplicative preconditioning

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

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

1.2 Our alternative approach

As an alternative or complementary tool, we propose *additive preprocessing* $A \leftarrow C = A + UV^H$, i.e., we add a matrix UV^H to an ill conditioned matrix A to obtain its better conditioned *additive modification* C. Here and hereafter A^H denotes the Hermitian (that is complex conjugate) transpose of a matrix A, which is just its transpose A^T where A is a real matrix. Hereafter we also write

- $(S_1, \dots, S_k)$ for a $1 \times k$ block matrix with the blocks S_1, …, S_k
- I_k for the $k \times k$ identity matrix
- A^{-H} for $(A^H)^{-1} = (A^{-1})^H$
- $\sigma_j(A)$ for the jth largest singular value of a matrix A
- $||A||_2 = \sigma_1(A)$ for its 2-norm
- $\rho = \operatorname{rank} A$ for its rank and
- $\operatorname{cond}_2 A = \sigma_1(A)/\sigma_\rho(A)$ for its condition number.

We call a matrix *normalized* if its 2-norm equals one, and we use the abbreviations of *MPPs*, *APPs*, *MPCs*, *APCs*, *A-modification*, and *M-* and *A-preprocessing* and *M- and A-preconditioning* for multiplicative and additive preprocessors and preconditioners, additive modification, and multiplicative and additive preprocessing and preconditioning, respectively.

In this paper we outline A-preconditioning and its applications to the solution of linear systems of equations and some other fundamental matrix computations. In other papers we elaborate upon this outline and present further technical details and the results of numerical tests that show the power of our approach. The tests have been designed by the first author and performed by his coauthors. Otherwise all this work as well as all typos and other errors are due to the first author.

1.3 Random A-preprocessors

Clearly, we can effectively precondition a matrix if we know its SVD, but even a random properly scaled APP of a suitable rank is likely to be a good APC. More precisely, suppose a matrix A has full rank ρ. Then according to our analysis and extensive experiments in [4]–[6], we are likely to arrive at an A-modification $C = A + UV^H$ with $\mathrm{cond}_2 C$ of the order of $\sigma_1(A)/\sigma_{\rho-r}(A)$ if an APP UV^H of a rank r is

a) random (general, sparse, or structured),

b) well conditioned, and

c) properly scaled so that the ratio $||A||_2/||UV^H||_2$ is neither very large nor very small.

In particular if $\sigma_r(A) \gg \sigma_{r+1}(A)$, but the ratio $\sigma_{r+1}(A)/\sigma_\rho(A)$ is not large, then an APP UV^H chosen according to rules a)–c) is likely to be an APC, that is to define a well conditioned A-modification $C = A + UV^H$, if and only if $\mathrm{rank}(UV^H) \geq r$. We can compute the threshold integer r by recursively incrementing its initial value from zero, updating the matrices U, V, and C, and estimating the condition number of the matrix C. Clearly, the process can be speeded up with the randomized binary search.

In contrast, random M-preprocessing cannot help much against ill conditioning because $\mathrm{cond}_2 A \leq \prod_i \mathrm{cond}_2 F_i$ if $A = \prod_i F_i$.

We can apply the effective norm and condition estimators in [7, Section 3.5.4] and [8, Section 5.3] for computing APPs under rules b) and c), as well as at the other stages of A-preconditioning. E.g., we can check if $\mathrm{cond}_2 C$ is as small as desired, and if it is not, we can recompute the A-modification C for new generators U and V chosen either again according to the rules a)–c) or (with more work and more confidence in success) according to the recipes at the end of Section 4.3.

Our extensive tests show that very weak randomization is sufficient for generating effective APCs, and we exhibit many general purpose APCs with various

patterns of structure and sparseness [6, Examples 4.1–4.6]. Here is Example 4.6 from [6].

Structured and sparse Hermitian APPs. Let k, $n_1, \dots, n_k$ be positive integers (fixed or random) such that $kr + n_1 + \cdots + n_k = n$. For $i = 1, \dots, k$, let $0_{r,n_i}$ denote $r \times n_i$ matrices filled with zeros and let T_i denote some $r \times r$ fixed or random structured well conditioned matrices, e.g., the matrices of the discrete Fourier, sign or cosine transforms, matrices with a fixed displacement structure, semi-separable (rank structured) matrices, or sparse matrices with fixed patterns of sparseness (see [9]–[13] and the bibliography therein and in [14]). Let $U = P(T_1, 0_{r,n_1}, \dots, T_k, 0_{r,n_k})^T$. Choose an $n \times n$ permutation matrix P (in the simplest case let $P = I$) and define the APP UU^H.

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

1.4 Dual A-preprocessing

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

$$(C_-)^{-1} = (A^{-1} + VU^H)^{-1} = A - AVH^{-1}U^H A, \quad H = I_q + U^H AV. \tag{1.1}$$

We call the latter expressions the *dual SMW inversion formula*, which is our simple counterpart to the *primal SMW inversion formula* of Sherman, Morrison, and Woodbury [7, page 50], [8, Corollary 4.3.2],

$$A^{-1} = (C - UV^H)^{-1} = C^{-1} + C^{-1}UG^{-1}V^H C^{-1}, \ G = I_r - V^H C^{-1}U. \tag{1.2}$$

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

$$\mathbf{y} = A^{-1}\mathbf{b} = \mathbf{z} - VU^H\mathbf{b}, \quad ((C_-)^{-1})^{-1}\mathbf{z} = \mathbf{b}. \tag{1.3}$$

By extending our analysis of A-preconditioning, we obtain that $\mathrm{cond}_2 C_- = \mathrm{cond}_2((C_-)^{-1})$ is likely to be of the order of the ratio $\sigma_{q+1}(A)/\sigma_n(A)$ if a dual APC VU^H of a rank q has been chosen according to rules a) and b) together with the following counterpart of rule c),

d) the ratio $||A^{-1}||_2/||VU^H||_2$ is neither large nor small.

