Linear Algebra and its Applications 434 (2011) 854–879

Contents lists available at ScienceDirect

Linear Algebra and its Applications

journal homepage: www.elsevier.com/locate/laa

Matrix computations and polynomial root-finding with preprocessing☆

Victor Y. Pan [a,b,*], Guoliang Qian [b], Ai-Long Zheng [b], Zhao Chen [c]

[a] *Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA*

[b] *Ph.D. Programs in Mathematics and Computer Science, The Graduate Center of the City University of New York, New York, NY 10036, USA*

[c] *Department of Mathematics, Polytechnique Institute of the City University of New York, USA*

ARTICLE INFO

Article history:
Received 4 March 2010
Accepted 4 April 2010
Available online 3 December 2010

Submitted by A. Böttcher

AMS classification:
65F05
65F22
65F35

Keywords:
Linear systems of equations
Randomized preprocessing
Eigen-solving
Polynomial equation
Secular equation

ABSTRACT

Solution of homogeneous linear systems of equations is a basic operation of matrix computations. The customary algorithms rely on pivoting, orthogonalization and SVD, but we employ randomized preprocessing instead. This enables us to accelerate the solution dramatically, both in terms of the estimated arithmetic cost and the observed CPU time. The approach is effective in the cases of both general and structured input matrices and we extend it and its computational advantages to the solution of nonhomogeneous linear systems of equations, matrix eigen-solving, the solution of polynomial and secular equations, and approximation of a matrix by a nearby matrix that has a smaller rank or a fixed structure (e.g., of the Toeplitz or Hankel type). Our analysis and extensive experiments show the power of the presented algorithms.

☆ Supported by PSC CUNY Awards 69330-0038 and 61406-0039.

* Corresponding author at: Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA.

E-mail addresses: victor.pan@lehman.cuny.edu (V.Y. Pan), gqian@gc.cuny.edu (G. Qian), ailongz@yahoo.com (A.-L. Zheng), c718212@yahoo.com (Z. Chen).

URL: http://comet.lehman.cuny.edu/vpan/ (V.Y. Pan).

0024-3795/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.laa.2010.04.011

1. Introduction

Solution of a homogeneous linear system of equations $M\mathbf{y} = \mathbf{0}$ is a basic operation of matrix computations. We call the solution vectors $\mathbf{y}$ the *null vectors* of the input matrix M and call the space $\mathbb{N}(M)$ of these vectors its *null space*. If the columns of a matrix B of full column rank span the null space $\mathbb{N}(M)$, then we call the matrix B a *null matrix basis* or *nmb* for the matrix M and write $B = \mathrm{nmb}(M)$.

The customary methods for computing null vectors and nmbs rely on orthogonalization and pivoting (see Section 3), which makes them costly, particularly for structured (e.g., Toeplitz or Hankel) matrices, but we employ randomized preprocessing instead, which enables dramatic acceleration of the computations. For example, in the case of $n \times n$ Toeplitz and Hankel input matrices the estimated running time decreases from quadratic to nearly linear, and in our extensive tests we observed the decrease of the respective CPU time by the factor $a(n)$ where $a(512) > 18, a(1024) > 90$, and $a(2048) > 300$ (see Section 12.1).

The study of randomized preprocessing was scattered throughout the papers [25,28,29,30]. In Sections 4–7 we summarize it, supply some perturbation analysis, and link to each other the three main variations of this approach, that is randomized additive and multiplicative preprocessing and randomized augmentation. In Sections 12.1–12.3 we present the results of supporting numerical experiments.

Then we cover the extensions of the resulting algorithms for the null space computations to

(a) approximation of a matrix by nearby matrices having smaller ranks or smaller displacement ranks in Sections 8 and 12.5,
(b) the solution of nonhomogeneous linear systems of equations in Sections 9 and 12.4,
(c) eigen-solving in Sections 10 and 12.6, and
(d) root-finding for polynomial and secular equations in Sections 11 and 12.6.

Our tests in Section 12 (the contribution of the last three authors) demonstrate that the approach is powerful and practically promising.

Let us briefly comment on the two latter links. The extension to eigen-solving relies on the observation that the eigenspace associated with the eigenvalue λ of a matrix M is just the null space of the shifted matrix $M - \lambda I$. The Rayleigh quotient iteration [12,36] amounts essentially to the solution of ill conditioned linear systems with the matrices $M - \lambda^{(i)}I$ for $\lambda^{(i)} \approx \lambda$ and $i = 0, 1, \ldots$ With our preprocessing we solve well conditioned linear systems instead, which enables us to employ Conjugate Gradient algorithms and iterative refinement and to use factorization of a single matrix $M - \lambda^{(h)}I$ for a number of successive iteration steps, $i = h, h + 1, \ldots$ Furthermore our preprocessing can simplify eigen-solving for structured matrices associated with polynomial and secular equations. Our tests show no substantial slowdown of the convergence, which could overweight the effect of our simplification of every iteration loop.

With the listed directions in mind we mostly restrict our presentation to the case of square input matrices, although the techniques for the null space computations, matrix inversion, and solving linear systems of equations can be extended to the case of rectangular inputs by means of the techniques in [23,25,28,29,30], and the first author is working on the extension of the presented approach to some other problems of matrix and polynomial computations.

2. Definitions

Hereafter ω_k denotes the kth root of unity $\omega_k = \exp\left(\frac{2\pi}{k}\sqrt{-1}\right)$ and the abbreviation "*nlns*" stands for "neither large nor small".

2.1. General and structured matrices

M^T and M^H denote the transpose and the Hermitian (complex conjugate) transpose of a matrix M, respectively.

$(M_1, \dots, M_k) = ((M_i^T)_{i=1}^k)^T$ is a $1 \times k$ block matrix with the blocks $M_1, \dots, M_k$.

$\mathrm{diag}(M_1, \dots, M_k) = \mathrm{diag}(M_i)_{i=1}^k$ is a $k \times k$ block diagonal matrix with the diagonal blocks $M_1, \dots, M_k$.

I_k or just I denote the $k \times k$ identity matrix. $\mathbf{e}_j$ is its jth column, $j = 1, \dots, k$, so that $I = (\mathbf{e}_1, \dots, \mathbf{e}_k)$. $O_{k,l}$ or just O denote the $k \times l$ null matrix, filled with zeros.

A matrix M is called unitary and orthonormal if $M^H M = I$.

$\nu = n - \rho$ is the nullity of an $m \times n$ matrix of a rank ρ.

$M = V_{\mathrm{full}} \Sigma_{\mathrm{full}} W_{\mathrm{full}}^H$ is a full SVD or just SVD (that is Singular Value Decomposition) of an $n \times n$ matrix M of a rank ρ provided V_{full} and W_{full} are two square unitary matrices (that is $V_{\mathrm{full}} V_{\mathrm{full}}^H = V_{\mathrm{full}}^H V_{\mathrm{full}} = I_m$, $W_{\mathrm{full}}^H W_{\mathrm{full}} = W_{\mathrm{full}} W_{\mathrm{full}}^H = I_n$), $\Sigma_{\mathrm{full}} = \mathrm{diag}(\Sigma, O)$, $\Sigma = \mathrm{diag}(\sigma_i)_{i=1}^{\rho}$, and $\sigma_1 \geqslant \sigma_2 \geqslant \dots \geqslant \sigma_\rho > 0$.

$\Sigma_{\mathrm{full}}^+ = \mathrm{diag}(\Sigma^{-1}, O)$, $M^+ = W_{\mathrm{full}} \Sigma_{\mathrm{full}}^+ V_{\mathrm{full}}^H$ is the generalized Moore–Penrose inverse of a matrix M of full rank, $M^+ = M^{-1}$ for a nonsingular matrix M.

$\|M\| = \sigma_1$ is the 2-norm of a matrix M, and $\mathrm{cond}(M) = \frac{\sigma_1}{\sigma_\rho}$ is its condition number, so that $\mathrm{cond}(M) = \|M\| \|M^+\|$. A matrix M is *ill conditioned* if $\mathrm{cond}(M)$ is large. Otherwise the matrix is *well conditioned*. A matrix has numerical nullity r if it has exactly r singular values that are small relatively to its norm. Here the words "small" and "large" are meant in the context of the assumed computational task and computer environment.

For a positive integer $r \leqslant n$ we call the matrix $W_r = W_{\mathrm{full}} \begin{pmatrix} O \\ I_r \end{pmatrix}$ the *r-tail* of the SVD of the matrix M.

$Q(M)$ denotes the $m \times n$ factor Q in the QR factorization of an $m \times n$ matrix M of full rank where $m \geqslant n$ and the factor R has positive diagonal entries (in this case the factorization is unique).

$\mathrm{range}(M) = \{\mathbf{z} : \mathbf{z} = M\mathbf{y}\}$ denotes the range of a matrix M. Its orthogonal complement $\mathbb{N}(M) = \{\mathbf{x} : M\mathbf{x} = \mathbf{0}\}$ is the *null space* of the matrix, made up of its *null vectors* $\mathbf{x}$.

We call a matrix B a *complete annihilator* or just a *ca* of a matrix M and denote it $\mathrm{ca}(M)$ if $\mathrm{range}(B) = \mathbb{N}(M)$.

A matrix M of full column rank is a *matrix basis* for $\mathrm{range}(M)$. $\mathrm{nmb}(M)$ or a *nmb* of M is a null matrix basis, that is a matrix basis for the null space $\mathbb{N}(M)$. A $\mathrm{ca}(M)$ is a $\mathrm{nmb}(M)$ if it has full column rank.

$\mathbb{S}$ is an invariant subspace or eigenspace of a matrix M if $M\mathbb{S} \subseteq \mathbb{S}$.

