2007

TR-2007009: Computations in the Null Spaces with Additive Preprocessing

Victor Y. Pan

Computations in the Null Spaces with Additive Preprocessing *

Victor Y. Pan

Department of Mathematics and Computer Science
Lehman College of the City University of New York
Bronx, NY 10468 USA
victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/

Abstract

We propose and analyze additive preprocessing for computing the vectors in and bases for the null spaces of matrices. Instead of singular linear systems we solve nonsingular ones that preserve the conditioning properties and the structure of the input matrices. For ill conditioned input we can extend our preprocessing further, to decrease the problem size. Our approach is readily extended to the eigenspace and singular space computations and to solving linear systems of equations.

Key words: Matrix computations, Additive preconditioning, Null space approximation

1 Introduction

Suppose we seek a vector in or a basis for the (right) null space $RN(A)$ of an $m \times n$ matrix A of a rank ρ. (We call them a *null vector* and a *null basis* for this matrix.) We can obtain them via computing its SVD, QRP or PLU factorizations [1]–[4] (cf., e.g., effective Algorithm 5.3.2 in [1]), or the inverse of a nonsingular $\rho \times \rho$ submatrix W of matrices A or MAN for some nonsingular multipliers M and N.

We, however, study an alternative approach based on *addititive preprocessing* of the input matrix A. Hereafter we use the abbreviation "A-" for "additive" and "APP" for "additive preprocessor". For simplicity let $m = n$, write $\operatorname{nul} A = n - \operatorname{rank} A$ to denote the *nullity* of the matrix A and write M^H to denote the

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

Hermitian (that is complex conjugate) transpose of a matrix M, which is just its transpose M^T for a real matrix M.

Now for two *generators* U and V of size $n \times r$, suppose an APP UV^H has rank $r = \text{nul}\, A$ and the A-modification $C = A + UV^H$ has full rank. Then the range (that is the span of the columns) of the matrix $C^{-1}U$ is a basis for the null space $RN(A)$ (see Theorem 3.1), so that our preprocessing reduces singular computations to nonsingular ones.

Our approach can be naturally extended to the approximation of the eigenspaces because the eigenspace associated with an eigenvalue λ of a matrix M is the null space of the matrix $\lambda I - M$ (see [5]). We also show the extensions to solving linear systems of equations and to generating and refining additive preconditioners, for which we use the abbreviation *APCs*.

The approach preserves the structural and conditioning properties of the input matrix and thus supports application of the Conjugate Gradient and GMRES algorithms provided the input matrix is well conditioned.

If it is ill conditioned, we can extend our additive preprocessing to *additive preconditioning* in [6]–[9] and combine it with the technique of the *Null Aggregation*, which we relate to the *Schur Aggregation* in [10]. Both aggregation techniques confine the remaining numerical problems to the matrices of smaller size, which we call *aggregates*. We handle these problems in [10] based on extending the Wilkinson's iterative refinement from [1, Section 3.5.3], [2, Sections 3.3.4 and 3.4.5], and [11, Chapter 11] and applying advanced algorithms for error-free multiplication and summation, which we call *MSAs* (see [10], [12]).

We organize our paper as follows. After presenting some definitions and preliminaries in the next section and our basic theorem in Section 3, we compute the nullity and right null bases in Sections 4–6. In Section 7 we compute left null bases, relate the null spaces for the square and rectangular inputs, and observe a link of our computations to the Schur Aggregation in [10]. In Section 8 we simplify our algorithms in the case where we seek just a single null vector. In Section 9 we extend the null space algorithms to solving linear system of equations. We assume error-free computations in Sections 4–9, which together with Section 3 make up Part I of the paper.

The shorter Part II of the paper is made up of Sections 10, 11, and the Appendix. In Section 10 we comment on numerical implementation of our null space algorithms and approximate some bases for the singular spaces associated with an isolated cluster of the smallest singular values of an ill conditioned matrix or its inverse. In Section 11 we cover the extension to generating and refining effective APCs. The Appendix shows some estimates for the affect of matrix perturbation on the null space.

2 Basic definitions

2.1 Some customary definitions

ϵ denotes the *unit roundoff* (*machine epsilon*).

Most of our next basic definitions reproduce or slightly modify the customary definitions for matrix computations in [1]–[4], [13], [14]. This includes the definitions of the Hermitian, unitary, singular, full-rank and rank deficient matrices, the $k \times k$ identity matrices I_k, $k \times l$ matrices $0_{k,l}$ filled with zeros, the 2-norm $||A||_2$ and Frobenius norm $||A||_F$ of an $m \times n$ matrix A, its condition number $\mathrm{cond}_2 A$, its range $\mathrm{range}\, A$ (the span of its column vectors), its (right) null space $N(A) = RN(A)$ and left null space $LN(A)$, its rank $\rho = \mathrm{rank}\, A$, left nullity $\mathrm{lnul}\, A = m - \rho$, right nullity $\mathrm{rnul}\, A = n - \rho$, and nullity $\mathrm{nul}\, A = \min\{m, n\} - \rho$.

A matrix A is normalized if $||A||_2 = 1$.

$\mathrm{diag}(B_1, B_2)$ (resp. $\mathrm{diag}(B_i)_{i=1}^k$) is the 2×2 and $k \times k$ block diagonal matrix with the diagonal blocks B_1 and B_2 (resp. $B_1, \ldots, B_k$).

(B, C) is the 1×2 block matrix with the blocks B and C.

A matrix A is a *matrix basis* for its range if its column set is linearly independent. A null vector, a *null basis*, and a *null matrix basis* for a matrix is a vector, a basis, and a matrix basis for its (right) null space, respectively. Similar concepts are defined for the left null space.

We write $Q(M)$ for the $m \times n$ Q-factor in the QR factorization of an $m \times n$ matrix M of the full rank.