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

The primal and dual SMW inversion formulae have the following simple counterparts for expressing the determinant of the matrix A via the matrices C and G in equations (1.2) and $(C_-)^{-1}$ and H in equations (1.1),

$$\det A = (\det G) \det C, \tag{1.4}$$

$$\det A = (\det H) \det((C_-)^{-1}). \tag{1.5}$$

2 Two impacts of preconditioning

1. *Preconditioning as a means of convergence acceleration*

Suppose the CG 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 an iteration step adds the order of $\sqrt{\text{cond}_2 A}$ new correct bits per a variable (cf. [7, 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.

Similar comments can be applied to the GMRES and various other algorithms of this kind [7, Sections 10.2–10.4], [15]–[17].

Likewise, preconditioning can accelerate the Wilkinson's iterative refinement algorithm in [7, Section 3.5.3], [8, Sections 3.3.4 and 3.4.5], and [18, Chapter 11]. Indeed, its iteration step adds the order of $\log(1/(||E||_2 \,\text{cond}_2 A))$ correct bits per a variable to the current approximate solution to a linear system $A\mathbf{y} = \mathbf{b}$ provided an approximate inverse $(A+E)^{-1}$ of the matrix A is available or just an approximate solution $\tilde{\mathbf{z}} = (A+E)^{-1}\mathbf{v}$ to a linear system $A\mathbf{z} = \mathbf{v}$ is readily available where, say, $||A^{-1}E||_2 < 1/2$.

For another example, preconditioning can accelerate Newton's iteration for the approximation of the inverse as well as the Moore–Penrose generalized inverse of a matrix A because $\log_2 \text{cond}_2 A + \log_2 \log_2(1/\delta) + O(1)$ Newton's iteration steps are sufficient to yield the residual norm bound δ (cf. [9, Chapter 6], [19], and the bibliography therein).

One can enhance the power of the above tecnhiques by combining them together. E.g., given a structured linear systems of equations, one can begin with A-preconditioning of its coefficient matrix, then approximate a generator for the inverse of this matrix by applying the algorithms of the CG/GMRES type, and finally refine the computed approximation by applying a structured version of Newton's iteration [9, Chapter 6].

The progress in solving linear systems of equations can be extended to various related computations. For example, the solution of a polynomial system of equations can be reduced to the solution of sparse multi-level Toeplitz linear systems [20], [21]. 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 CG/GMRES types are not much effective in this case because the matrices are typically ill conditioned and their singular values are widely spread out. Structured APCs of larger ranks promise critical support. We have similar situation with some matrix methods for polynomial root-finding (see the Appendix).

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

With preconditioning we can obtain a 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., but in some areas has not been recognized yet.

E.g., 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" [22, 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.

Likewise, for a large and important class of inputs, preconditioning facilitates certified numerical computation of the sign of the determinant of an ill conditioned matrix, which is required for computing convex hulls, Voronoi diagrams, and in many other fundamental geometric and algebraic computations (see [23] and the bibliography therein).

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

In this section and the next one we facilitate the solution of some fundamental problems of matrix computations provided suitable APPs and the respective A-modifications are available. In this section we deal with the APPs of ranks one and two. We first confine our original numerical problems to computation of and/or operations with some auxiliary matrices of smaller sizes, which we call *aggregates*. We overcome these problems by applying a variant of extended iterative refinement and *MSAs*, which is our abbreviation for advanced *multiplication/summation algorithms*. MSAs can be applied to the evaluation of any polynomial and, in combination with algorithms that approximate reciprocals and with error analysis, can be extended to the approximate evaluation of a rational function, but we use them essentially just for computing sums and dot products. We cover extended iterative refinement and MSAs in Section 6.

3.1 The Schur Aggregation

To solve a nonsingular but ill conditioned linear system $A\mathbf{y} = \mathbf{b}$ as well as to compute the determinant $\det A$, we can first compute a rank-one APC $\mathbf{u}\mathbf{v}^H$ for

the matrix A to obtain its well conditioned A-modification $C = A + \mathbf{u}\mathbf{v}^H$ (cf. Section 1).

Having the A-modification C inverted, we can apply the (primal) SMW inversion and determinant formulae of (1.2) and (1.4), for $U = \mathbf{u}$, $V = \mathbf{v}$, that is

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

and $\det A = (1 - \mathbf{v}^H C^{-1}\mathbf{u}) \det C$, respectively. The formulae reduce the solution of a linear system $A\mathbf{y} = \mathbf{b}$ and the computation of the determinant $\det A$ to well conditioned computations, except for the stage of computing the value $g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}$. The latter value is absolutely small under the above assumptions about the matrices A and C (see Theorem 4.1 in Section 4.1), and so its computation cancels many its leading significant bits. We overcome the problem by extending the iterative refinement algorithm.

The scalar $g = 1 - \mathbf{v}^H C^{-1}\mathbf{u}$ is the Gauss transform of the 2×2 block matrix $\begin{pmatrix} C & \mathbf{u} \\ \mathbf{v}^H & 1 \end{pmatrix}$ and the Schur complement of its block C. For $n > 1$, this scalar is a *Schur aggregate*, and the reduction to its computation from our original task is the *(primal) Schur Aggregation*. We combine such an aggregation technique with A-preconditioning. This is a new feature versus the *aggregation methods* in [24] (our ancestor), which in the 1980s evolved into the *Algebraic Multigrid*.