$\mathrm{dist}(\mathbb{S}, \mathbb{T}) = \max_{\mathbf{s} \in \mathbb{S}, \|\mathbf{s}\| = 1} \min_{\mathbf{t} \in \mathbb{T}} \|\mathbf{s} - \mathbf{t}\|$ is the distance between two linear spaces $\mathbb{S}$ and $\mathbb{T}$.

$\{\lambda, \mathbb{X}, \mathbb{Y}\}$ is an eigentriple and $\{\lambda, \mathbb{Y}\}$ is an eigenpair of a matrix M if λ is its eigenvalue, whereas $\mathbb{X}$ and $\mathbb{Y}$ are the associated left and right eigenspaces. For two matrices X and Y we also call $\{\lambda, X, Y\}$ an eigentriple and $\{\lambda, Y\}$ an eigenpair of the matrix M if $\mathrm{range}(X) = \mathbb{X}$ and $\mathrm{range}(Y) = \mathbb{Y}$.

The basic concepts and results on computations with matrices having displacement structure of Toeplitz, Hankel, Cauchy, and Vandermonde types can be found in [21] and the bibliography therein.

2.2. Random sampling, random matrices, and Gaussian random variables

$|\Delta|$ is the cardinality of a set Δ. *Random sampling* of elements from a set Δ is their selection from this set at random, independently of each other, and under the uniform probability distribution on the set. A matrix is *random* if its entries are randomly sampled from a fixed set Δ, e.g., the set of all double precision numbers with the exponents in a fixed range, for numerical computations. A $k \times l$ *random unitary* matrix is the $k \times l$ Q-factor $Q(M)$ in the thin QR factorization of random $k \times l$ matrix M of full rank where the R-factor $R(M)$ has positive diagonal entries. (QR factorization reveals whether a matrix has full rank.)

Lemma 2.1 ([9] (cf. also [34,38])). *For a set Δ of cardinality $|\Delta|$ in a ring $\mathcal{R}$, let a polynomial in m variables not vanish identically on the set Δ^m, let it have the total degree d, and let the values of its variables be randomly sampled from the set Δ. Then the polynomial vanishes with a probability of at most $\frac{d}{|\Delta|}$.*

3. Three standard algorithms for computations in the null spaces

Suppose we Seek B, a nmb for an $n \times n$ matrix M that has a rank ρ and the nullity $\nu = n - \rho$. Having a full SVD $M = V_{\text{full}}\Sigma_{\text{full}}W_{\text{full}}^H$ computed, we can choose $B = W_{\text{full}}\begin{pmatrix} O \\ I_\nu \end{pmatrix}$. Likewise, having *QRP* (resp. *PULP*$_1$) factorization of the matrix M^H computed, we can choose $B = Q\begin{pmatrix} O \\ I_\nu \end{pmatrix}$ (resp. $B = P\begin{pmatrix} O \\ I_\nu \end{pmatrix}$). In the above *QRP* and *PLUP*$_1$ factorizations, L can be any matrix, Q denotes an $n \times n$ unitary matrix, P and P_1 denote some $n \times n$ permutation matrices, such that $P^TP = P_1^TP_1 = I$, and R and U denote $n \times n$ matrices of the form $(W, O)^T$ for $n \times (n - \nu)$ matrices W.

Application of orthogonalization and SVD above is more costly (and more reliable), but even pivoting "usually degrades the performance" [12, page 119], readily destroys matrix structure and sparseness, and threatens or undermines application of block matrix algorithms. For example, in the case of $n \times n$ input matrices M with structure of Toeplitz or Hankel type application of pivoting increases the arithmetic computational cost from $O(n \log^2 n)$ flops to the order of n^2.

4. Multiplicative preprocessing for null space computations

Suppose an $n \times n$ matrix $M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$ has nonsingular $\rho \times \rho$ leading (that is northwestern) block submatrix M_{00}. Then a single block Gauss–Jordan step outputs the block factorization

$$M = \begin{pmatrix} I_\rho & O \\ M_{10}M_{00}^{-1} & I_\nu \end{pmatrix} \begin{pmatrix} M_{00} & O \\ O & S \end{pmatrix} \begin{pmatrix} I_\rho & M_{00}^{-1}M_{01} \\ O & I_\nu \end{pmatrix}, \tag{4.1}$$

where $S = S(M_{00}, M) = M_{11} - M_{01}M_{00}^{-1}M_{01}$ denotes the Schur complement of the block M_{00} in M. We immediately verify the following lemma.

Lemma 4.1. *Suppose the $n \times n$ matrix M above has rank ρ and so does its $\rho \times \rho$ leading block M_{00}. Write $\nu = n - \rho$. Then $S = O$ and $B = B(M) = \begin{pmatrix} -M_{00}^{-1}M_{01} \\ I_\nu \end{pmatrix}$ is a* nmb(M).

For a nonsingular matrix M we can shift to the matrix M^HM or MM^H to relax the assumption that the matrix M_{00} is nonsingular at the price of squaring the condition number. We pay no such a price if we shift to the matrix $W = C_lMC_r$ for two appropriately structured random matrices C_l and C_r, defined by random parameters sampled from a large set Δ. One can deduce from Lemma 2.1 that with this *structured multiplicative preprocessing*, the $i \times i$ leading submatrices $W^{(i)}$ of the resulting matrix $W = \begin{pmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{pmatrix}$ are nonsingular for all $i \leqslant \rho$ with a probability converging to one as $|\Delta| \to \infty$ (see specific probability estimates in [21, Section 5.6]). In particular, if the matrix M has the displacement structure of Toeplitz, Hankel, Vandermonde or Cauchy type, then we can choose the multipliers C_l and C_r that ensure the same structure of any of these types for the matrix W. In this case the superfast divide-and-conquer MBA algorithm (cf. [21, Chapter 5]) only needs $O(n \log^2 n)$ flops to compute shortest displacement generators that represent the matrices W_{00}^{-1}, $-W_{00}^{-1}W_{01}$, $B(W) = \begin{pmatrix} -W_{00}^{-1}W_{01} \\ I_\nu \end{pmatrix}$, and $B(M) = C_rB(W)$, which is a nmb(M) as long as the matrices C_l and C_r are nonsingular.

According to the test results in [30], the above preprocessing tends to keep its power even under *weak randomization*, where the matrices C_l and C_r are circulant and are filled with the values -1 and 1 chosen at random. Moreover in the tests this preprocessing tended to be preconditioning, that is the leading submatrices $W^{(i)}$ for all $i \leqslant \rho$ tended not only to be nonsingular but also to have condition numbers of at most the same order as $\text{cond}(M) = \frac{\sigma_1(M)}{\sigma_\rho(M)}$. Such properties have been proved in [30] for general Gaussian random matrices C_l and C_r. The tests in [30] sometimes showed minor increase of

the value cond(W) versus cond(M) and the respective minor loss of accuracy in the computed inverse W_{00}^{-1}, but the full precision output was always recovered in one or two steps of iterative refinement.

5. Additive preprocessing for null space computations

We apply additive preprocessing based on the following results.

Theorem 5.1. *Suppose*

(a) *M is an $n \times n$ matrix having a rank ρ and the nullity $\nu = n - \rho$,*
(b) *U and V are two matrices of size $n \times r$, and*
(c) *the matrix*

$$K = M + UV^H \tag{5.1}$$

is nonsingular. Then

$$r \geqslant \text{rank}(U) \geqslant \nu, \tag{5.2}$$

$$\mathbb{N}(M) \subseteq \text{range}(K^{-1}U). \tag{5.3}$$

Furthermore if

$$\text{rank}(UV^H) = \nu, \tag{5.4}$$

then

$$\text{range}(K^{-1}U) = \mathbb{N}(M), \tag{5.5}$$

$$V^H K^{-1} U = I_\nu. \tag{5.6}$$

Proof. See Theorem 3.1 in [28]. □

The following theorem is immediately verified.

Theorem 5.2. *Under the assumptions (a)–(c) of Theorem 5.1 we have* range$(K^{-1}UX) = \mathbb{N}(M)$ *if and only if* range$(X) = \mathbb{N}(MK^{-1}U)$ *and consequently $K^{-1}U$ is a* ca(M) *if and only if $MK^{-1}U = 0$.*

Theorem 5.3. *Under the assumptions (a)–(c) of Theorem 5.1, $\mathbb{N}(MK^{-1}U) = \mathbb{N}(I_\nu - V^H K^{-1}U)$ if the matrix U has full rank.*

Proof. See Theorem 4.1 in [32] or Corollary 3.2 in [28]. □

Randomized computation of the nullity ν and a ca(M) can employ the following properties in Theorems 5.1–5.3.

1. For $n \times r$ matrices U and V, the matrix $K = M + UV^H$ is singular if $r < \nu$ (in virtue of bounds (5.2)) but is likely to be nonsingular if $r \geqslant \nu$ and if the matrices U and V are random or even just random within a fixed class of structured matrices (see [25] for specific probability estimates, based on Lemma 2.1).
2. Suppose the matrix K is nonsingular, and so range$(K^{-1}U) \supseteq \mathbb{N}(M)$. Then
$$B = K^{-1}U \tag{5.7}$$
is a ca(M) if and only if $MK^{-1}U = 0$.
3. Suppose the matrix K is nonsingular and $MK^{-1}U \neq 0$. Then we have rank$(UV^H) > \nu$ and range$(K^{-1}U) \supset \mathbb{N}(M)$. Furthermore in this case $K^{-1}UX$ is a ca(M) if X is a ca$(MK^{-1}U) =$ ca$(I_r - V^H K^{-1}U)$.