The *Singular Value Decomposition* (hereafter *SVD*, also called the *full SVD*) of an $m \times n$ matrix A of a rank ρ, is given by the equation $A = S\Sigma T^H$ where $S = (\mathbf{s}_j)_{j=1}^m$ and $T = (\mathbf{t}_j)_{j=1}^n$ are square unitary matrices; $\Sigma = \mathrm{diag}(\Sigma^{(\rho)}, 0_{l,r})$ is an $m \times n$ matrix; $l = \mathrm{lnul}\, A = m - \rho$, $r = \mathrm{rnul}\, A = n - \rho$, $\Sigma^{(\rho)} = \mathrm{diag}(\sigma_j)_{j=1}^\rho$ is a diagonal matrix; $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho > 0$, and $\sigma_j = 0$ for $j > \rho$, and we write $\sigma_j = +\infty$ for $j < 1$. The scalars σ_j for $j \geq 1$ are the *singular values* of the matrix A, and $||A||_2 = \sigma_1$. The vectors $\mathbf{s}_j$ for $j = 1, \ldots, m$ and $\mathbf{t}_j$ for $j = 1, \ldots, n$ are the associated left and right *singular vectors*, respectively, so that the null vectors are the singular vectors associated with the singular value zero. The singular vectors associated with a fixed singular value or a fixed set of them form the associated singular spaces. The decomposition $A = S^{(\rho)}\Sigma^{(\rho)}T^{(\rho)H} = \sum_{j=1}^\rho \sigma_j \mathbf{s}_j \mathbf{t}_j^H$ is the *compact SVD* of the matrix A.

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 [1], and we consistently write A^{-H} for $(A^H)^- = (A^-)^H$. We have $A^- = A^{-1}$ if $m = n = \rho$,

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

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

$\mathrm{cond}_2 A = \sigma_1(A)/\sigma_\rho(A) = ||A||_2\, ||A^-||_2$ is the condition number of a matrix A of a rank ρ.

We write $n \gg d$ where the ratio n/d is large. A matrix A of a rank ρ is *ill conditioned* if $\sigma_1 \gg \sigma_\rho$ and is *well conditioned* otherwise. A matrix A of any rank $\rho > 1$ can be ill conditioned where $\sigma_j(A) \gg \sigma_{j+1}(A)$ for some j, $1 \leq j < \rho$, but for a larger rank ρ it can be ill conditioned even if $\sigma_j(A)/\sigma_{j+1}(A) \leq c$ for all j and a smaller bound c. E.g., we can have $\mathrm{cond}_2 A = 2^{100}$ for $c = 2$ and

$\rho = 101$. Effective norm and condition estimators can be found in [1, Section 3.5.4] and [2, Section 5.3].

2.2 Additive preprocessing

Hereafter "A" stands for "additive", "ACs" for "additive complements", "APPs" for "additive preprocessors", "APCs" for "additive preconditioners".

Definition 2.1. *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, the matrix $C = A + UV^H$ is its* A-modification, *the matrices U and V are* generators *of the APP, and the transition $A \leftarrow C$ is* A-preprocessing of rank r *for the matrix A. An APP UV^H for a matrix A is an* APC *and A-preprocessing is* A-preconditioning *if* $\mathrm{cond}_2 A \gg \mathrm{cond}_2 C$. *An APP is an* AC *and A-preprocessing is* A-complementation *if the matrix A is rank deficient, whereas the A-modification C has full rank. An APP UV^H is* unitary *if the matrices U and V are unitary.*

2.3 The g-heads, (extended) h-tails, and (g, h) matrices

Preconditioning is linked to the parts of the SVD represented by its largest and its smallest singular values, and its effect is measured in terms of the ratios of singular values. This motivates our next definitions.

The *g-head* (resp. *h-tail*) of the SVD of a matrix is a triple made up of its g largest (resp. h smallest positive) singular values and of the pair of matrix bases for the associated left and right singular spaces. By combining the h-tail with the null matrix bases, we arrive at the *extended h-tail*.

Formally, the g-head, h-tail, and extended h-tail of the SVD are the triples

$$(S^{(g)}, \Sigma^{(g)}, T^{(g)}), \quad (S_h, \Sigma_h, T_h), \quad \text{and} \quad (S_h^{(\mathrm{ext})}, \Sigma_h^{(\mathrm{ext})}, T_h^{(\mathrm{ext})}),$$

respectively, where g and $h, \rho - g$ are two nonnegative integers, $S^{(g)} = (\mathbf{s}_j)_{j=1}^{g}$,

$$S_{g,h} = (\mathbf{s}_j)_{j=g+1}^{\rho-h}, \quad S_h = (\mathbf{s}_j)_{j=\rho-h+1}^{\rho}, \quad S_h^{(\mathrm{ext})} = (S_h, S^{(nul)}) = (\mathbf{s}_j)_{j=\rho-h+1}^{m},$$

$$T^{(g)} = (\mathbf{t}_j)_{j=1}^{g}, \quad T_{g,h} = (\mathbf{t}_j)_{j=g+1}^{\rho-h}, \quad T_h = (\mathbf{t}_j)_{j=\rho-h+1}^{\rho},$$

$$T_h^{(\mathrm{ext})} = (T_h, T^{(nul)}) = (\mathbf{t}_j)_{j=\rho-h+1}^{n},$$

$$\Sigma^{(g)} = \mathrm{diag}(\sigma_j)_{j=1}^{g}, \Sigma_{g,h} = \mathrm{diag}(\sigma_j)_{j=g+1}^{\rho-h},$$

$$\Sigma_h = \mathrm{diag}(\sigma_j)_{j=\rho-h+1}^{\rho}, \quad \Sigma_h^{(\mathrm{ext})} = \mathrm{diag}(\Sigma_h, 0_{l,r}),$$

and $M^{(0)}$ and M_0 are empty matrices for M denoting S, Σ, or T.

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 the matrix A, respectively.

A matrix A of a rank ρ is a (g,h) *matrix* if $g+h<\rho$, $\sigma_1 \gg \sigma_{g+1}$, $\sigma_{\rho-h+1} \gg \sigma_\rho$, and the ratio $\sigma_{g+1}/\sigma_{\rho-h+1}$ is not large. A matrix is a *strictly* (g,h) *matrix* if $\sigma_g \gg \sigma_{g+1}$, $\sigma_{\rho-h+1} \gg \sigma_{\rho-h}$, and the ratios σ_1/σ_g, $\sigma_{g+1}/\sigma_{\rho-h+1}$, and $\sigma_{\rho-h}/\sigma_\rho$ are not large.

In this paper the (g,h) matrices are usually $(g,0)$ or $(0,h)$ matrices. For a strictly $(0,h)$ matrix A, the integer h is its *numerical nullity*, to be denoted $\operatorname{nnul} A$.