Now suppose we have an ill conditioned matrix A where $\sigma_1(A) \gg \sigma_2(A)$ and the ratio $\sigma_2(A)/\sigma_n(A)$ is not large. In this case we define the dual A-modification $C_- = A^{-1} + \mathbf{v}\mathbf{u}^H$. According to Section 1, the matrix C_- is likely to be well-conditioned for two random properly scaled vectors $\mathbf{u}$ and $\mathbf{v}$. Instead of its direct computation, however, we first compute its inverse $(C_-)^{-1}$ defined by the dual SMW inversion formula of (1.1) for $U = \mathbf{u}$, $V = \mathbf{v}$, that is

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

For $n > 1$, h is the *dual Schur aggregate*, and its computation is the *dual Schur Aggregation*. We can apply it to the solution of a linear system $A\mathbf{y} = \mathbf{b}$ via equation (1.3) for $U = \mathbf{u}$, $V = \mathbf{v}$, that is

$$\mathbf{y} = A^{-1}\mathbf{b} = \mathbf{z} - \mathbf{v}\mathbf{u}^H\mathbf{b}, \quad (C_-)^{-1}\mathbf{z} = \mathbf{b},$$

as well as to computing the determinant $\det A = (1 + \mathbf{u}^H A\mathbf{v}) \det((C_-)^{-1})$.

Besides computing the reciprocal of the scalar h and the inversion of a well conditioned matrix $(C_-)^{-1}$, we only use additions and multiplications, which we perform error-free by applying MSAs.

3.2 Computations in the null space of a matrix

Given an $n \times n$ matrix A of rank $n-1$, suppose we seek its nonzero null vector $\mathbf{y}$, such that $A\mathbf{y} = \mathbf{0}$. Let a rank-one APP $\mathbf{u}\mathbf{v}^H$ define a nonsingular A-modification $C = A + \mathbf{u}\mathbf{v}^H$. Then $\mathbf{y} = C^{-1}\mathbf{u}$, so that the problem is essentially reduced to solving a nonsingular linear system of equations $C\mathbf{y} = \mathbf{u}$.

For a pair of properly scaled random vectors $\mathbf{u}$ and $\mathbf{v}$, the ratios $\sigma_n(C)/\sigma_{n-1}(A)$ and $(\mathrm{cond}_2\, C)/\,\mathrm{cond}_2\, A$ are likely to be neither large nor small (cf. Section 1). If so, the A-modification C is well conditioned if and only if so is the matrix A. In this case we remove singularity with no numerical sacrifice [25].

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

In this case $A\mathbf{y} = \mathbf{b}$ for the vectors $\mathbf{y} = C^{-1}U\mathbf{x}$ and $\mathbf{x} \neq \mathbf{0}$ such that $AC^{-1}U\mathbf{x} = \mathbf{0}$. We have reduced the search for a null vector $\mathbf{y}$ of an $n \times n$ matrix A to the similar problem for its $n \times 2$ *null aggregate* $AC^{-1}U$. We call this technique the *Null Aggregation*. We have $AC^{-1}U = UG$ where $G = I_2 - V^HC^{-1}U$, and so the original problem can be reduced to the case of 2×2 input if the matrix U has full rank two.

The homogeneous linear systems $AC^{-1}U\mathbf{x} = \mathbf{0}$ or $G\mathbf{x} = \mathbf{0}$ have a nontrivial solution $\mathbf{x}$. Numerically, we would approximate the vector $\mathbf{x}$ by applying the orthogonalization and least-squares methods [7, Chapter 5], [8, Chapter 4], [26], [27], but we must first compute the matrix G with a high precision, overcoming the cancellation of many leading significant bits in its diagonal entries. Numerical benefits and limitations of this approach are quite similar to the case of the Schur Aggregation (see Sections 3.1 and 4.1).

3.3 Extension to the algebraic eigenproblem

An eigenvector of an $n \times n$ matrix A associated with an eigenvalue λ is a null vector of the matrix $\lambda I - A$, and we can incorporate A-preconditioning and the Null Aggregation into the inverse power iteration for refining an eigenvalue/eigenvector pair. (This iteration is also called the inverse iteration and the Rayleigh quotient iteration. We will use the abbreviation *IIRQ*.) The incorporation enables us to replace the solution of an ill conditioned linear system with the solution of a well conditioned one in each IIRQ step. An obvious benefit for sparse and/or structured input matrices A is a chance for applying the GMRES and CG algorithms. In spite of this simplification of every inverse iteration step, we do not need to increase the number of these steps, according to our extensive tests [28]. These techniques effectively approximate a single eigenvalue or a cluster of eigenvalues separated from the other eigenvalues of an input matrix (cf. [29, Section 4.4]).

3.4 Extension to the solution of a linear system of equations

The Null Aggregation can be readily extended to the solution of a nonsingular but ill conditioned linear system of n equations with n unknowns, $A\mathbf{y} = \mathbf{b}$, where

the ratio $\sigma_1(A)/\sigma_{n-1}(A)$ is not large but $\sigma_{n-1}(A) \gg \sigma_n(A)$. The null vector computation is a special case where $\mathbf{b} = \mathbf{0}$, but also vice versa the solution of general linear system $A\mathbf{y} = \mathbf{b}$ is equivalent to computing the null vector $(\tilde{\mathbf{y}}) = \begin{pmatrix} 1 \\ \mathbf{y} \end{pmatrix}$ of the matrix $(-\mathbf{b}, A)$. We can symmetrize this matrix or just append a zero row to it (to yield a square matrix) and then apply the techniques discussed at the end of Section 3.2.

4 Extension to general ill conditioned input matrices

Let us extend our methods to $n \times n$ nonsingular ill conditioned matrices A with $\sigma_{n-k}(A) \gg \sigma_n(A) > 0$ or $\sigma_1(A) \gg \sigma_k(A)$ for $k > 1$. In this case we must use APPs of larger ranks to yield well conditioned A-modifications C or C_-, and so the sizes of the Schur and null aggregates increase. Otherwise the extension is quite straightforward unless we run into ill conditioned aggregates. (Surely this occurs where $\sigma_{i+1}(A) \gg \sigma_i(A)$ for more than one subscript i, but for larger dimensions n also where, say, $2 \le \sigma_{i+1}(A)/\sigma_i(A) \le 3$ for all i, in which case $\mathrm{cond}_2 A \ge 2^{n-1}$.) If so, we must overcome some additional numerical problems.

We outline the respective modifications of our aggregation methods in the next three subsections. As before, we assume square matrices A. We extend this study to the case of rectangular matrices A and cover further technical details in other papers.