The transitions $M \Longrightarrow I_r - V^H K^{-1} U$ can be viewed as aggregation and can be extended recursively. (See [28, Section 6.2] on some examples of aggregation for matrix computations and tensor decompositions and recall recursive hierarchial aggregation in [17], evolved into Algebraic Multigrid.)

According to the formal analysis in [25] and [28, Theorem 3.12] we can expect that the ratio $\frac{\mathrm{cond}(M)}{\mathrm{cond}(K)}$ is nlns (or equivalently that $\mathrm{cond}(K)$ has the order $\frac{\sigma_1(M)}{\sigma_{n-r}(M)}$) if the matrix K is nonsingular, if the ratio $\frac{\|M\|}{\|UV^H\|}$ is nlns, and if U and V are Gaussian random matrices. The same property of the ratio was consistently observed in the extensive experiments in [25] with *weakly randomized additive preprocessors* for $U = V = c(\pm I_r, \dots, \pm I_r, \pm I'_{n,r})^T, I'_{n,r} = (I_{r'}, 0)^T, r' = n \bmod r = n - hr, 0 \leqslant r' < r$, and $c^2 h \approx \|M\|$, where each $\pm$ denoted the sign $-$ or $+$ chosen at random.

If the matrix M is ill conditioned, whereas the matrix K is well conditioned, then the matrices $MK^{-1}U$ and $I_\nu - V^H K^{-1} U$ tend to have small norms, large condition numbers, or both, and thus one must compute these matrices with higher accuracy, e.g., by applying the extended iterative refinement from [23] to computing the matrix $K^{-1}U$. The gain from preconditioning is the reduction of the computations to the case of a well conditioned input matrix K, so that we can apply and extend iterative refinement (cf. [23]).

The test results in Tables 6–9 confirm the efficiency of the respective algorithms.

We conclude this section by representing multiplicative preprocessing in Section 4 as additive preprocessing of a 2×2 block matrix. Assume that an $n \times n$ matrix $M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$ of rank ρ has nonsingular $\rho \times \rho$ leading block M_{00}. Then $B = \begin{pmatrix} -M_{00}^{-1} M_{01} \\ I_\nu \end{pmatrix}$ is a $\mathrm{nmb}(M)$ and $\nu = n - \rho$ in virtue of Lemma 4.1. Let us also deduce this fact from Theorem 5.1. Namely, write $U = \begin{pmatrix} O \\ I_\nu \end{pmatrix}, V^H = (-M_{10}, I_\nu - M_{11})$, and $\nu = n - \rho$ and obtain the nonsingular matrix $K = M + UV^H = \begin{pmatrix} M_{00} & M_{01} \\ O & I_\nu \end{pmatrix}$. Then Theorem 5.1 implies that

$$B = K^{-1} \begin{pmatrix} O \\ I_\nu \end{pmatrix} = \begin{pmatrix} -M_{00}^{-1} M_{01} \\ I_\nu \end{pmatrix} \tag{5.8}$$

is a $\mathrm{nmb}(M)$.

6. Preprocessing by means of randomized augmentation

Given an $n \times n$ singular matrix M and its rank ρ, define *preprocessing by means of augmentation*

$$M \to A = \begin{pmatrix} M & P_{01} \\ O & \theta I_\nu \end{pmatrix} \to K = A + UV^H = \begin{pmatrix} M & P_{01} \\ P_{10} & \theta I_\nu \end{pmatrix}. \tag{6.1}$$

Here $\nu = n - \rho$, $U = \begin{pmatrix} O \\ P_{10} \end{pmatrix}$, $V^H = (I_n, O)$, and we choose the scalar θ and scaled Gaussian random matrices P_{01} and P_{10} such that the ratios $\theta / \|M\|$, $\|P_{01}\| / \|M\|$, $\|P_{10}\| / \|M\|$, and $\|M\| / \|K\|$ are nlns. (The matrix K is Hermitian if so is the matrix M, if θ is real, and if $P_{01} = P_{10}^H$.) For a singular matrix M we can deduce from Lemma 2.1 that the above augmentation produces a nonsingular matrix K with a high probability (specified in [30]). If the matrix K is indeed nonsingular, then the matrix $\begin{pmatrix} B \\ B_1 \end{pmatrix} = K^{-1} \begin{pmatrix} O \\ P_{10} \end{pmatrix}$ is a $\mathrm{ca}(A)$ and therefore the matrix $B = (I_n, O) K^{-1} \begin{pmatrix} O \\ P_{10} \end{pmatrix}$ is a $\mathrm{ca}(M)$.

Furthermore it is proved in [30] that the condition number $\mathrm{cond}(K)$ is expected to have the same order as $\mathrm{cond}(M) = \sigma_1(M)/\sigma_\rho(M)$. If we are given a nonsingular matrix $\widetilde{M} \approx M$ and augment it as above to obtain the matrix $\widetilde{K} = \begin{pmatrix} \widetilde{M} & P_{01} \\ P_{10} & \theta I_\nu \end{pmatrix}$, then clearly $\mathrm{cond}(\widetilde{M}) \gg \mathrm{cond}(M)$, whereas $\mathrm{cond}(\widetilde{K}) \approx$

$\mathrm{cond}(K) \approx \mathrm{cond}(M)$, so that the transition $\widetilde{M} \to \widetilde{K}$ is preconditioning, in good accordance with the test results in Table 10. In fact the tests consistently show preconditioning power of even *weakly randomized augmentation* $\widetilde{M} \to \overline{K} = \begin{pmatrix} \widetilde{M} & \overline{P}_{01} \\ \overline{P}_{10} & \overline{P}_{11} \end{pmatrix}$ where we allow only a small number of random parameters in the matrices $\overline{P}_{01}, \overline{P}_{10}$ and $\overline{P}_{11}$ and choose these parameters to keep the structure of the input matrix M intact in the above transition to the matrix $\overline{K}$. Note that a $\nu \times \nu$ random matrix $\overline{P}_{11}$ is nonsingular with a probability close to one, and if it is indeed nonsingular, then $\overline{K} = \mathrm{diag}\left(I_{n-\nu}, \frac{1}{\theta}\overline{P}_{11}\right)\widetilde{K}$ where $\widetilde{K} = \begin{pmatrix} \widetilde{M} & P_{01} \\ P_{10} & \theta I_\nu \end{pmatrix}$ and $\mathbb{N}(\overline{K}) = \mathbb{N}(\widetilde{K})$, for $\theta \neq 0, P_{01} = \overline{P}_{01}$, and $P_{10} = \theta \overline{P}_{11}^{-1}\overline{P}_{10}$.

Now suppose the value $\rho = \mathrm{rank}\, M$ is not known. Then we can search for it by extending the recipes in the previous section based on Theorems 5.1–5.3. For $\nu < n - \rho$ the matrix K is definitely singular, and then we should increment the integer ν and recompute this matrix. If $\nu \geqslant n - \rho$ and the matrices P_{01} and P_{01} are random or random structured, then the matrix K is likely to be nonsingular. If indeed it is nonsingular and if $\nu = n - \rho$, then the matrix B is expected to be a $\mathrm{ca}(M)$. If $\mathrm{rank}\, M > n - \nu$ and if the matrix K is nonsingular, then the same algorithm would output a matrix B whose range would contain the null space $\mathbb{N}(M)$. In this case B is a $\mathrm{ca}(M)$ if and only if $MB = O$. If $MB \neq O$ we can reapply the same algorithm to the aggregate MB of a smaller size to compute the matrices X (a ca MB)) and BX or $Q(BX)$ (a $\mathrm{ca}(M)$) (cf. Theorem 5.2).

7. Estimates for the impact of input perturbations

Let us estimate the impact of input perturbations in the cases of computations with multiplicative and additive preprocessing. The latter estimates can be readily extended to preprocessing via augmentation either directly or by using the link to additive preprocessing in [28, Section 4].

For a matrix M multiplicative preprocessing in Section 4 produces the matrices $W = C_l M C_r$, $B(W) = \begin{pmatrix} -W_{00}^{-1}W_{01} \\ I_\nu \end{pmatrix} = \mathrm{nmb}(W)$, and $B(M) = C_r B(W) = \mathrm{nmb}(M)$, provided that the matrices C_l and C_r are nonsingular.

Now suppose that $\widetilde{M} \approx M$ and $\widetilde{W} \approx W$, write $F = -C_r W_{00}^{-1} W_{01}$ and $\widetilde{F} = -C_r \widetilde{W}_{00}^{-1}\widetilde{W}_{01}$, and obtain that $\delta(F) = \widetilde{F} - F = -C_r \delta(W_{00}^{-1}W_{01}) = -C_r(\delta(W_{00}^{-1})W_{01} + \widetilde{W}_{00}^{-1}\delta(W_{01}))$. Therefore

$$\|\delta(F)\| \leqslant \|C_r\|(\|\delta(W_{00}^{-1})\|\|W_{01}\| + \|\widetilde{W}_{00}^{-1}\|\|\delta(W_{01})\|), \tag{7.1}$$

$$\|W_{01}\| \leqslant \|C_r\|(\|\widetilde{M}\| + \|\delta M\|)\|C_l\|, \|\widetilde{W}_{00}^{-1}\| \leqslant \|C_l^{-1}\|\|\widetilde{M}_{00}^{-1}\|\|C_r^{-1}\|, \tag{7.2}$$

$$\|\delta(W_{0j})\| \leqslant \|C_r\|\|\delta(M_{0j})\|\|C_l\| \leqslant \|\delta(M)\|\|C_r\|\|C_l\|, j = 0, 1. \tag{7.3}$$

Further assume that $\delta_{00} = \|\widetilde{W}_{00}^{-1}\delta(W_{00})\| < 1$ and obtain that

$$\|\delta(W_{00}^{-1})\| \leqslant \frac{1}{1 - \delta_{00}}\|\delta(W_{00})\|\|\widetilde{W}_{00}^{-1}\|^2 \tag{7.4}$$

(cf. [12, Theorem 2.3.4] for $A = \widetilde{W}_{00}, E = -\delta(W_{00})$).