2.4 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 $k\times l$ *random unitary* matrix is the $k\times l$ Q-factor $Q(M)$ in the QR factorization of random $k\times l$ matrix M of full rank.

PART I. THE ERROR-FREE NULL SPACE COMPUTATIONS

3 The basic theorem

To compute null bases, we rely on a simple theorem that we next sketch (together with some corollaries) and then state and prove formally. In our sketch we write $C=A+UV^H$ 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 3.1. *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*

$$r\ge \operatorname{rank} U\ge n-\rho=\operatorname{rnul} A=\operatorname{nul} A, \tag{3.1}$$

$$N(A)\subseteq \operatorname{range}(C^-U). \tag{3.2}$$

Furthermore if

$$r=\operatorname{rank} U=n-\rho=\operatorname{rnul} A=\operatorname{nul} A, \tag{3.3}$$

then we have

$$N(A) = \text{range}(C^{-}U), \tag{3.4}$$

$$V^H C^{-} U = I_r. \tag{3.5}$$

Proof. Bound (3.1) follows because $\text{rank}(B + C) \le \text{rank}\, B + \text{rank}\, C$. 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)),

$$\mathbf{y} = C^{-}U(V^H\mathbf{y}). \tag{3.6}$$

This proves (3.2).

(3.4) immediately follows from (3.2) and (3.3).

To prove (3.5), pre-multiply equation (3.6) by V^H, recall equation (3.4), and deduce that $(V^H C^{-} U - I_r)V^H C^{-} U = 0$. Now (3.5) follows unless the matrix $V^H C^{-} U$ is singular, but if it is, then $V^H C^{-} U\mathbf{z} = \mathbf{0}$ 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. □

Corollary 3.1. *Under the assumptions of Theorem 3.1 let equations (3.1) and (3.2) hold. Then* $N(A) = \text{range}(C^{-}UX)$ *if* X *is a matrix bases for the null space* $N(AC^{-}U)$.

4 Right null bases via A-preprocessing

Next, in two sections, we describe six algorithms that output right null matrix bases. Then in Section 6 we summarize the algorithms in a single flowchart, assuming rectangular input matrices. In Section 7 we cover the straightforward extension to the computations in the left null space, some routine reductions to the case of square matrices, and some simplifications of the algorithms in this case.

Our algorithms rely on Theorem 3.1 and Corollary 3.1 and employ two black box Subroutines **AC** and LIN·SOLVE.

The Subroutine **AC** has an $m \times n$ matrix A and a nonnegative integer r' as its input. The subroutine always fails if $r' < \text{nul}\, A$. Otherwise it either fails with a low probability or outputs a pair of generator matrices U and V of full rank, an AC UV^H of a rank r such that $\text{nul}\, A \le r \le r'$, and the A-modification $C = A + UV^H$ of full rank. The algorithms in [8, Section 4.4 and Examples 4.1–4.6] can serve as such subroutines.

The Subroutine LIN·SOLVE is applied to an $m \times h$ matrix B and an $m \times n$ matrix C. The subroutine outputs FAILURE if the matrix C is rank deficient. Otherwise it outputs the $n \times h$ matrix $C^{-}B$.

Correctness of our first algorithm is implied by equations (3.3) and (3.4).

Algorithm 4.1. 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, a positive integer $r' = \operatorname{nul} A$, and Subroutines* **AC** *and LIN·SOLVE.*

OUTPUT: *FAILURE or a unitary matrix basis for the null space $N(A)$.*

COMPUTATIONS:

1. *Apply Subroutine* **AC**. *Output FAILURE if so does the Subroutine.*
2. *Otherwise apply Subroutine LIN·SOLVE to compute the matrix $C^{-}U$.*
3. *Compute and output the Q-factor of the matrix $C^{-}U$.*

In our two next algorithms, we apply Algorithm 4.1 to the values r' that we recursively update (up or down) until our test shows that we have yielded $r' = \operatorname{nul} A$. In Algorithm 4.2 we increase these values beginning with $r' \leq \operatorname{nul} A$, e.g., $r' = 0$, and stop where the matrix C has full rank. In Algorithm 4.3 we decrease these values beginning with $r' \geq \operatorname{nul} A$, e.g., $r' = n$, and stop where $AC^{-}U = 0$. Correctness of Algorithms 4.1 and 4.2 follows from Theorem 3.1 (see equations (3.3) and (3.4)) and Corollary 3.1.

Algorithm 4.2. A null basis with the full-rank certification.

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

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

COMPUTATIONS: *Stages 1 and 2 as in Algorithm 4.1.*

3. *$r' \leftarrow r + 1$ and reapply the algorithm if Subroutine LIN·SOLVE outputs FAILURE.*
4. *Otherwise output r. Compute and output the Q-factor of the matrix $C^{-}U$.*

Algorithm 4.3. 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 $r' \geq \operatorname{rnul} A = \operatorname{nul} A < 0$, and Subroutines* **AC** *and LIN·SOLVE.*

OUTPUT *and Stages 1 and 2 of* COMPUTATIONS *as in Algorithm 4.2.*

COMPUTATIONS:

3. *Test whether $AC^{-}U = 0$. If so, output the integer r. Compute and output the Q-factor of the matrix $C^{-}U$.*
4. *Otherwise $r' \leftarrow r - 1$ and reapply the algorithm.*

Remark 4.1. *At Stage 3 we can first test whether $AC^{-}U\mathbf{x} = \mathbf{0}$ for a random vector $\mathbf{x}$. If so, continue Stage 3. Otherwise perform Stage 4. The same simplified test can precede testing whether $AC^{-}U = 0$ in Algorithms 5.1 and 5.3 as well as in Flowchart 6.1 in the next section.*

We can combine Algorithms 4.2 and 4.3 to support the binary search for $\operatorname{nul} A$. For example, we can recursively apply Stages 1 and 2 of Algorithm 4.2 for $r' = 2^i - 1$, $i = 0, 1, \dots$, as long as the Subroutine LIN·SOLVE outputs FAILURE. Whenever for some $r' = 2^i - 1$ it does not fail and outputs the matrix $C^{-}U$, we test whether $AC^{-}U = 0$. If so, we output the integer r' and the matrix $Q(C^{-}U)$ (cf. Section 2.1). Otherwise we perform the binary search for $\operatorname{nul} A$ in the open interval $(2^{i-1} - 1, 2^i - 1)$, where we decrease r' if the computed matrix C is rank deficient and increase r' if $AC^{-}U \neq 0$. In this process we deal with $m \times n$ matrices A and C in every recursive step, but in the next section we recursively decrease the size of an input matrix.