4.1 The Schur Aggregation

Suppose we have computed an APC UV^H of a rank $r < n$ and a well conditioned nonsingular A-modification $C = A + UV^H$ for an ill conditioned nonsingular $n \times n$ input matrix A. Now the (primal) SMW inversion formula (1.2) reduces the linear system $A\mathbf{y} = \mathbf{b}$ to the $r+1$ linear systems $C(W, \mathbf{z}) = (U, \mathbf{b})$ with the coefficient matrix given by the A-modification C and to the n linear systems of equations $GX^H = V^H$ with the coefficient matrix $G = I_r - V^H C^{-1} U$. Likewise, equation (1.4) reduces the computation of $\det A$ to computing two matrices C and G and two scalars $\det C$ and $\det G$.

The matrix $G = I_r - V^H C^{-1} U$ is the Gauss transform of the 2×2 block matrix $\begin{pmatrix} C & U \\ V^H & I_r \end{pmatrix}$ and the Schur complement of its block C. For $n > r$, this matrix is a (primal) Schur aggregate and the reduction to its computation from our original task is the (primal) Schur Aggregation.

To analyze the Schur Aggregation, we rely on the following results in [30], which relate the singular values of the matrices A, C, and G to each other. (Here we state these results in a simplified form, for square matrices A and C.)

Theorem 4.1. *[30, Theorem 7.3]. For two positive integers n and $r < n$, a normalized $n \times n$ matrix A, and a pair of $n \times r$ matrices U and V, write*

$C = A+UV^T$ and $G = I_r - V^T C^{-1} U$. Suppose the matrices A and $C = A+UV^T$ have full rank $\rho \geq r$. Then the matrix G is nonsingular, and we have

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

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

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

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

$$||G||_2 = \sigma_1(G) = 1/\sigma_j(G^{-1}) \leq 1/(\sigma_r(A^-)\sigma_-^2(C) - \sigma_-(C)).$$

It follows that the matrix G is well conditioned if so is the matrix C and if the ratio $\sigma_{n-r+1}(A)/\sigma_n(A)$ is not large where $r = \operatorname{rank}(UV^H)$ [30]. In particular numerical problems caused by a gap between the singular values $\sigma_{n-r}(A)$ and $\sigma_{n-r+1}(A)$ are likely to be eliminated for both matrices C and G if the APP of rank r has been chosen according to rules a)–c) in Section 1.

If this gap is the only source of ill conditioning of the matrix A, then the matrices C and G are likely to be well conditioned. If so, the original numerical problems are confined to the computation of the Schur aggregate $G = I_r - V^H C^{-1} U$ of a small norm. We solve these problems by extending iterative refinement and if needed applying MSAs [30], [31].

If the ratio $\sigma_{n-r}(A)/\sigma_n(A)$ is large, then the matrices C and/or G are still ill conditioned. For larger ranks r the A-modification C is likely to be well conditioned, but the aggregate G is not, whereas this property is reversed for smaller ranks r (if $r = 1$, then $\operatorname{cond}_2 G = 1$, but $\operatorname{cond}_2 C$ can be large). We can reapply the Schur Aggregation to such ill conditioned matrices C and/or G, but we must compute these matrices with a higher precision to yield uncorrupted matrices $V^H C^{-1}$, $C^{-1}U$, UG^{-1}, or $G^{-1}V^H$. This would mean a higher computational cost except for matrices G of smaller sizes, which can be inverted efficiently with the GMRES or CG algorithms.

4.2 The dual Schur Aggregation

The dual Schur Aggregation is the Schur Aggregation associated with dual A-preconditioning, that is A-preconditioning of the matrix A^{-1}. Suppose we have computed a dual APC VU^H of a rank $q < n$ and the well conditioned inverse $(C_-)^{-1} = A - AVH^{-1}U^H A$ of the dual A-modification $C_- = A^{-1} + VU^H$ (cf. equation (1.1)), where the matrix $H = I_q + U^H AV$ is the dual Schur aggregate. Then equations (1.1) and (1.3) reduce a linear system $A\mathbf{y} = \mathbf{b}$ to linear systems with the coefficients given by the dual Schur aggregate H, where $\operatorname{cond}_2 H$ has the order of the ratio $\sigma_1(A)/\sigma_q(A)$. Likewise, equation (1.4) reduces the computation of $\det A$ to computing two matrices $(C_-)^{-1}$ and H and two scalars $(\det C_-)^{-1}$ and $\det H$.

Unless the latter ratio is large, the dual Schur aggregate H is well conditioned. Then, similarly to the case of primal Schur Aggregation, the remaining numerical problems are confined to the computation of this aggregate, and we

can overcome them by applying the MSAs. Unlike the primal case, we compute the aggregate H with no matrix inversions and have no need for iterative refinement.

If the ratio $\sigma_1(A)/\sigma_q(A)$ is large, then the dual Schur aggregate H is ill conditioned, and we can apply primal and/or dual A-preconditioning and the pimal and/or dual Schur Aggregation to invert it. (We must invert the aggregate to compute the matrix $(C_-)^{-1}$.) If, however, we run into a new ill conditioned aggregate, we must compute and invert it with a higher precision (at a higher cost) to support the inversion of the ill conditioned matrix H with a high precision.

4.3 Computations in the null space and extensions

Let us first extend our study in Section 3.2. Let A be an $n \times n$ singular matrix that has a rank $n-r$ and thus has the nullity $\operatorname{nul} A = r = n - \operatorname{rank} A$. Suppose UV^H is its (primal) APP of the rank r and the A-modification $C = A + UV^H$ is nonsingular. Then the matrix $V^H C^{-1}$ (resp. $C^{-1}U$) is a left (resp. right) *null matrix basis* for the matrix A, that is the rows (resp. columns) of this matrix span the left (resp. right) null space of the matrix A.

If the singular matrix A is well conditioned, then so is the A-modification C, and we avoid singularity with no numerical sacrifice.