Estimates (7.1)–(7.4) together imply that $\|\delta(F)\| = O(\|\delta(M)\|)$.

Remark 7.1. Suppose the Schur complement S in Eq. (4.1) is nonsingular. Then we can invert both sides of this equation and obtain that

$$M^{-1} = \begin{pmatrix} I_\rho & -M_{00}^{-1}M_{01} \\ 0 & I_\nu \end{pmatrix} \begin{pmatrix} M_{00}^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I_\rho & 0 \\ -M_{10}M_{00}^{-1} & I_\nu \end{pmatrix}, \tag{7.5}$$

and so $\|M^{-1}\| \leqslant \max\{\|M_{00}^{-1}\|, \|(S^{-1}\|\}(1 + \|M\|\|M_{00}^{-1}\|)^2$, whereas we have $\|(C_l M C_r)^{-1}\| \leqslant \|C_l^{-1}\| \|M^{-1}\|\|C_r^{-1}\|$.

In the case of additive preprocessing in Section 5 we have the following simple estimate.

Theorem 7.1. *For the matrices M, U, V, and K in Theorem 5.1 and an $n \times n$ matrix $\Delta = \delta(M)$, assume that $\mathbf{y} \in \mathbb{N}(M)$, $\|\mathbf{y}\| = 1$, and the matrix $K + \Delta = M + \Delta + UV^H$ is nonsingular. Then* $\operatorname{dist}(\mathbf{y}, \operatorname{range}((K + \Delta)^{-1}U)) \leqslant \|(K + \Delta)^{-1}\Delta\| \leqslant \|(K + \Delta)^{-1}\|\|\Delta\|$.

Proof. We have $(K + \Delta)\mathbf{y} = \Delta\mathbf{y} + UV^H\mathbf{y}$, and therefore $\mathbf{y} = (K + \Delta)^{-1}\Delta\mathbf{y} + (K + \Delta)^{-1}UV^H\mathbf{y}$. The theorem follows because $(K + \Delta)^{-1}UV^H\mathbf{y} \in \operatorname{range}((K + \Delta)^{-1}U)$. □

For a well conditioned nonsingular matrix K and a small-norm perturbation matrix $\Delta = \delta(M) = \delta(K)$, the theorem implies that the range of the matrix $(K + \Delta)^{-1}U$ approximates the null space $\mathbb{N}(M)$ within $O(\|\Delta\|)$.

8. Approximation by nearby structured or lower rank matrices

Similarly to the previous section assume a nonsingular ill conditioned $n \times n$ input matrix $\widetilde{M}$ represented as $\widetilde{M} = M + \delta(M)$ where $\operatorname{rank}(M) = \rho < n$, the norm $\|\delta(M)\|$ is small, the matrix $\widetilde{M}$ has *numerical rank* ρ and has *numerical nullity* $\operatorname{nnul}(M) = \nu = n - \rho$, that is has exactly ν singular values that are small relatively to the norm $\|M\|$. Application of error-free algorithms to this matrix models numerical application of the same algorithms to the matrix M.

Hereafter for a matrix function $F = f(M)$, we write $\widetilde{F} = f(\widetilde{M})$ and $\delta(F) = \widetilde{F} - F$.

The algorithms from Sections 3–6 applied to a matrix $\widetilde{M}$ output a matrix $\widetilde{B}$ expected to approximate an $n \times \nu$ matrix $B = \operatorname{nmb}(M)$, and if it does, then $\operatorname{range}(\widetilde{B})$ approximates the ν-tail of the SVD of the matrix $\widetilde{M}$. This immediately leads us to the approximation of a nearly rank deficient matrix by a smaller rank matrix $\widetilde{M}(I - \widetilde{Q}\widetilde{Q}^H)$ such that $\widetilde{Q} = \widetilde{Q}(B)$ is a unitary approximate $\operatorname{nmb}(M)$ for B in (5.7), (5.8), or (6.1) where M is replaced by $\widetilde{M}$. An alternative expression in [28, Section 7.2] relies on a dual variation of the Sherman–Morrison–Woodbury classical formula for matrix inversion [12, page 50]. (Hereafter we use the abbreviation *SMW*.)

As a special case we can apply such techniques to approximate the displacement $\widetilde{M} = \operatorname{disp}(\widetilde{A})$ of a matrix $\widetilde{A}$ by the matrix $M = \operatorname{disp}(A)$ of a smaller rank (provided that there exists such a matrix M). Then we can approximate the input matrix $\widetilde{A}$ by a structured matrix A recovered from its displacement $M = \operatorname{disp}(A)$.

The respective computations can be reduced to the solution of linear systems of equations with the matrix $\widetilde{K}$ given by $\widetilde{K}Y = U$ for $\widetilde{K} = \widetilde{M} + UV^H$, $\widetilde{K} = K + \delta(K)$, and K in Sections 5 or 6, and so the perturbations of the outputs have the norms in $O(\|\delta M\|)$ provided the auxiliary linear systems are well conditioned (cf. [12,13,35,36]).

Table 13 displays the results of our experimental computations for this section.

9. Extension to the solution of a nonhomogeneous linear system

We can readily extend our null space algorithms to a nonhomogeneous linear system $M\mathbf{y} = \mathbf{b}$, for $\mathbf{b} \neq \mathbf{0}$: observe that the solution vector $\mathbf{y}$ is a subvector of the null vector $\mathbf{z} = (\mathbf{y}^T, 1/\theta)^T$ of the matrix $(M, -\theta\mathbf{b})$ for a scalar $\theta \neq 0$. We refer the reader to the second last paragraph of Section 5 and to the paper [23] on handling the numerical problems that arise where the linear system $M\mathbf{y} = \mathbf{b}$ is nonsingular and ill conditioned and to Section 12.4 on the implementation of this approach and experiments that demonstrate its power.

10. Applications to eigen-solving

10.1. The inverse iteration for eigen-solving, RQs and SQs

The Rayleigh quotient iteration (also called the inverse iteration [12]) is a popular eigen-solver. Given a square matrix M and an approximation λ_0 to its simple eigenvalue λ, one computes the matrix $M_0 = M - \lambda_0 I$, fixes a vector $\mathbf{y}_0$, and recursively updates approximate eigenpairs $\{\lambda_i, \mathbf{y}_i\}$ for $i = 0, 1, \ldots$ as follows:

$$M_i\mathbf{z}_i = \mathbf{y}_i, \tag{10.1}$$

$$\delta_i = \frac{\mathbf{z}_i^H M_i \mathbf{z}_i}{\mathbf{z}_i^H \mathbf{z}_i}, \tag{10.2}$$

$$M_{i+1} = M_i - \delta_i I, \quad \mathbf{y}_{i+1} = \mathbf{z}_i/s_i, \quad \lambda_{i+1} = \lambda_i + \delta_i$$

where s_i are positive scalars such that the ratios $\|\mathbf{z}_i\|/s_i$ are nlns for all i.

Hereafter we use the abbreviations *RQs* for the *Rayleigh quotients* $\frac{\mathbf{z}_i^H M_i \mathbf{z}_i}{\mathbf{z}_i^H \mathbf{z}_i}$ in (10.2) and *SQs* for the *simple quotients* $\frac{\mathbf{e}_j^H M_i \mathbf{z}_i}{\mathbf{e}_j^H \mathbf{z}_i}$ in the following alternative to (10.2),

$$\delta_i = \frac{\mathbf{e}_j^T M_i \mathbf{z}_i}{\mathbf{e}_j^T \mathbf{z}_i}, \quad \mathbf{e}_j^T \mathbf{z}_i \neq 0. \tag{10.3}$$

We choose the integer j that maximizes the value $|\mathbf{e}_j^T \mathbf{z}_i|$ in a fixed or random set $\mathbb{J}$ of integers j (e.g., over three or five random integers or just over the set $\{1, \lceil n/2 \rceil, n\}$). Algorithms 10.1(rq) and 10.1(sq) below specify the RQ iteration (10.1), (10.2) and SQ iteration (10.1), (10.3), respectively. Both iterations can employ the standard stopping criterion

$$\|M_i\mathbf{z}_i\| \leqslant t\|\mathbf{z}_i\|, \tag{10.4}$$

where t is either a fixed tolerance or $t = t'|\lambda_i|$ for a fixed tolerance t'. To save some flops one can skip checking this criterion where $|\delta_{i-1}| > \theta t$ for a fixed positive scalar θ and similarly in all our eigen-solvers.

Under (10.1) one should substitute $\mathbf{y}_i = M_i\mathbf{z}_i$ into Eqs. (10.2)–(10.4) to obtain $\delta_i = \frac{\mathbf{z}_i^H \mathbf{y}_i}{\mathbf{z}_i^H \mathbf{z}_i}$ instead of (10.2), $\delta_i = \frac{\mathbf{e}_j^T \mathbf{y}_i}{\mathbf{e}_j^T \mathbf{z}_i}$ instead of (10.3), and $\|\mathbf{y}_i\| \leqslant t\|\mathbf{z}_i\|$ instead of (10.4), thus saving the vector $\mathbf{y}_i = M_i\mathbf{z}_i$ rather than recomputing it.

The iteration is equivalent to Newton's eigen-solving iteration and has local quadratic convergence [37,31], [36, Section 2.2.1].