5 Right null bases via A-preprocessing and Aggregation

Let us back up Algorithm 4.3 with three alternatives.

Corollary 3.1 implies correctness of the following algorithm.

Algorithm 5.1. A null basis via aggregation *(see Remark 5.2).*

INPUT, OUTPUT, *and Stages 1–3 of* COMPUTATIONS *as in Algorithm 4.3.*

COMPUTATIONS:

4. *Otherwise compute a unitary matrix basis X for the null space $N(AC^{-}U)$ and output the number of its columns.*
5. *Compute and output the Q-factor of the matrix $C^{-}UX$.*

In virtue of Theorem 3.1, 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 $U_1V_1^H$ such that the A-modifications $C = A + UV^H$ and $C_1 = A + U_1V_1^H$ have full rank (cf. (3.2)). Furthermore, the ranges are likely to share nothing else where the ACs UV^H and $U_1V_1^H$ are random and unitary. These implications can be sketched as follows.

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

$$\text{and for matrices } C,\ C_1,\ \text{and } X \text{ of full rank}\} \Longrightarrow$$

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

In our next algorithm we assume that $m = n$ and choose generators U, V, U_1, and V_1 such that $N(A) = \text{range}(C^{-}U) \bigcap \text{range}(C_1^{-}U_1)$.

Algorithm 5.2. A null basis via space intersection *(cf. Remarks 4.1, 5.1, and 5.2).*

INPUT, OUTPUT, *and Stages 1–3 of* COMPUTATIONS *as in Algorithm 4.3, except for the restriction that* $m = n$ *added in the present* INPUT. *(In this case* $C^- = C^{-1}$ *and* $C_1^- = C_1^{-1}$ *for the full rank matrices* C *and* C_1 *below.)*

COMPUTATIONS:

4. *Otherwise choose an integer* $r_1' \geq \operatorname{nul} A$ *such that* $r_1' + r \leq m$. *If there is no such an integer, output FAILURE. Otherwise apply Stages 1–3 of Algorithm 4.3 to the pair of* A *and* r_1' *to output FAILURE or a positive integer* r_1 *such that* $\operatorname{nul} A \leq r_1 \leq r_1'$, *a pair of* $n \times r_1$ *unitary matrices* U *and* U_1, *and an* $AC\,U_1V_1^H$ *of rank* r_1. *Choose the matrix* U_1 *such that*

$$(\operatorname{range} U_1) \bigcap \operatorname{range}(AC^-U) = \{\mathbf{0}\}, \tag{5.1}$$

(This equation is ensured if, say, $U_1^H AC^-U = 0$.*) Write* $C_1 = A + U_1V_1^H$.

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

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

To prove correctness of the algorithm, write $L = \operatorname{range}(C^-UX)$ and observe that

$$L = \operatorname{range}(C_1^-U_1Z) = \operatorname{range}(C^-U) \bigcap \operatorname{range}(C_1^-U_1).$$

Therefore, $N(A) \subseteq L$ (cf. (3.1) and (3.2)). 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 $C_1\mathbf{y} = A\mathbf{y} + U_1V_1^H\mathbf{y}$. Simultaneously we have $A\mathbf{y} \in AL \subseteq \operatorname{range}(AC^-U)$. Consequently, $A\mathbf{y} = \mathbf{0}$ due to assumption (5.1). 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 all our Null Aggregation algorithms (in terms of arithmetic operations 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 of Algorithm 5.2). If $m = n$ we can compute the r columns of the matrix $C^-U = C^{-1}U$ by using about $(2/3)n^3 + 2rn^2$ arithmetic operations.

Without restrictive equations (5.1) and $m = n$, we still have

$$L = \text{range}(C^{-}U) \bigcap \text{range}(C_1^{-}U_1) \supseteq N(A)$$

but not necessarily $L = N(A)$. Clearly, however, we are likely to have the equality $L = N(A)$ for a pair of random well conditioned full rank matrices U of size $m \times r$ and U_1 of size $m \times r_1$ such that $r + r_1 \leq m$. Indeed, for such a choice, the dimension of the intersection is unlikely to exceed its minimum over all choices of U and U_1. If the unlikely inequality $L \neq N(A)$ holds, we can generate a new APP $U_1 V_1^H$ and repeat our computations. This leads us to the following algorithm.

Algorithm 5.3. A null basis via recursive space intersection *(cf. Remarks 4.1, 5.1, and 5.2).*

INPUT, OUTPUT, *and Stages 1–5 of* COMPUTATIONS *as in Algorithm 5.2 except that the matrix U_1 is random, no restriction that $m = n$ is imposed anymore, and equation (5.1) is not enforced at Stage 4 of the computations anymore.*

COMPUTATIONS:

6. *Compute the $m \times s$ matrix $C^{-}UX$ of a rank $s_{-} \leq s$. 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.*
7. *Otherwise $C^{-}U \leftarrow C^{-}UX$ and reapply Stages 4–7.*

Remark 5.1. *For $m = n$ the matrix C_1 is nonsingular, and so the matrices $(C^{-}U, C_1^{-}U_1)$ and $(C_1 C^{-}U, U_1)$ share their null space. Therefore, seeking matrix bases at Stage 4 of Algorithms 5.2 and 5.3, we can avoid the computation of the matrices C_1^{-} and $C_1^{-}U_1$ and thus save some arithmetic operations.*

Remark 5.2. *In Stage 3 of Algorithm 5.1 we compute a null basis X for the $m \times r$ aggregate $AC^{-}U$ of an $m \times n$ input matrix A. In Stage 4 we disaggregate this basis X to obtain a null basis for the matrix A. The smaller r, the simpler the computation of a null basis. Likewise, Algorithm 5.2 aggregates the original null basis problem into the same problem for the $n \times (r + r_1)$ matrix $(C^{-}U, C_1^{-}U_1)$, and we can simialrly interpret Algorithm 5.3. The process can be continued recursively. We call these techniques the* Null Aggregation. *They are natural descendants of the* aggregation methods *in [15], which in the 1980s evolved into the* Algebraic Multigrid. *Seeking a better insight into A-preconditioning and aggregation, one can compare these two aggregation approaches with the techniques of* the Schur Aggregation *in [10] (cf. our Theorem 7.1) and* trilinear aggregating *in [16]. The latter technique has been an indispensible ingredient in the design of the currently fastest algorithms for $n \times n$ matrix multiplication, both the fastest theoretically for immense dimensions n [17] and the fastest for the dimensions n from 20 to, say, 2^{20} [16], [18], in which case efficient numerical implementations can be found in [19], [20].*

6 A flowchart for the null basis computation

Let us summarize our null basis algorithms for a matrix A and a nonnegative integer r.