If $\operatorname{rank}(UV^H) = \operatorname{nul} A$ and if the matrix A is ill conditioned, then so is the A-modification $C = A + UV^H$, and we face numerical problems when we test its nonsingularity. To counter them we can keep our APP well conditioned and recursively increase its size. E.g., we can recursively add to the input matrix $C_0 = A$ some suitable rank-one matrices $\mathbf{u}(k)\mathbf{v}(k)^H$, compute the pairs of generator matrices $U_k = (\mathbf{u}(j))_{j=1}^k$ and $V_k = (\mathbf{v}(j))_{j=1}^k$ and the A-modifications $C_k = C_{k-1} + \mathbf{u}(k)\mathbf{v}(k)^H = A + U_k V_k^H$ for $k = 1, 2, \dots$, and output the matrices $U = U_k$, $V = V_k$, $UV^H = U_k V_k^H$, and $C = C_k$ as soon as we arrive at a well conditioned A-modification C_k.

At this point the row (resp. column) span of the null aggregate $V^H C^{-1}$ (resp. $C^{-1}U$) contains the left (resp. right) null space of the matrix A. Moreover, we can obtain a left (resp. right) null matrix basis $Z^H V^H C^{-1}$ (resp. $C^{-1}UX$) for the matrix A as soon as we obtain a left (resp. right) null matrix basis Z^H (resp. X) for the null aggregate $V^H C^{-1} A$ (resp. $AC^{-1}U$), which has a smaller size. Since

$$V^H C^{-1} A = G V^H \quad \text{and} \quad AC^{-1}U = UG \quad \text{for} \quad G = I_r - V^H C^{-1} U, \tag{4.1}$$

we can compute the matrix basis Z^H (resp. X) as a left (resp. right) null matrix basis for the $r \times r$ Schur aggregate G provided the matrix V (resp. U) has full rank. Numerically, we can compute the matrix Z^H (resp. X) by applying the orthogonalization and least-squares methods, but first we must approximate the matrices $V^H C^{-1} A$ (resp. $AC^{-1}U$) with high precision, overcoming the cancellation of many leading significant bits in this process. Then again we apply extended iterative refinement and MSAs. If we need, we can extend the process recursively.

Since the eigenspace associated with an eigenvalue λ of an $n \times n$ matrix A is the null space of the matrix $\lambda I_n - A$, our approach can be readily extended to approximating the eigenvectors and eigenspaces, and we can extend our comments in Section 3.3. If the singular matrix $\lambda I_n - A$ with $\mathrm{nul}(\lambda I_n - A) = r$ is well conditioned or, more generally, if λ represents a cluster of r eigenvalues isolated from the other eigenvalues, then our A-preprocessing of the rank r can eliminate both singularity and ill conditioning.

In particular we can achieve this by incorporating our A-preprocessing into the IIRQ. Conversely, we can apply the inverse iteration for the eigenspace associated with the eigenvalue $\lambda = 0$ to compute or improve approximations to the null vectors and null matrix bases for the matrix A.

As in Section 3.4 we can reduce the solution of a nonsingular linear system $A\mathbf{y} = \mathbf{b}$ to computing the null vector $\begin{pmatrix} 1 \\ \mathbf{y} \end{pmatrix}$ of the matrix $(-\mathbf{b}, A)$.

To describe another extension of the Null Aggregation, suppose we apply it numerically to a singular well conditioned $n \times n$ matrix $\tilde{A}$ with nullity $r = n - \mathrm{rank}\ \tilde{A}$. Then the same output would be computed by the same algorithm performed error-free and applied to a small-norm perturbation $A \approx \tilde{A}$, which is nonsingular but ill conditioned and has exactly r small singular values.

For random and properly scaled APPs UV^H of rank r and for $C = A + UV^H$, the ranges of the aggregates $V^H C^{-1}$ and $C^{-1}U$ (that is their respective row and column spans) would approximate the *r-tail* of its SVD, that is the singular spaces associated with the r smallest singular values of the matrix A. We call this technique the *Tail Approximation*.

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

Finally, if $\mathrm{cond}_2 C$ for an A-modification $C = A + UV^H$ is too large, we can decrease it by recomputing the APP. We can simply rely on rules a)–c) for choosing the new APPs, but alternatively (with more work and more confidence in success) we can apply the Null/Tail Approximation to improve an APC UV^H as follows,

$$(U \leftarrow Q(C^{-1}U), \quad V \leftarrow Q(C^{-H}V)). \tag{4.2}$$

Here $Q(M)$ denotes the $k \times l$ Q-factor in the QR factorization of a $k \times l$ matrix M of the full rank.

Computation of the aggregates $C^{-1}U$ and $C^{-H}V$ is simpler where the matrix C is better conditioned, and we can more readily obtain a well conditioned A-modification $C = A + UV^H$ by choosing an APP UV^H of a larger rank. Then we can extend the transform in (4.2) to obtain an APC of a smaller rank for which we still have $\mathrm{cond}_2 C$ nicely bounded (cf. [4]–[6], [25]), [32]. Specifically, assuming that the ratio $\sigma_1(A)/\sigma_{n-r}(A)$ is not large, whereas $\sigma_{n-r}(A) \gg \sigma_{n-r+1}(A)$ for an $n \times n$ ill-conditioned input matrix A, we can proceed as follows.

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

2. *(The Tail Approximation.)* Compute two properly scaled and well conditioned matrix bases $T(U)$ and $T(V)$ for the singular spaces of the matrices $AC^{-}U$ and $A^HC^{-H}V$, respectively, associated with the r smallest singular values of these matrices. If A is a square matrix and if U and V are unitary matrices, then the matrices $T(U)$ and $T(V)$ can be also computed as two matrix bases for the left and right singular spaces associated with the r smallest singular values of the matrix $G = I_h - V^HC^{-}U$ (cf. equation (4.1)).