The RQ in (10.2) can be considered an average over all subscripts $j, j = 1, 2, \ldots, n$, for the SQs in (10.3), and so for random choice of the integers j the SQs are expected to have the same order as the RQs. Consequently quadratic rate of local convergence of RQ iteration (10.1), (10.2) is expected to hold also for the SQ iteration (10.1), (10.3) under a random choice of the integers j. In the tests for global convergence (initiated far from the solution), the SQ iteration converged slightly slower than the RQ iteration, but this was always more than compensated by the simplicity of the SQ iteration steps. Similar patterns characterize using RQs and SQs in our algorithms in the next subsections.

Algorithm 10.1. The SQ iteration.

INPUT: an $n \times n$ matrix M, an approximation λ_0 to its simple eigenvalue, a positive integer N, and a tolerance t.

OUTPUT: either FAILURE or an approximate eigenpair $\{\lambda, \mathbf{y}\}$ of the matrix M such that $\|M\mathbf{y} - \lambda\mathbf{y}\| \leqslant t\|\mathbf{y}\|$.

INITIALIZATION: Set $i \leftarrow 0$ and $k \leftarrow 0$, and $M_0 \leftarrow M - \lambda_0 I$ and fix a normalized vector $\mathbf{y}_0$, $\|\mathbf{y}_0\| = 1$ and a set $\mathbb{J}$ of integers in the range $[1, n]$.

COMPUTATIONS:

1. If $k \geqslant N$, output FAILURE and stop. Otherwise compute the vector $\mathbf{z}_i = M_i^{-1}\mathbf{y}_i$. Compute the value $\lambda_i = \mathbf{e}_1^T(M_i - M)\mathbf{e}_1$, output the pair $\{\lambda, \mathbf{y}\} = \{\lambda_i, \mathbf{y}_i\}$ and stop if $\|\mathbf{y}_i\| \leqslant t\|\mathbf{z}_i\|$.
2. Otherwise compute an integer j maximizing the value $\mathbf{e}_j^T\mathbf{z}_i$ over the set $\mathbb{J}$. If $\mathbf{e}_j^T\mathbf{z}_i = 0$, output FAILURE and stop.

3. Otherwise compute the ratio $\delta_i = \frac{\mathbf{e}_j^T \mathbf{y}_i}{\mathbf{e}_j^T \mathbf{z}_i}$, the matrix $M_{i+1} = M_i - \delta_i I$, and the vector $\mathbf{y}_{i+1} = \mathbf{z}_i / s_i$ for a nonzero scalar s_i such that the norm $\|\mathbf{y}_{i+1}\|$ is nlns (e.g., $s_i = \|\mathbf{z}_i\|$), set $i \leftarrow i+1$ and $k \leftarrow k+1$, and reapply Stage 1.

We refer to this SQ iteration as Algorithm 10.1(sq). By expressing δ_i as the RQ $\frac{\mathbf{z}_i^T \mathbf{y}_i}{\mathbf{z}_i^T \mathbf{z}_i}$ we arrive at Algorithm 10.1(rq), the RQ iteration.

10.2. Inverse iteration with additive preprocessing

In an eigenpair $(\lambda, \mathbf{y})$ of a matrix M the eigenvector $\mathbf{y}$ is a null vector of the shifted matrix $M - \lambda I$, and this prompts us to apply our null space algorithms at the stage of the solution of linear systems (10.1) for updating the eigenvectors. We specify application of scaled randomized additive preprocessing, but one can apply augmentation instead.

Systems (10.1) are singular for $\lambda_i = \lambda$ and become ill conditioned as λ_i converges to λ. Therefore they resist application of such effective iterations as the Conjugate Gradient algorithms and iterative refinement. With randomized preprocessing, however, we fix this deficiency.

Suppose that λ is a simple isolated eigenvalue and rewrite expressions (10.1) by applying the SMW formula,

$$\mathbf{z}_i = K_i^{-1}(1 + g_i^{-1}\mathbf{u}_i\mathbf{v}_i^H K_i^{-1})\mathbf{y}_i, \quad \text{for } K_i = M_i + \mathbf{u}_i\mathbf{v}_i^H, \quad g_i = 1 - \mathbf{v}_i^H K_i^{-1}\mathbf{u}_i. \tag{10.5}$$

Here $\mathbf{u}_i$ and $\mathbf{v}_i$ are random vectors (or $\mathbf{u}_i = \mathbf{v}_i$ is a single random vector) such that the matrix K_i is nonsingular and the ratio $\frac{\|\mathbf{u}_i\mathbf{v}_i^H\|}{\|M_i\|}$ is nlns. We refer to the resulting modifications of RQ and SQ iterations as the *RQ/SMW* and *SQ/SMW iterations* and also as Algorithms 10.1(sq/smw) and 10.1(rq/smw), respectively. Mathematically expressions (10.1) and (10.5) define the same vector $\mathbf{z}_i$, so that the RQ/SMW and SQ/SMW iterations have local quadratic convergence as well.

According to the study in [25] the matrix K_i is expected to be well conditioned for λ_i near a simple and isolated eigenvalue λ.

In an alternative iteration we keep the expression $K_i = M_i + \mathbf{u}_i\mathbf{v}_i^H$ and the recipes for choosing the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ but replace Eq. (10.5) as follows (cf. Theorem 5.1),

$$K_i\mathbf{z}_i = \mathbf{u}_i. \tag{10.6}$$

We call the respective extensions of the RQ and SQ iterations the *PRQ and PSQ iterations* with the abbreviation "P" for "preprocessed".

Eq. (10.6) implies that $M_i\mathbf{z}_i = K_i\mathbf{z}_i - \mathbf{u}_i\mathbf{v}_i^H\mathbf{z}_i = \mathbf{u}_i - \mathbf{u}_i\mathbf{v}_i^H\mathbf{z}_i = g_i\mathbf{u}_i$ for $g_i = 1 - \mathbf{v}_i^H\mathbf{z}_i = 1 - \mathbf{v}_i^H K_i^{-1}\mathbf{u}_i$ from Eq. (10.5). We can substitute the expression $M_i\mathbf{z}_i = g_i\mathbf{u}_i$ into Eqs. (10.2)–(10.4) and obtain the equivalent expressions $\delta_i = g_i\frac{\mathbf{z}_i^H\mathbf{u}_i}{\mathbf{z}_i^H\mathbf{z}_i}$ (cf. (10.2)), $\delta_i = g_i\frac{\mathbf{e}_j^H\mathbf{u}_i}{\mathbf{e}_j^H\mathbf{z}_i}$ for $\mathbf{e}_j^H\mathbf{z}_i \neq 0$ (cf. (10.3)), and $|g_i|\|\mathbf{u}_i\| \leqslant t\|\mathbf{z}_i\|$ (cf. (10.4)). Substitute $\mathbf{y}_i = \mathbf{z}_i/s_i$ and $\bar{g}_i = g_i/s_i = \frac{1}{s_i} - \mathbf{v}_i^H\mathbf{y}_i$ for a nonzero scalar s_i and obtain

$$\delta_i = \bar{g}_i\frac{\mathbf{y}_i^H\mathbf{u}_i}{\mathbf{y}_i^H\mathbf{y}_i}, \quad \delta_i = \bar{g}_i\frac{\mathbf{e}_j^T\mathbf{u}_i}{\mathbf{e}_j^T\mathbf{y}_i}, \quad |\bar{g}_i|\|\mathbf{u}_i\| \leqslant t\|\mathbf{y}_i\|, \tag{10.7}$$

respectively. The following algorithm employs these equations.

Algorithm 10.2. PSQ iteration.

INPUT and OUTPUT as in Algorithm 10.2.

INITIALIZATION: Set $i \leftarrow 0$, $k \leftarrow 0$, and $M_0 \leftarrow M - \lambda_0 I$ and fix a moderately large positive value γ.

COMPUTATIONS:

1. If $k \geqslant N$, output FAILURE and stop. Otherwise generate a pair of $n \times \nu$ random vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ scaled so that $\frac{1}{\gamma} < \frac{\|\mathbf{u}_i\mathbf{v}_i^H\|}{\|M_i\|} < \gamma$. Compute the matrix $K_i = M_i + \mathbf{u}_i\mathbf{v}_i^H$. If it is singular, set $k \leftarrow k+1$ and reapply Stage 1.
2. Otherwise fix a positive scalar s_i and compute the vectors $\mathbf{z}_i = K_i^{-1}\mathbf{u}_i$ and $\mathbf{y}_i = \mathbf{z}_i/s_i$ and the scalar $\bar{g}_i = \frac{1}{s_i} - \mathbf{v}_i^H\mathbf{y}_i$ for a fixed scalar s_i. Compute the value $\lambda_i = \mathbf{e}_1^T(M_i - M)\mathbf{e}_1$, output the pair $(\lambda, \mathbf{y}) = (\lambda_i, \mathbf{y}_i)$ and stop if $\|\bar{g}_i\mathbf{u}_i\| \leqslant t\|\mathbf{y}_i\|$.
3. Proceed as in Stage 2 in Algorithm 10.2.
4. Otherwise compute the value $\delta_i = \bar{g}_i\frac{\mathbf{e}_j^T\mathbf{u}_i}{\mathbf{e}_j^T\mathbf{y}_i}$ and the matrix $M_{i+1} = M_i - \delta_i I$. Set $i \leftarrow i+1$ and $k \leftarrow k+1$ and reapply Stage 1.

We refer to this PSQ iteration as Algorithm 10.2(sq). By expressing δ_i as $\bar{g}_i\frac{\mathbf{y}_i^H\mathbf{u}_i}{\mathbf{y}_i^H\mathbf{y}_i}$ we obtain Algorithm 10.2(rq), the PRQ iteration.