Flowchart 6.1. The Null Basis Algorithms.

1. *Apply Subroutine* **AC** *to generate an APC* UV^H *of rank* r. *Output FAILURE if so does the Subroutine.*
2. *Compute the* A*-modification* $C = A + UV^H$.
3. *If it is rank deficient (cf. Algorithm 4.2), increase the integer* r *by one and reapply Stage 1. Otherwise compute the matrix* $\tilde{U} = Q(C^-U)$.
4. *In Algorithms 4.1 and 4.2 output the matrix* $\tilde{U}$ *and stop as soon as this stage is reached.*
5. *In the other algorithms do this if* $AC^-U = 0$ *(cf. Remark 4.1). Otherwise in Algorithm 4.3 decrease the integer* r *by one and go to Stage 1, whereas in Algorithms 5.1–5.3 compute and output the matrix* $Q(C^-UX)$. *Here* X *is a null matrix basis for the aggregate* AC^-U *in Algorithm 5.1 and the full-rank block made up of the first* r *rows of a null matrix basis for the aggregate* $(C^-U, C_1^-U_1)$ *in Algorithms 5.2 and 5.3.*

7 The left null bases and the reduction to square inputs and to the Schur Aggregation

We can apply our previous study to the matrix A^H to compute the integer $\operatorname{rnul} A^H = \operatorname{lnul} A$ and a right null matrix basis for the matrix A, which is a left null matrix basis for the matrix A. If $m = n$, we can apply the same factorization of the matrix C to computing both left and right null bases of the matrix A. Equation (3.5) remains valid. Here are the dual counterparts of equations (3.1)–(3.2) and (3.3)–(3.4), respectively:

$$\{r \geq \operatorname{rank} V \geq \operatorname{lnul} A\} \Longrightarrow \{LN(A) \subseteq \operatorname{range}(V^H C^-)\}, \tag{7.1}$$

$$\{r = \operatorname{rank} V = \operatorname{lnul} A\} \Longrightarrow \{LN(A) = \operatorname{range}(V^H C^-)\}. \tag{7.2}$$

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 any $n \times m$ 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. In these cases we lose symmetry but have $\mathrm{cond}_2 \tilde{A} = \mathrm{cond}_2 A$ and add no errors in the numerical computation of the matrix $\tilde{A}$.

If $m > n$, we can apply Algorithms 4.1–4.3 and 5.1–5.3, but alternatively we can append the $m - n$ column vectors filled with zeros 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.

Furthermore, for any pair of m and n 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 $\mathrm{rank}\, \tilde{A} = 2\, \mathrm{rank}\, A$, so that $\mathrm{nul}\, \tilde{A} = 2n - 2\, \mathrm{rank}\, A = 2\, \mathrm{nul}\, A$ for $m = n$.

In Remark 5.1 we pointed out a simplification of Algorithm 5.1 in the case of square matrices A. In fact for $m = n$ we can simplify Algorithms 4.3 and 5.1–5.3 further, based on the following simple fact.

Theorem 7.1. *For matrices A, U, and V of sizes $m \times n$, $m \times r$, and $n \times r$, respectively, such that $\min\{m, n\} > r$, let the matrix C have the full rank m and write $G = I_r - V^H C^- U$. Then $V^H C^- A = G V^H$ if $m \geq n$ and $A C^- U = U G$ if $m \leq n$.*

Proof.

$$V^H C^- A = V^H C^- (C - U V^H) = V^H - V^H C^- V^H = G V^H \text{ if } m \geq n,$$

$$A C^- U = (C - U V^H) C^- U = U - U V^H C^- U = U G \text{ if } m \leq n.$$

□

The theorem reduces the null space computation for square matrices A to the Schur Aggregation in [10]. If the matrices U and U_1 have full rank, then in Algorithms 4.3 and 5.1–5.3 we can replace the matrices $AC^-U \leftarrow G$ and $AC_1^- U_1 \leftarrow G_1$ for $G = I_r - V^H C^- U$ and $G_1 = I_{r_1} - V_1^H C_1^- U_1$.

8 Computing the null vectors via A-preprocessing

If we only need to compute a normalized null vector $\mathbf{y}$ of a matrix A, we can simplify our Null Aggregation algorithms as follows.

a) Change Stage 3 (resp. 4) in Algorithm 4.1 (resp. 4.2) as follows. 3 (resp. 4). Compute and output the vector $\mathbf{y} = C^- U \mathbf{x} / ||C^- U \mathbf{x}||_2$ for a normalized vector $\mathbf{x}$.

b) Change Stage 3 in Algorithms 4.3 and 5.1–5.3 as follows.

3. If there exists a normalized solution $\mathbf{x}$ to the vector equations $AC^{-}U\mathbf{x} = \mathbf{0}$ (or equivalently $G\mathbf{x} = \mathbf{0}$ for $G = I_r - V^H C^{-} U$ where the matrix U has full rank), then compute such a solution and compute and output the vector $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$.

c) Change Stages 5 and 6 in Algorithm 5.2 as follows.

5. Compute the matrix $C^{-1}U$. 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 $\mathbf{y} = C^{-}U\mathbf{x}/||C^{-}U\mathbf{x}||_2$.

d) Initialize Algorithm 5.3 with setting $i \leftarrow 1$ and change its Stages 4–7 as follows.