3. *(Compression.)* Update the generators $U \leftarrow Q(C^{-1}UT(U))$ and $V \leftarrow Q(C^{-H}VT(V))$. Output them and the new APC UV^H.

The efficiency of these recipes has been confirmed by extensive tests in [4]–[6], [32].

5 Aggregation for structured matrices

Unlike [24], *all our present nonrecursive aggregation methods allow us to preserve matrix structure.* More precisely, if an input matrix A has the displacement or semiseparable structure [9], [14], then we can choose a pair of generators U and V with consistent structure [6, Examples 4.1–4.6] and preserve it in a few matrix additions, multiplications, and inversions required for the transition to the APP UV^H, the A-modification $C = A + UV^H$, and the aggregates V^HC^{-1} and $C^{-1}U$.

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

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

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

6 Extended iterative refinement and MSAs

For an ill conditioned matrix A and a well conditioned matrix C, the primal Schur Aggregation leads us to the task of computing the Schur aggregates $G = I - V^HC^{-1}U$ that have very small norms $||G||_2$ (see Section 4.1). This means that many leading significant bits of the output entries are cancelled, which poses a numerical challenge. We meet it by extending the iterative refinement algorithm and if needed using MSAs. (Similar problems arise and similar

solution recipes can be applied in the case of the dual Schur Aggregation, except that in that case we less depend on iterative refinement.)

We extend iterative refinement to compute the matrix $W = C^{-1}U$ with high accuracy. (We can compute the matrix $V^H C^{-1}$ instead). We do not store the computed segments of bits of the binary representation of the entries of the matrix W but immediately employ them into the multiplication $V^H W$, and store the respective segments that represent the entries of the matrix $G = I_r - V^H C^{-1} U$. More precisely, we begin storing these segments as soon as we arrive at a nonvanishing approximation to the matrix G that remains stable in some consecutive steps of iterative refinement. Since the norm $||G||_2$ is small, the segments of the leading bits vanish in a number of the initial steps of the refinement.

We can choose any pair of matrices U and V up to a perturbation within a fixed small norm as long as this perturbation keeps the A-modification $C = A + UV^H$ well conditioned. Likewise, in our computation of the matrices $W_i = C^{-1}U_i$ we can allow any errors within a fixed small norm bound as long as this ensures that the residual norm $u_i = ||U_i||_2$ decreases by at least a fixed factor $1/\theta > 1$ in each iteration.

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

Apart from the approximation of the matrices C^{-1} and W_i within some fixed error norm bounds, we perform all other arithmetic operations error-free, that is we allow no errors at the stages of computing the matrices $C \leftarrow A + UV^H$, $U_{i+1} \leftarrow U_i - CW_i$, $F_i \leftarrow -V^T W_i$, and $G_{i+1} \leftarrow G_i + F_i$ for $i = 0, 1, \ldots, k$. At these stages, computing with the double precision can be insufficient for some input matrices, but we can meet the challenge by applying the MSAs, which we outline next and cover in more details in [31].

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

Our MSAs combine the Dekker's and Veltkamp's algorithms in [33] to compute the product of a dpn-μ and a dpn-ν error-free as a dpn-γ for $\gamma \le 2\mu\nu$. To add a dpn-μ and a dpn-ν we just combine them into a dpn-$(\mu + \nu)$.

For the reader's convenience, here is the Dekker's basic algorithm in [33] for splitting a floating-point number into two parts. We assume that g is an integer, $0 < g \le p$.

Algorithm 6.1. Splitting of a floating-point number into two parts.

$$\begin{aligned} \text{function}[x,y] &= \text{Split}(a) \\ c &= \text{fl}(\text{factor}\cdot a) = \text{fl}(2^g a + a) \qquad \% \;\; \text{factor} = 2^g + 1 \\ x &= \text{fl}(c-(c-a)) \\ y &= \text{fl}(a-x) \end{aligned}$$

The shorter precision numbers x and y satisfy the equation $a = x+y$. Under the common assumption that $0 \le \lceil p/2 \rceil - g \le 1$, these are the half-precision numbers.

We use this algorithm also in our *compressing summation*, which rewrites a dpn-μ as a dpn-ν for a nearly minimum ν. We must perform this operation as soon as we encounter a *swelling sum*, that is a dpn-ν for a too large value ν. This is particularly important in the computation of the residuals in the extended iterative refinement.

To perform this operation we adjust and advance the approach in [34] (also covered in [35]), which relies on the *real modular reduction*, that is modular reduction over the real numbers. Namely, for real s and $t \ne 0$, we write $s \mod t$ to denote a unique real q such that t divides $s-q$, $|q|$ is minimum, and $q \ne t/2$, so that $(t/2) \mod t = -t/2$. We have the following simple lemma.

Lemma 6.1. *For three real numbers a, b, and $t \ne 0$, we have $(a \mod t)(\text{rop})(b \mod t) \mod t = (a(\text{rop})b) \mod t$ where (rop) denotes a ring operation, that is addition, subtraction, or multiplication.*

To explain how this technique can help us, we first recall the following upper estimates for the errors of some customary floating-point summation algorithms that compute approximations s to the sum s^* of h numbers $s_1, \dots, s_h$, where $s_j = \sigma 2^{e_j} f_j$, $\sigma = 1$ or $\sigma = -1$, e_j are integers, and f_j are fractions, $0.5 \le f_j < 1$ for $j = 1, \dots, h$,

$$|s - s^*| \le \delta_+ = e s_+, \tag{6.1}$$

$$s = \text{fl}(s_1 + \cdots + s_h),\;\; s_+ = \text{fl}(|s_1| + \cdots + |s_h|), \tag{6.2}$$

$$e \le (1 + c\gamma)c\gamma,\;\; \gamma = h/(2^{p+1} - h). \tag{6.3}$$

Here $c = 1.12$ for the customary floating-point summation in [8, Section 2.4] provided $h < 0.1\ 2^{p+1}$. By applying the summation algorithms in [36] one can decrease the parameter c to one provided $h < 2^{p+1}$ (cf. [36, Lemma 4.2]). The more advanced Algorithm 4.4 in [36] supports the improved error bound

$$|s_1 + \cdots + s_h - s| = es + e\gamma s_+ \tag{6.4}$$

under (6.2) and (6.3) where $c = 1$ and $h < 2^{p+1}$ [36, Proposition 4.5].