By replacing the stopping criterion and the expression for δ_i in these two algorithms with $\|M_i\mathbf{y}_i\| \leqslant t\|\mathbf{y}_i\|$ (cf. (10.4)) and choosing either $\delta_i = \frac{\mathbf{e}_j^T M_i\mathbf{y}_i}{\mathbf{e}_j^T\mathbf{y}_i}$ for an integer j such that $\mathbf{e}_j^T\mathbf{y}_i \neq 0$ (cf. (10.3)) or $\delta_i = \frac{\mathbf{y}_i^H M_i\mathbf{y}_i}{\mathbf{y}_i^H\mathbf{y}_i}$ (cf. (10.2)) we obtain Algorithms 10.2(sq0) and 10.2(rq0), respectively.

A proof of local quadratic convergence of these algorithms is given in [33] in the case where $\mathbf{u}_i = \mathbf{y}_{i-1}$ for all i.

The algorithms can be readily extended to the case where the values λ_i approximate an eigenvalue λ having geometric and algebraic multiplicity $\nu > 1$ (see [12, Section 7.1.4] on the definition of multiplicity) . In this case one should use rank-ν modifications $K_i = M_i + UV^H$ where U and V are $n \times \nu$ matrices and should modify the RQ/SMW and SQ/SMW iterations based on the following equations,

$$Z_i = K_i^{-1}(I_\nu + U_iG_i^{-1}V_i^HK_i^{-1})Y_i, \quad K_i = M_i + U_iV_i^H, G_i = I_\nu - V_i^HK_i^{-1}U_i,$$

$$\delta_i = \frac{\mathbf{e}_\alpha^HZ_i^HY_i\mathbf{e}_\alpha}{\mathbf{e}_\alpha Z_i^HZ_i\mathbf{e}_\alpha} \quad \text{or} \quad \delta_i = \frac{\mathbf{e}_g^TY_i\mathbf{e}_h}{\mathbf{e}_g^TZ_i\mathbf{e}_h}, \quad \mathbf{e}_g^TZ_i\mathbf{e}_h \neq 0,$$

$$\|Y_i\| \leqslant t\|Z_i\|,$$

$$Y_{i+1} = Z_i/s_i, \quad M_{i+1} = M_i - \delta_i I, \quad \lambda_i = \mathbf{e}_1^T(M_i - M)\mathbf{e}_1.$$

Likewise one should modify the PRQ and PSQ iterations, by employing in particular the following equations,

$$K_iZ_i = U_i, \quad G_i = I_\nu - V_i^HZ_i,$$

$$\delta_i = \frac{\mathbf{e}_\alpha^HZ_i^HU_iG_i\mathbf{e}_\alpha}{\mathbf{e}_\alpha Z_i^HZ_i\mathbf{e}_\alpha} \quad \text{or} \quad \delta_i = \frac{\mathbf{e}_g^TU_iG_i\mathbf{e}_h}{\mathbf{e}_g^TZ_i\mathbf{e}_h}, \quad \mathbf{e}_g^TZ_i\mathbf{e}_h \neq 0,$$

$$\|U_iG_i\| \leqslant t\|Z_i\|.$$

Remark 10.1. We can extend all eigen-solvers in this section to the approximation of the eigenspaces associated with a fixed set of eigenvalues $\Lambda = \{\lambda^{(1)}, \ldots, \lambda^{(k)}\}$. We should just redefine the matrices M_i as $\prod_{j=1}^k(M - \lambda_i^{(j)}I)$ where $\lambda_i^{(j)}$ denote the current approximations to the eigenvalue $\lambda^{(j)}$ for $j = 1, \ldots, k$ and $i = 0, 1, \ldots$, and we should update these approximations and matrices by applying the Rayleigh–Ritz process [36,2]. For $k = 1$ we come back to the algorithms of this section.

10.3. Newton's linearization with additive preprocessing

Theorem 10.1. *Suppose $\lambda + \delta\lambda$ is an eigenvalue having geometric multiplicity ν for an $n \times n$ matrix M, whereas U and V are $n \times \nu$ matrices. Write $\delta = |\delta\lambda|, M(\mu) = M - \mu I, K(\mu) = M(\mu) + UV^H$ for $\mu = \lambda$ and $\mu = \lambda + \delta\lambda$,*

$$X^H = V^H K^{-1}(\lambda), \quad (X + \delta X)^H = V^H K^{-1}(\lambda + \delta\lambda),$$
$$Y = K^{-1}(\lambda)U, Y + \delta Y = K^{-1}(\lambda + \delta\lambda)U,$$
$$F = X^H Y = V^H K^{-2}(\lambda)U,$$
$$G = I_\nu - X^H U = I_\nu - V^H Y = I_\nu - V^H K^{-1}(\lambda)U.$$

Suppose $\delta \to 0$ and the matrices $K(\lambda)$ and $K(\lambda + \delta\lambda)$ are nonsingular. Then

(a) *$\{\lambda + \delta\lambda, X + \delta X, Y + \delta Y\}$ is an eigentriple made up of an eigenvalue $\lambda + \delta\lambda$ of the matrix M and the matrix bases $X + \delta X$ and $Y + \delta Y$ for its associated left and right eigenspaces,*
(b) $\delta Y = (\delta\lambda)K^{-1}(\lambda)(I - (\delta\lambda)K^{-1}(\lambda))^{-1}Y = (\delta\lambda)K^{-1}(\lambda)Y + O(\delta^2)$,
(c) $\delta X = (\delta\lambda)K^{-H}(\lambda)(I - (\delta\lambda)K^{-H}(\lambda))^{-1}X = (\delta\lambda)K^{-H}(\lambda)X + O(\delta^2)$,
(d) $(\delta\lambda)UF = M(\lambda)Y + O(\delta^2)$,
(e) $(\delta\lambda)VF^H = M(\lambda)^H X + O(\delta^2)$, *and*
(f) *if at least one of the matrices U and V has full column rank, then* $(\delta\lambda)F = G + O(\delta^2)$.

Proof. Part (a) follows from Theorem 5.1.

To prove part (b), combine the matrix equations $K(\lambda + \delta\lambda)(Y + \delta Y) = U$ (implied by Theorem 5.1), $K(\lambda)Y = U$, and $K(\lambda + \delta\lambda) = K(\lambda) - (\delta\lambda)I$ (implied by the definitions of the matrices Y and $K(\mu)$). Obtain that $Y + \delta Y = K^{-1}(\lambda)U + (\delta\lambda)K^{-1}(\lambda)(Y + \delta Y)$. Recall that $Y = K^{-1}(\lambda)U$ and obtain that $\delta Y = (\delta\lambda)K^{-1}(\lambda)(Y + \delta Y)$ and consequently $\delta Y = (\delta\lambda)K^{-1}(\lambda)(I - (\delta\lambda)K^{-1}(\lambda))^{-1}Y$.

Part (c) is proved similarly.

Next recall that $M(\lambda + \delta\lambda)(Y + \delta Y) = (M(\lambda) - (\delta\lambda)I)(Y + \delta Y) = 0$. Therefore $M(\lambda)(Y + \delta Y) = (\delta\lambda)(Y + \delta Y) = (\delta\lambda)Y + O(\delta^2)$, and so $(\delta\lambda)Y = M(\lambda)Y + M(\lambda)\delta Y + O(\delta^2)$. Substitute the expression for δY from part (b) and obtain that $(\delta\lambda)Y = M(\lambda)Y + (\delta\lambda)M(\lambda)K^{-1}(\lambda)Y + O(\delta^2)$.

Recall that $M(\lambda) = K(\lambda) - UV^H$ and obtain that $M(\lambda)K^{-1}(\lambda) = I_n - UV^H K^{-1}(\lambda)$. Substitute this expression and deduce that $(\delta\lambda)UV^H K^{-1}(\lambda)Y = M(\lambda)Y + O(\delta^2)$. This implies part (d) because $V^H K^{-1}(\lambda)Y = X^H Y = F$.

Part (e) is proved similarly.

Recall that $M(\lambda)Y = K(\lambda)Y - UV^H Y = U - UV^H Y = U(I_\nu - V^H Y) = UG$. Substitute the matrix equation $M(\lambda)Y = UG$ into the equation of part (d) and obtain that $(\delta\lambda)UF = UG + O(\delta^2)$, which implies part (f) where the matrix U has full column rank. Similarly deduce from part (e) that $(\delta\lambda)VF^H = VG^H + O(\delta^2)$. This implies part (f) where the matrix V has full column rank. □

Remark 10.2. We can expect that the matrix $K(\lambda)$ is well conditioned, and then part (f) of Theorem 10.1 implies that the matrix G has a small norm where $\tilde{\lambda} \approx \lambda$. If so, the computation of this matrix can lead to numerical stability problems because $\|I_\nu\| = 1$. We can still perform the computations with the standard IEEE double precision if we apply the advanced fast and accurate algorithms for sums and products (cf. [8,14,18]) and the extended iterative refinement in [23].

Here is our algorithm that relies on Theorem 10.1.

Algorithm 10.3. Newton's eigen-solving with additive preprocessing.

INPUT: an $n \times n$ matrix M, an approximation λ_0 to its eigenvalue having algebraic and geometric multiplicity ν, a positive integer N, and a tolerance t.

OUTPUT: either FAILURE or an approximation $\{\lambda, X, Y\}$ to an eigentriple of the matrix M such that $\|X^H M - \lambda X^H\| \leqslant t\|X\|$, $\|MY - \lambda Y\| \leqslant t\|Y\|$ (cf. Remark 10.3).