4. Otherwise choose an integer $r_i' \geq \operatorname{nul} A$ such that $r_i' + r \leq m$ (output FAILURE if this is impossible) and apply Stages 1–3 of Algorithm 4.3 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 8.1. *Let us extend our null vector algorithms to the computation of a null basis for an $m \times n$ matrix A without reverting back to the original algorithms. Compute the value $\sigma \approx \sigma_1(A)$ and two normalized (right) null vectors $\mathbf{y}$ and $\mathbf{z}$ for the matrices A and A^H, respectively, so that $\mathbf{z}^H$ is a normalized left null vector $\mathbf{z}$ for the matrix A. Set $A \leftarrow A + \sigma\mathbf{y}\mathbf{z}^H$ and reapply the null vector 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 Solving Linear Systems of Equations via the Null Aggregation

The matrix equation $AY = B$ can be reduced to computing an $(n + k) \times k$ matrix $Z = \begin{pmatrix} I_k \\ Y \end{pmatrix} \in N(\widehat{A})$ where $\widehat{A} = (-B, A)$ is an $m \times (n + k)$ matrix. For

this task we can extend our Null Aggregation algorithms in Sections 4 and 5. If $n+k > m$ we should first apply our recipes in Section 7 to obtain an equivalent system of μ equations with μ unknowns for $\mu \geq n+k$. (Conversely, the null space computation is a special case of solving matrix equation $AY = B$ where $B = 0$.) In our next algorithm we elaborate upon the reduction from equation $AY = B$ in the case where $m \geq n+k$.

We assume that the input includes the Subroutine LIN·SOLVE defined in Section 4 and the Subroutine NULL·BASIS that has an $m \times n$ matrix A and an integer $r \geq \operatorname{nul} A$ as the input and outputs a unitary null matrix basis Y for the matrix A. Such a subroutine can be defined by our algorithms in Sections 4 and 5 and in Remark 8.1.

Algorithm 9.1. Solutions to linear systems as null vectors.

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

OUTPUT: *an $n \times k$ 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 choose an integer r that supports generating a pair of unitary matrices $\widehat{U}$ of size $m \times r$ and $\widehat{V}$ of size $(n+k) \times r$ such that, for the $m \times (n+k)$ matrix $\widehat{A} = (-B, A)$, the matrix $\widehat{C} = \widehat{A} + \widehat{U}\widehat{V}^H$ has full rank. Apply the Subroutine NULL·BASIS to the pair $(r, \widehat{A})$ to compute an $(n+\nu) \times \nu$ unitary null matrix basis $\widehat{Y}$ for the matrix $\widehat{A}$.

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

To prove correctness of the algorithm, first observe 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 $Z = \begin{pmatrix} I_k \\ Y \end{pmatrix} \in N(\widehat{A})$. Therefore, the equation $AY = B$ is consistent if and only if the matrix Y_0 has full rank k, and in this case we have

$$\operatorname{range} Y_0 = \operatorname{range} I_k. \tag{9.1}$$

It remains to show that $AY = B$ for $Y = Y_1 Y_0^-$. From the equation $(-B, A)\widehat{Y} = 0$ deduce that $(-B, A)\widehat{Y} Y_0^- = 0$, whereas $\widehat{Y} Y_0^- = \begin{pmatrix} I_k \\ Y_1 Y_0^- \end{pmatrix}$. Therefore, $AY = B$ for $Y = Y_1 Y_0^-$, and this completes the correctness proof.

For smaller integers k the complexity of Algorithm 9.1 is dominated by the cost of application of the Subroutine NULL·BASIS at Stage 1.

Remark 9.1. *In its numerical implementation Algorithm 9.1 can fail at Stage 2 if the matrix Y_0 is nonsingular but ill conditioned. 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 $k \times k$ unitary matrix. Then at Stage 2 we would readily compute Y_0^- based on equation (2.2). The entire problem, however, practically disappears in the case where $k = 1$ and Y_0 is a scalar. Indeed, for $k = 1$ the matrix equation $AY = B$ turns into a linear system $A\mathbf{y} = \mathbf{b}$ of m equations with n unknowns, the matrices Y_0 and Y_1 turn into a scalar and a vector, respectively, and it remains to compute the vector $\widehat{Y} = \widehat{\mathbf{y}} = (\widehat{y}_i)_{i=0}^n$ such that $\widehat{y}_0 \neq 0$. For larger values k we can concurrently solve the k linear systems $A\mathbf{y}_j = \mathbf{b}_j$ for $j = 1, \dots, k$ where $Y = (\mathbf{y}_j)_{j=1}^k$ and $B = (\mathbf{b}_j)_{j=1}^k$. The advantages of concurrent processing and the simplicity of these computations seem to give them the upper hand over the application of Algorithm 9.1 for $k > 1$.*

PART II. NUMERICAL COMPUTATIONS IN THE NULL SPACES AND EXTENSIONS

10 Numerical computations in the null spaces and the Tail and Head Approximation

Given a normalized $n \times n$ matrix A of rank $n-1$ (with $||A||_2 = 1$), suppose we seek its normalized null vector $\mathbf{y}$ such that $A\mathbf{y} = \mathbf{0}$. Let a normalized rank-one APP $\mathbf{u}\mathbf{v}^H$ define a nonsingular A-modification $C = A + \mathbf{u}\mathbf{v}^H$. Then $\mathbf{y} = \tilde{\mathbf{y}}/||\tilde{\mathbf{y}}||_2$, $\tilde{\mathbf{y}} = C^{-1}\mathbf{u}$, so that the problem is essentially reduced to solving a nonsingular linear system of equations $C\tilde{\mathbf{y}} = \mathbf{u}$.

According to our study in [8], for a pair of random normalized vectors $\mathbf{u}$ and $\mathbf{v}$ we can expect that the ratios $\sigma_n(C)/\sigma_{n-1}(A)$ and therefore $\mathrm{cond}_2 C/\mathrm{cond}_2 A$ are neither large nor small, so that the A-modification C is well conditioned if and only if so is the matrix A.