Now, suppose the floating-point summation of h summands $s_1, \dots, s_h$ has output s such that $\delta_+ > |s|/2$ for the error bound δ_+ estimated based on equations (6.1)–(6.4). Clearly, this can only occur because $|s| \ll \max_j |s_j|$. Now the recipe in [34] and [35] is to reduce the summands modulo 2^d for

$$d = 3 + \lfloor \log_2 \tilde{s} \rfloor,\;\; \tilde{s} \ge |s| + \delta_+ \tag{6.5}$$

and to repeat the computations. Here $1+\lfloor \log_2 \tilde{s} \rfloor$ is the exponent of the floating-point representation of the number $\tilde{s}$, and so we can still readily recover this number from the sum of the residues modulo 2^d of the original summands s_j. Our gain is the decrease of the values $\max_j |s_j|$, s_+, and δ_+.

The process can be repeated recursively until we approximate the sum $s_1 + \ldots + s_h$ with the relative error bound of at most $1/2$. Then the exponent d in equation (6.5) decreases. More precisely, when we replace all summands by their residues modulo 2^d, we decrease their absolute values below $3s^*$. Now, by applying the customary floating-point summation algorithms, we obtain approximation s to the sum s^* within a relative approximation error below 2^{-k} where $k > \theta p - \log_2 h - O(1)$, $1 + p$ is the precision of computing, $\theta = 1$ if we rely on the floating-point summation algorithms supporting bounds (6.1)–(6.3), whereas $\theta = 2$ if we rely on the floating-point summation Algorithm 4.4 in [36], supporting bound (6.4).

Finally we can reapply the algorithm to approximate the error $s_1+\cdots+s_{h+1}$ for $s_{h+1} = -s$ and recursively refine the approximation to any fixed tolerance level.

To implement this approach semi-numerically, we can perform real modular reduction by adapting the Dekker's splitting algorithm, but we prefer to replace the real modular reduction itself with this splitting, to deviate less from the pure floating-point computations. Namely, our algorithm for the summation of h numbers performs only floating-point operations except that it accesses the exponent d of equation (6.5) as soon as the current approximation s to the sum s^* is updated. We easily deduce that in every updating the exponent d decreases by at least $k \geq \theta p - \log_2 h - O(1)$, whereas the absolute value of the approximation s and the error bound δ_+ decrease by the factor 2^k.

As soon as we obtain the exponent d in equation (6.5), we apply Dekker's algorithm for $a = s_j$ and $g = e_j + p + 1 - d$ to split every summand s_j into two subsummands, x_j and y_j such that $x_j + y_j = s_j$ and the subsummand x_j is obtained by rounding the summand s_j to the level of 2^d.

Then we reapply the algorithm to compute the sum $x = x_1 + \ldots + x_h$. We observe that $|x|2^d$ is an integer not exceeding h and thus has at most $\lceil \log_2 h \rceil$ nonzero bits. Therefore, the sum x can be readily computed error-free.

At this point we rewrite our sum $s_1 + \cdots + s_h$ as $x + y_1 + \cdots + y_h$ where $|x| < (h+1)2^d$ and $|y_j| < 2^g$ for all j. Now we reapply our algorithm to compute the sum within the relative error of at most 2^{-k} for $k \geq \theta p - \log_2 h - O(1)$. This leads to the same results as in the approach based on the real modular reduction, but now, besides the floating-point operations, we just periodically access the exponent d.

We perform this operation rarely, and moreover its cost is not high. The IEEE floating point standard defines the function *logb(x)* to extract the significand and exponent of a floating point number. Floating point units (FPUs) in Intel's Pentium processor family provide hardware implementations of an instruction, FXTRACT, offering a superset of the *logb(x)* functionality [37]. For double precision floating point numbers, the FPU of the Pentium 4 processor can execute the FXTRACT instruction in 12 cycles [38] (almost three times as

fast as the same FPU handles division). Because FXTRACT is a floating point instruction, the FPU can overlap the early cycles of FXTRACT with late cycles of various other floating point instructions when they immediately precede FXTRACT, thereby allowing further speed up [38].

The construction is most effective for computing a dot product whose absolute value is much less than that of its absolutely maximal term. In this case (precisely of our current concern) many leading significant bits of the output value are cancelled. Our MSAs determine these bits and cancel them. Then the routine summation with double precision outputs the sum with a high precision.

7 The preceding study

Small-rank modification is a known tool for decreasing the rank of a matrix [39], [40], fixing its small-rank deviations from the Hermitian, positive definite, and displacement structures, and supporting the divide-and-conquer eigen-solvers [7], [41], [29], but these isolated efforts have not been identified as additive preconditioning in matrix computations. The discussions that followed the presentations by the first author at the International Conferences on the Matrix Methods and Operator Equations in Moscow, Russia, in June 20–25, 2005, and on the Foundations of Computational Mathematics (FoCM'2005) in Santander, Spain, June 30–July 9, 2005, revealed only a few other touches to what we call A-preconditioning. They were sporadic and rudimentary versus our present work. We are aware of no earlier use of the nomenclature of A-preconditioning and APCs as well as of no attempts of devising and employing random and/or structured primal and dual APCs, studying APCs and their affect on the rank and conditioning systematically, relating the APCs to aggregation, iterative refinement, and MSAs, improving APCs based on the Null Aggregation, or applying them to numerical approximation of the bases for trailing singular spaces of ill conditioned matrices.