INITIALIZATION: Set $i \leftarrow 0, k \leftarrow 0$, and $M_0 \leftarrow M - \lambda_0 I$. Fix a moderately large positive scalar γ.
COMPUTATIONS:

1. If $k \geqslant N$, output FAILURE and stop. Otherwise generate a pair of $n \times \nu$ random matrices U_i and V_i scaled so that $\frac{1}{\gamma} < \frac{\|U_i V_i^H\|}{\|M_i\|} < \gamma$. Compute the matrix $K_i = M_i + U_i V_i^H$. If it is singular or ill conditioned, set $k \leftarrow k + 1$ and reapply Stage 1.
2. Otherwise compute the matrices $X_i^H = V_i^H K_i^{-1}$, $Y_i = K_i^{-1} U_i$, $F_i = X_i^H Y_i$, and $G_i = I_\nu - X_i^H U_i$. Compute the value $\lambda_i = \mathbf{e}_1^T (M_i - M)\mathbf{e}_1$, output the triple $\{\lambda, X, Y\} = \{\lambda_i, X_i, Y_i\}$ and stop if
$$\|X_i^H M_i\| \leqslant t\|X_i\|, \|M_i Y_i\| \leqslant t\|Y_i\|. \tag{10.8}$$
3. Otherwise select a pair of integers α and β such that $1 \leqslant \alpha \leqslant \nu, 1 \leqslant \beta \leqslant \nu, \mathbf{e}_\alpha^T F_i \mathbf{e}_\beta \neq 0$ (if there exists no such a pair of integers, output FAILURE and stop). Compute the ratio $\delta_i = \frac{\mathbf{e}_\alpha^T G_i \mathbf{e}_\beta}{\mathbf{e}_\alpha^T F_i \mathbf{e}_\beta}$. Compute the matrix $M_{i+1} = M_i - \delta_i I$, set $i \leftarrow i + 1$, and reapply Stage 2.

Theorem 10.1 implies correctness and local quadratic convergence of Algorithm 10.3.

Remark 10.3. We can apply the stopping criteria $\|G_i V_i^H\| \leqslant t\|X_i\|$, $\|U_i G_i\| \leqslant t\|Y_i\|$ instead of (10.8). Let us show equivalence. We have $X_i^H M_i = V_i^H K_i^{-1} M_i = V_i^H (I_n - K_i^{-1} U_i V_i^H) = V_i^H - V_i^H K_i^{-1} U_i V_i^H = G_i V_i^H$ and likewise $M_i Y_i = M_i K_i^{-1} U_i = (I_n - U_i V_i^H K_i^{-1}) U_i = U_i - U_i V_i^H K_i^{-1} U_i = U_i G_i$. We can save some flops by checking only one of the two inequalities in (10.8) or above and by skipping the test where $|\delta_i| > \theta t$ for a tolerance θ. For $\nu = 1$ the matrix G_i turns into a scalar g_i, the matrices U_i, V_i, X_i, and Y_i turn into vectors $\mathbf{u}_i, \mathbf{v}_i, \mathbf{x}_i$, and $\mathbf{y}_i$, respectively, and stopping criteria (10.8) into the bound $|g_i| \leqslant t\mu_i$, $\mu_i = \min\left\{\frac{\|\mathbf{x}_i\|}{\|\mathbf{v}_i\|}, \frac{\|\mathbf{y}_i\|}{\|\mathbf{u}_i\|}\right\}$.

Remark 10.4. Unless the norm $\|\delta_i K_i^{-1}\|$ is small enough, convergence and numerical stability of Algorithm 10.3 can be endangered where the matrices F_i have small norms. Assume for simplicity that λ is a simple eigenvalue, so that $\nu = 1$ and let a triple $\{\lambda_i, \mathbf{x}_i, \mathbf{y}_i\}$ approximate the eigentriple $\{\lambda, \mathbf{x}, \mathbf{y}\}$. Then $\mathbf{x}^H \mathbf{y} = 1$ and the matrices F_i turn into scalars $f_i = \mathbf{x}_i^H \mathbf{y}_i$. Suppose the coordinates $u_i^{(j)}$ of the vector $\mathbf{u}_i = (u_i^{(j)})_{j=1}^n$ are random variables independent of each other and uniformly distributed in the range $[-1, 1)$ or in the circle $\{\|u_i^{(j)}\| \leqslant 1\}$. Then one can estimate that the random scalar function $\mathbf{z}_i^H \mathbf{u}_i$ is expected to converge to zero as $n \to \infty$. The matrix K_i and therefore the vector $\mathbf{v}_i^H K_i^{-2}$ depend on the vector $\mathbf{u}_i$, but rather weakly, and in our tests the scalars f_i tended to nearly vanish already for moderately large dimensions n such as 128 and 256, thus making Stage 3 of Algorithm 10.3 prone to numerical stabilty problems. Moreover this stage relies on the estimates in part (d) of Theorem 10.1, but they are meaningful only where $\delta_i = o(f_i)$. If, however, $M = M^H$ is a Hermitian matrix, we choose $\mathbf{v}_i = \mathbf{u}_i$, so that $f_i = \mathbf{v}_i K_i^{-2} \mathbf{u}_i = \|K_i^{-1} \mathbf{u}_i\|^2 = \|\mathbf{y}_i\|^2$. In a heuristic extension of this recipe to the nonHermitian matrices M, we first choose $\mathbf{v}_i = \mathbf{u}_i$ and compute the vector $K_i^{-2} \mathbf{u}_i$ and the scalar $|\mathbf{u}_i^H \mathbf{z}_i|$. Then if this scalar is too small, we redefine the vector $\mathbf{v}_i$ by setting it equal to $K_i^{-2} \mathbf{u}_i$. We could have extended this process recursively, but in our tests never needed to do this.

The recipe in the following remark can be extended to all eigen-solvers in this section.

Remark 10.5. Given an approximation $\tilde{\lambda}$ to an eigenvalue λ, we can fix $\lambda_i = \tilde{\lambda}$ for all i and update the matrices U_i and V_i as follows, $U_i = Y_{i-1}$ and $V_i = X_{i-1}$ for all i. Theorem 10.1 implies that the linear spaces range(X_i) and range(Y_i) converge to the left and right eigenspaces associated with the eigenvalue λ. The convergence is linear, and for $\tilde{\lambda} \approx \lambda$ the overhead constants are small. Having $\lambda_i = \tilde{\lambda}$ for all i and a small integer ν, we can readily obtain the matrices K_i from K_{i-1} via the SMW formula and extend this iteration to the approximation of the eigenspace associated with a fixed cluster of eigenvalues.

10.4. Modifications of the inverse iterations with additive preprocessing

Here are some natural modifications of the algorithms in the two previous subsections.

1. The cost of performing Algorithms 10.2 and 10.3 is dominated at the stage of solving linear systems with the matrices K_i. This stage, however, can be simplified where the norm $\|\delta_{i-1}K_i\| = |\delta_{i-1}|\|K_i\|$ is small because $K_i = K_{i-1} - \delta_{i-1}I = (I - \delta_{i-1}K_{i-1}^{-1})K_{i-1}$ and so $K_i^{-1} = \sum_{j=0}^{\infty}\delta_{i-1}^j K_{i-1}^{-1-j}$. Instead of a linear system with the matrix K_i we can solve two systems, either one with the matrix K_{i-1} and another with the strongly diagonally dominant matrix $I - \delta_{i-1}K_{i-1}^{-1}$ or both systems with the matrix K_{i-1} provided $K_i^{-1} \approx K_{i-1}^{-1} + \delta_{i-1}K_{i-1}^{-2}$.
2. Suppose $U = U_i$ and $V = V_i$ for all i and modify Algorithm 10.3 as follows. Recall that $K_i = K_{i-1} - \delta_{i-1}I = K_{i-1}(I - \delta_{i-1}K_{i-1}^{-1})$ and obtain (ignoring the terms in $O(|\delta_{i-1}|^3)$) that $K_i^{-1} = K_{i-1}^{-1}(I - \delta_{i-1}K_{i-1}^{-1})^{-1} = K_{i-1}^{-1}(I + \delta_{i-1}K_{i-1}^{-1} + \delta_{i-1}^2K_{i-1}^{-2})$, $K_i^{-2} = K_{i-1}^{-2}(I + 2\delta_{i-1}K_{i-1}^{-1} + 3\delta_{i-1}^2K_{i-1}^{-2})$. Now write $G_{i,j} = V^HK_i^{-j}U$, so that $F_i = G_{i,2}$ and $G_i = I_\nu - G_{i,1}$. Keep ignoring the terms in $O(|\delta_{i-1}|^3)$ and deduce that $F_i = G_{i-1,2} + 2\delta_{i-1}G_{i-1,3} + 3\delta_{i-1}^2G_{i-1,4}$, $G_i = G_{i-1} - \delta_{i-1}F_{i-1} - \delta_{i-1}^2G_{i-1,3}$. Suppose $\mathbf{e}_\alpha^TG_{i,h}\mathbf{e}_\beta \neq 0$ for $h = 2$ and $h = 3$, $\delta_{i-1} = \frac{\mathbf{e}_\alpha^TG_{i-1}\mathbf{e}_\beta}{\mathbf{e}_\alpha^TF_{i-1}\mathbf{e}_\beta}$, and $\delta_i = \frac{\mathbf{e}_\alpha^TG_i\mathbf{e}_\beta}{\mathbf{e}_\alpha^TF_i\mathbf{e}_\beta}$. Then $\mathbf{e}_\alpha^TG_i\mathbf{e}_\beta = -\delta_{i-1}^2\mathbf{e}_\alpha^TG_{i-1,3}\mathbf{e}_\beta$, whereas $F_i = F_{i-1} + O(|\delta_{i-1}|)$. Therefore

$$\delta_i = -\delta_{i-1}^2\frac{\mathbf{e}_\alpha^TG_{i-1,3}\mathbf{e}_\beta}{\mathbf{e}_\alpha^TF_{i-1}\mathbf{e}_\beta}. \tag{10.9}$$