Now suppose that the ratio $\sigma_1(A)/\sigma_{n-2}(A)$ is not large but $\sigma_{n-2}(A) \gg \sigma_{n-1}(A)$, so that our null space computations are ill conditioned. Then for random normalized APPs $\mathbf{u}\mathbf{v}^H$ and $\mathbf{u}_1\mathbf{v}_1^H$ and for the A-modifications $C_1 = A + \mathbf{u}_1\mathbf{v}_1^H$ and

$$C = C_1 + \mathbf{u}\mathbf{v}^H = A + UV^H, \quad U = (\mathbf{u},\ \mathbf{u}_1), \quad V = (\mathbf{v}, \mathbf{v}),$$

we can expect that $\sigma_{n-1}(C_1) \gg \sigma_n(C_1)$, whereas the ratios $\sigma_1(C_1)/\sigma_{n-1}(C_1)$ and $\sigma_1(C)/\sigma_n(C)$ are not large.

It remains to solve at first the well conditioned matrix equation $CW = U$ and then the homogeneous linear system $AW\mathbf{x} = 0$ of n equations with two unknowns. If the matrix U has full rank two, we can solve just the homogeneous linear system $G\mathbf{x} = \mathbf{0}$ of two equations with two unknowns where $G = I_2 - V^H C^{-1} U$ (cf. Theorem 7.1). Numerically we should apply the orthogonalization and least-squares methods [1, Chapter 5], [2, Chapter 4], [21], [22], but first we should ensure high precision computation of the ill conditioned matrices $AC^{-1}U$ or G. We refer the reader to [10] and [12] on these computations, which rely on the extended iterative refinement and MSAs.

Now observe that any nonsingular small-norm perturbation $\tilde{A} = A + E$ of the matrix A is a strictly $(0, 2)$ matrix, whereas every nonsingular strictly $(0, 2)$ matrix $\tilde{A}$ can be obtained by a small norm perturbation of a well conditioned matrix A of rank $n - 2$. Given a strictly $(0, 2)$ matrix $\tilde{A}$, we can apply our null space algorithms to output an approximation $\tilde{C}^{-1}\tilde{U}$ to an $n \times 2$ matrix basis for the 2-tail of the SVD of the matrix A (that is the singular subspace associated with the two smallest singular values of the matrix). This motivates using the nomenclature of the *Tail Approximation*.

We can preserve the structure of an input matrix in this approximate matrix basis even where the respective singular space has no structured matrix bases.

The Tail Approximation can be readily extended to an $m \times n$ matrix A of a rank ρ for which $\sigma_i(A) \gg \sigma_{i+1}(A)$ for some positive $i < \rho$, whereas the ratios $\sigma_1(A)/\sigma_i(A)$ and $\sigma_{i+1}(A)/\sigma_\rho(A)$ are not large. Some additional care is required for an $m \times n$ well conditioned matrix A of a rank ρ where $\sigma_{i+1}(A) \gg \sigma_i(A)$ for more than one subscript $i < \rho$ or where, say, $\rho > 50$ and $2 \le \sigma_{i+1}(A)/\sigma_i(A) \le 3$ for $i = 1, 2, \dots, \rho - 1$, so that $\mathrm{cond}_2 A \ge 2^{50}$. In such cases we need APPs of larger ranks to yield well conditioned A-modifications C. Then we can apply the Tail Approximation with an APP of a smaller size to the matrix $C^{-1}U$ to approximate at first a null basis X for the matrix $AC^{-1}U$ and/or G and then the null basis $C^{-1}UX$ for the matrix A.

We refer the reader to [5] on the extension of the Tail Approximation to eigen-solving, with further applications to polynomial root-finding.

By applying the Tail Approximation to the matrix A^- for an $n \times m$ strictly $(q, 0)$ matrix A and an APP VU^H, we define the *Head Approximation*. In this case the matrix $C_- = A^- + VU^H$ plays the role of the A-modification C used in the Tail Approximation, but for smaller integers q it is more efficient to operate with the matrix $(C_-)^-$ and to adjust the algorithms respectively. For full rank matrices A and C_- we have the convenient expression

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

which defines the *dual aggregate* H and the *dual aggregation*.

11 Improving APCs

Suppose for an ill conditioned matrix A of full rank we obtain a crude APC and the integer $\mathrm{nnul}\, A$, e.g., by extending Algorithms 4.2, 4.3, 5.1–5.3 to the Tail

Aggregation. In this section we refine such an APC.

At first let A denote a rank deficient matrix with nullity r and let UV^H denote an AC of the rank r such that the A-modification $C = A + UV^H$ has full rank. We may have $\mathrm{cond}_2 C > \mathrm{cond}_2 A$ and even $\mathrm{cond}_2 C \gg \mathrm{cond}_2 A$, but the following transform serves as a remedy,

$$(U \leftarrow Q(C^{-}U), \quad V \leftarrow Q(C^{-H}V)). \tag{11.1}$$

Clearly, the new APP UV^H is still an AC, and our next theorem shows that A-preprocessing with this AC preserves the condition number of the matrix A.

Theorem 11.1. *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 $r = n - \rho = \mathrm{nul}\, A$ and the matrix $C = A + UV^H$ is nonsingular. Let $U_1 = Q(C^{-}U)$ and $V_1 = Q(C^{-H}V)$ denote the respective updates of the matrices U and V according to policy (11.1). Then the matrix $A + U_1 V_1^H$ is nonsingular and $\mathrm{cond}_2(A + U_1 V_1^H) = \mathrm{cond}_2 A$.*

Proof. Due to Theorem 3.1, 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 11.1 follows because $r = n - \rho$ and $\sigma_1 = 1$. □

Suppose the matrix A in this theorem is well conditioned. Then so is the matrix $A + U_1 V_1^H$ as well as all nearby matrices. Therefore, $U_1 V_1^H$ is an APC for any $(0, r)$ (ill conditioned) matrix lying near the matrix A. If UV^H is a crude APC of rank r for such a $(0, r)$ matrix A, then the transformation (11.1) dramatically increases the power of this APC according to our extensive tests (cf. [8, Table 7.2]). In the following algorithm this property of transform (11.1) is extended to the compression of the APCs of larger ranks, which can be generated readily. We denote this $(0, r)$ matrix by A rather than $\tilde{A} = A + E$.