We arrived at A-preconditioning while applying the inverse (Rayleigh quotient) iteration for the algebraic eigenproblem with semiseparable input to polynomial root-finding (see the Appendix). While elaborating upon the application of APCs to eigen-solving, we also observed applications to the null space computations, constructed random and structured pseudo random ACs and APCs, estimated their affect on conditioning, defined various classes of primal and dual APCs, and devised all our aggregation techniques for the transition from APCs to the solution of linear systems and computing determinants.

8 Further research subjects

We have introduced the large new areas of A-preconditioning and aggregation and have demonstrated their great potential impact on some most fundamental matrix computations. Clearly, many subjects in these areas invite further theoretical and experimental study, e.g.,

- the approximation and error analysis for the Tail and Head Approximation based on A-preconditioning
- the analysis of recursive numerical application of the Schur and Null Aggregation to singular and nonsingular ill conditioned matrices having multiple jumps in the spectrum of their singular values
- the comparison in efficiency of the primal and dual Schur Aggregation with each other and with the Null Approximation as well as the combined application of these techniques to solving linear systems of equations
- decreasing the running time and memory space in our MSAs.

Recalling that the aggregation methods in [24], based on M-preconditioning, evolved into the Algebraic Multigrid in the 1980s, we now ask whether our A-preconditioning and aggregation methods will eventually evolve into A-Algebraic Multigrid. Will they be ever extended to yield other classes of effective preconditioning and/or aggregation methods?

Among nontrivial techniques of this kind that could be pointers to such extensions, we recall *trilinear aggregating* in [42]. This technique has been an indispensible ingredient in the design of the currently fastest algorithms for $n \times n$ matrix multiplication. This includes the fastest known algorithms for both immense dimensions n [43] and moderate dimensions n from 20 to, say, 10^{20} [42], [44], in which case we have efficient numerical implementations [45], [46].

Our further research directions also include applications to solving systems of multivariate polynomial equations. We refer the reader to [47] on application of A-preconditioning and the Schur Aggregation to computing determinants.

Acknowledgements. E. E. Tyrtyshnikov, S. A. Goreinov, and N. L. Zamarashkin from the Institute of Numerical Analysis of the Russian Academy of Sciences in Moscow, Russia, and B. Mourrain from the INRIA in Sophia Antipolis, France, provided the first author of this paper with the access to the computer and library facilities during his visits to their Institutes in 2005/06. X. Wang was the first reader of our papers on A-preconditioning and aggregation and has replied with his new advance in [32]. Helpful and encouraging were the interest and comments to our work from the participants of the cited Conferences in Moscow and Santander (particularly from J. W. Demmel, G. H. Golub, V. Olshevsky, L. Reichel, M. Van Barel, and a participant of FoCM'2005 who proposed the substitution of APCs for pivoting in the Gaussian elimination algorithm).

Appendix. Application to polynomial root-finding

Polynomial root-finding is an example of further applications of A-preconditioning and aggregation. This is a classical and highly developed subject but

is still an area of active research [48], [49]. Increasingly popular are the matrix methods for appriximating the polynomial roots as the eigenvalues of an associated *generalized companion* matrix. Matlab computes relatively crude approximations to the polynomial roots by applying the QR eigen-solver to the Frobenius companion matrix. Malek and Vaillantcourt in [50] and [51] and Fortune in [52] recursively apply the QR algorithm to a diagonal plus rank-one (hereafter *DPR1*) generalized companion matrix, updating it whenever new approximations to the roots are computed. This process turns out to be highly effective.

In [53] similar approach employs the IIRQ iteration instead of the QR iteration. This decreases the running time per iteration step from quadratic to linear due to the structure of the DPR1 matrices (and similarly for the Frobenius matrices and for the shift-and-invert enhancement of the Lanczos, Arnoldi, Jacobi–Davidson, and other eigen-solvers). The idea of exploiting DPR1 matrix structure for polynomial root-finding first succeeded in [53] and then again in [54] and [55] (cf. also [56]), which caused substantial interest among the experts on semiseparable matrices.

According to the test results in [53] the IIRQ iteration is quite effective for the DPR1 and Frobenius matrices, so that the algorithm is already slightly superior to the Durand–Kerner's (Weierstrass') celebrated root-finder. Application of A-preconditioning and aggregation should further enhance the power of this approach.

References

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

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

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

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

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

[6] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning for Matrix Computations, Technical Report TR 2007

003, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

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

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

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

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

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

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

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

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

[15] 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.

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

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

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

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

[20] 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.

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

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

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

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

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

[26] C. L. Lawson, R. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974. Reissued with a survey of recent debvelopments by SIAM, Philadelphia, 1995.

[27] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

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

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

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

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

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

[33] T. J. Dekker, A Floating-point Technique for Extending the Available Precision,

[34] 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.

[35] 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.

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

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

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

[39] 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.

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

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

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

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

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

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

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

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

[48] *NAG Fortran Library Manual*, Mark 13, Vol. **1**, 1988.

[49] V. Y. Pan, Solving a Polynomial Equation: Some History and Recent Progress, *SIAM Review*, **39, 2**, 187–220, 1997.

[50] F. Malek, R. Vaillancourt, Polynomial Zerofinding Iterative Matrix Algorithms, *Computers and Math. (with Applications)*, **29**, **1**, 1–13, 1995.

[51] F. Malek, R. Vaillancourt, A Composite Polynomial Zerofinding Matrix Algorithm, *Computers and Math. (with Applications)*, **30**, **2**, 37–47, 1995.

[52] S. Fortune, An Iterated Eigenvalue Algorithm for Approximating Roots of Univariate Polynomials, *J. of Symbolic Computation*, **33**, **5**, 627–646, 2002. (Proc. version in *Proc. Intern. Symp. on Symbolic and Algebraic Computation (ISSAC'01)*, 121–128, ACM Press, New York, 2001.)

[53] 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, January 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.)

[54] 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.)

[55] 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.

[56] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Wang, X. Yan, Root-finding with Eigen-solving, pages 219–245 in *Symbolic-Numerical Computation*, (Dongming Wang and Lihong Zhi editors), Birkhäuser, Basel/Boston, 2007.