Flowchart 11.1. Inflation/Compression of an APC *(cf. [23]).*

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 unitary or well conditioned matrix bases $T(U)$ and $T(V)$ for the r-trailing right singular spaces of the matrices $AC^{-}U$ and $A^H C^{-H} V$, respectively. (If $m = n$ and the matrices U and V are unitary, we can apply Theorem 7.1 and compute just the r-tail of the matrix $G = I_h - V^H C^{-} U$.)*

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

X. Wang in [23] has applied an algorithm similar to Flowchart 11.1 to 10×10 Hilbert input matrices $A = (\frac{1}{i+j-1})_{i,j=1}^{10}$ and has consistently arrived at

$\mathrm{cond}_2 C \approx \frac{\sigma_1(A)}{\sigma_{10-h}(A)}$ in his extensive tests for various choices of positive $h \le 10$ and $r < h$.

Random or pseudo random APPs UV^H whose rank exceeds $\mathrm{nnul}\, A$ is a safe initial choice for obtaining a well conditioned matrix $C = A + UV^H$ according to our tests. Flowchart 11.1 complements this choice to yield effective APCs of the rank $\mathrm{nnul}\, A$.

Finally, the flowchart extends Algorithm 5.1, but we can similarly extend Algorithms 5.2 and 5.3.

Here is a natural extension of our policy (11.1) to dual APCs VU^H,

$$V \leftarrow Q((C_-)^- V), \quad U \leftarrow Q((C_-)^{-H} U)). \tag{11.2}$$

APPENDIX

Some estimates for the tails of the SVDs

Theorem A-1. *Assume an $m \times n$ matrix A for $m \ge n$ and an APP UV^H such that the A-modification $C = A + UV^H$ has full rank n. Then the vector $\mathbf{y} - C^- A\mathbf{y}$ lies in the space* $\mathrm{range}(C^- U)$.

Proof. Post-multiply the matrix equation $C = A + UV^H$ by $\mathbf{y}$, pre-multiply it by C^-, substitute $C^- C = I$, and obtain that $\mathbf{y} = C^- A\mathbf{y} + C^- U\mathbf{z}$ for $\mathbf{z} = V\mathbf{y}$. □

The theorem implies that for a matrix A and a full rank matrix $C = A + UV^H$, a vector $\mathbf{y}$ lies near the space $\mathrm{range}(C^- U)$ provided the vector $A\mathbf{y}$ has a small norm (and the norm $||C^-||_2$ is not very large). Conversely, our next theorem bounds the norm $||A\mathbf{y}||_2$ for the vectors $\mathbf{y}$ from the space $\mathrm{range}(C^- U)$. In the proof we use the two following lemmas.

Lemma A-1. *Let C and $C + E$ be two matrices of full rank. Then*

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

Lemma A-2. *For two square matrices C and E of the same size such that 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 [2, Theorem 1.4.18] for $P = C^{-1}E$. □

Theorem A-2. *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 $n \times r$, write $C = A + UV^H$ and assume that $r = \mathrm{nul}(A - E)$, the matrix C has full rank,*

$$||A||_2 = 1,\ \delta = ||E||_F < \sigma_- = \sigma_n(C) = 1/||C^-||_2 \le ||C||_2 \le 2,$$

and $\mathbf{y} = C^- U\mathbf{x}$ for a normalized vector $\mathbf{x}$. Then $||A\mathbf{y}||_2 \le \tau ||A||_2 ||\mathbf{y}||_2$ where $\tau \le \delta + (4 + 4\delta)\delta/(\sigma_- - \delta)^2$ (for $m \ge n$), and if $m = n$, then $\tau \le \delta + (1 + \delta)\delta/(\sigma_- - \delta)$.

Proof. We have $(A-E)(C-E)^{-}U\mathbf{x}=\mathbf{0}$ in virtue of Theorem 3.1 (cf. (3.4)). Therefore, $A\mathbf{y}=E\mathbf{y}+\mathbf{z}$ where

$$\mathbf{z}=(A-E)\mathbf{y}=(A-E)C^{-}U\mathbf{x}=(A-E)(C^{-}-(C-E)^{-})U\mathbf{x}.$$

It follows that $||\mathbf{z}||_2\le||A-E||_2||C^{-}-(C-E)^{-}||_2\le(1+\delta)||C^{-}-(C-E)^{-}||_2$ because $||E||_2\le||E||_F=\delta$, $||\tilde{A}||_2=1$, and consequently $||A-E||_2\le||A||_2+||E||_2\le1+\delta$.

Moreover, $2||\mathbf{y}||_2\ge||\mathbf{y}||_2||C||_2\ge||C\mathbf{y}||_2$, and since $C\mathbf{y}=U\mathbf{x}$ for $m\ge n$, we obtain that $2||\mathbf{y}||_2\ge||U\mathbf{x}||_2=1$. Furthermore, $||(C-E)^{-}||_2\le1/(\sigma_{-}-\delta)$, $||C^{-}||_2=1/\sigma_{-}$. Combine all these estimates with Lemma A-1 and obtain the claimed bound on τ for $m\ge n$.

For $m=n$ we have $C^{-}-(C-E)^{-}=(I-(C-E)^{-1}C)C^{-1}$, and therefore $\mathbf{z}=(A-E)(I-(C-E)^{-1}C)C^{-1}U\mathbf{x}$. Substitute $\mathbf{y}=C^{-}U\mathbf{x}$ and obtain $\mathbf{z}=(A-E)(I-(C-E)^{-1}C)\mathbf{y}$. Consequently $||\mathbf{z}||_2\le||A-E||_2||I-(C-E)^{-1}C||_2||\mathbf{y}||_2$. To estimate the norm $||I-(C-E)^{-1}C||_2$, apply Lemma A-2 and substitute the bound $||C^{-1}E||_2\le||C^{-1}||_2||E||_2\le\delta/\sigma_{-}$. Combine the resulting estimate for the norm $||\mathbf{z}||_2$ with the bound $||A||_2\le1+\delta$ and the equation $A\mathbf{y}=E\mathbf{y}+\mathbf{z}$ and obtain the theorem for $m=n$. □

References

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

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

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

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

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

[6] 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.

[7] 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.

[8] 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.

[9] V. Y. Pan, D. Ivolgin, B. Murphy, R. E. Rosholt, Y. Tang, X. Yan, Additive Preconditioning and Aggregation in Matrix Computations, Technical Report TR 2007 002, *CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York*, March 2007.

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

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

[12] 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.

[13] 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.

[14] 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.

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

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

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

[18] 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.

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

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

[21] 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.

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

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