   Now assume the value $\mathbf{e}_\alpha^TF_{i-1}\mathbf{e}_\beta = \mathbf{e}_\alpha^TF_i\mathbf{e}_\beta + O(|\delta_{i-1}|)$ and the vector $\mathbf{u}_{i-1,\beta} = K_{i-1}^{-2}U\mathbf{e}_\beta$ available. Then we can readily compute the vector $\bar{\mathbf{u}}_{i-1,\beta} = G_{i-1,3}\mathbf{e}_\beta = K_{i-1}^{-1}\mathbf{u}_{i-1,\beta}$ and the values $\mathbf{e}_\alpha^TG_{i-1,3}\mathbf{e}_\beta = \mathbf{e}_\alpha^T\bar{\mathbf{u}}_{i-1,\beta}$ and δ_i in (10.9). We use these expressions for computing the values δ_i at Stage 3 of Algorithm 10.3 where i is even, that is $i = 1, 3, 5, \ldots$, and keep the original expressions for δ_i in Algorithm 10.3 where i is odd, that is, $i = 0, 2, 4, \ldots$. Then at stages where i is even, we compute the vectors $K_{i-1}^{-1}\bar{\mathbf{u}}_{i-1,\beta}$ but avoid computing the vectors $\mathbf{e}_\alpha^TV^HK_i^{-1}$ and $K_i^{-1}U\mathbf{e}_\beta$. We refer to the latter modification of Algorithm 10.3 as Algorithm 10.3a.
3. In Algorithms 10.2, 10.3, and 10.3a we modify the matrix M by adding matrices $U_iV_i^H$ of a fixed smaller rank. We can choose matrices U_i and V_i for which the solution of the linear systems $K_iY_i = U_i$ is simplified. Unless this slows down convergence, we yield overall simplification.

10.5. How can we initialize the inverse iteration and its extensions?

Generally, for the initialization of the iteration, one can employ the customary initialization policies for polynomial root-finding because eigen-solving for an $n \times n$ matrix M amounts to root-finding for its characteristic polynomial of degree n.

If we seek all n eigenvalues, we can begin with the initial approximate eigenvalues $\lambda_j^{(0)} = c + a\omega_{\tilde{n}}^j$ for $j = 0, 1, \ldots, \tilde{n} - 1$, the $\tilde{n}$th root of unity $\omega_{\tilde{n}} = \exp\left(\frac{2\pi}{\tilde{n}}\sqrt{-1}\right)$, $c = 0$ or $c = \gamma + \frac{1}{n}\text{trace}(M)$, a sufficiently large positive scalar a, a scalar γ reasonably close to the origin, and an integer $\tilde{n} \geqslant n$, say, $a \approx 2\|M\|$ and $\tilde{n} \approx 2n\log_2 n$. One can either choose $\tilde{n}$ distinct (possibly random) initial eigenvectors or reuse some of them.

Seeking a single eigenvalue (with possible extension to the other eigenvalues via deflation), one can initialize the iteration at one of these points, at $c_0 = \frac{1}{n}\text{trace}(M)$ (that is at the average of the eigenvalues), or at $c_0 + \gamma$.

Remark 10.6. In some cases an initial approximation is readily available. For example, seeking a basis for the ν-tail of a matrix M that has a positive numerical nullity ν, we can apply the iterations of this section to the matrix M^HM or MM^H initializing them at $\lambda_0 = 0$.

11. Root-finding for polynomial and secular equations

With a polynomial $p(x) = \sum_{i=0}^{n} p_i x^i = p_n \prod_{j=1}^{n}(x - \lambda_j)$, $p_n \neq 0$, one can associate the Frobenius companion matrix $F_{\mathbf{p}} = Z - \mathbf{p}\mathbf{e}_n^T = Z_1 - (\mathbf{p} + \mathbf{e}_1)\mathbf{e}_n^T$ where we write $\mathbf{p} = \left(\frac{p_i}{p_n}\right)_{i=0}^{n-1}$,

$$F_{\mathbf{p}} = \begin{pmatrix} 0 & & & & -\frac{p_0}{p_n} \\ 1 & \ddots & & & -\frac{p_1}{p_n} \\ & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & -\frac{p_{n-2}}{p_n} \\ & & & 1 & -\frac{p_{n-1}}{p_n} \end{pmatrix}, \quad Z_f = \begin{pmatrix} 0 & & & & f \\ 1 & \ddots & & & 0 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & 0 \\ & & & 1 & 0 \end{pmatrix}, \tag{11.1}$$

$Z = Z_0$ is the downshift matrix, Z_1 is the matrix of cyclic shift, $Z\mathbf{v} = (v_{i-1})_{i=0}^{n-1}$ and $Z_1\mathbf{v} = (v_{i-1 \bmod n})_{i=0}^{n-1}$ for $\mathbf{v} = (v_i)_{i=0}^{n-1}$ and $v_{-1} = 0$.

The roots of the polynomial $p(x)$ are precisely the eigenvalues of the matrix $F_{\mathbf{p}}$, but they are also precisely the eigenvalues of the generalized companion diagonal + rank-one matrix (hereafter we refer to it as a *DPR1* matrix),

$$C = C_{\mathbf{s},\mathbf{d}} = D_{\mathbf{s}} - \mathbf{u}\mathbf{v}^H \tag{11.2}$$

for $\mathbf{d} = (d_i)_{i=1}^{n}$, $\mathbf{s} = (s_i)_{i=1}^{n}$, $\mathbf{u} = (u_i)_{i=1}^{n}$, $\mathbf{v} = (v_i)_{i=1}^{n}$, n distinct values $s_1, \ldots, s_n$,

$$D_{\mathbf{s}} = \operatorname{diag}(s_i)_{i=1}^{n}, \tag{11.3}$$

$$d_i = u_i v_i = \frac{p(s_i)}{q_i(s_i)} \neq 0, \quad q_i(x) = \prod_{j \neq i}(x - s_i), \quad q_i(s_i) = q'(s_i), \ i = 1, \ldots, n, \tag{11.4}$$

$$q_i(s_i) = q'(s_i), \ i = 1, \ldots, n, \quad q(x) = \prod_{j=1}^{n}(x - s_i). \tag{11.5}$$

To define such a DPR1 matrix, one can choose any n-tuple of distinct scalars $s_1, \ldots, s_n$ (possibly crude approximations to the roots) and any pair of vectors $\mathbf{u} = (u_i)_{i=1}^{n}$ and $\mathbf{v} = (v_i)_{i=1}^{n}$ such that $u_i v_i = -p(s_i)/q'(s_i)$. Note that $C - \mu I$ is also a DPR1 matrix and that, unlike the Frobenius companion matrices, DPR1 matrices are defined by the values of the associated polynomial on a fixed set of points rather than by the coefficients. We recall the following result.

Theorem 11.1 (cf., e.g., [4, Theorem 4.4]). *The eigenvalues of the matrix C in (11.2) coincide with the roots of the associated secular equation (see [10,16] on its earlier study)*

$$\sum_{i=1}^{n} \frac{u_i v_i}{s_i - \lambda} = 1. \tag{11.6}$$

Theorem 11.2. *Suppose we are given $3n$ scalars u_i, v_i, and s_i, $i = 1, \ldots, n$, that define a DPR1 generalized companion matrix C in Eq. (11.2) and suppose we seek similar representation of the three following DPR1 generalized companion matrices,*

(a) *$C - \mu I$ for a fixed scalar μ,*
(b) *C^{-1} and*
(c) *C_{rev} associated with the polynomial $p_{\mathrm{rev}}(x)$.*

Write $s = 1 - \sum_{i=1}^{n} \frac{u_i v_i}{s_i}$ and suppose $s \neq 0$. (For $s = 0$ Eq. (11.6) has the root $\lambda = 0$.) Then we can compute the respective $3n$-tuples of parameters $u_i^{(\mathrm{new})}$, $v_i^{(\mathrm{new})}$, and $s_i^{(\mathrm{new})}$, $i = 1, \ldots, n$, by using (a) n flops, (b) $6n$ flops, and (c) $4n + 1$ flops, respectively.

Proof

(a) Define a DPR1 matrix $C - \mu I$ by reusing all the parameters $u_i = u_i^{(\text{new})}$ and $v_i = v_i^{(\text{new})}$ and recomputing only the values $s_i^{(\text{new})} = s_i - \mu$.

(b) Compute the matrix C^{-1} by applying the SMW formula $C^{-1} = (D - \mathbf{u}\mathbf{v}^H)^{-1} = D^{-1} + g^{-1}D^{-1}\mathbf{u}\mathbf{v}^H D^{-1} = D_- + \mathbf{u}_-\mathbf{v}_-^H$. The computation of the matrix $D_- = D^{-1}$ and the vectors $\mathbf{w} = D^{-1}\mathbf{u}$, $g = 1 - \mathbf{v}^H\mathbf{w}$, $\mathbf{u}_- = g\mathbf{w}$, and $\mathbf{v}_-^H = \mathbf{v}^H D^{-1}$ involves n, n, $2n$, n, and n flops, respectively.

(c) To define a DPR1 matrix C_{rev}, we seek $3n$ parameters $u_i^{(\text{new})}$, $v_i^{(\text{new})}$, and $s_i^{(\text{new})}$, $i = 1, \dots, n$, such that

$$\sum_{i=1}^{n} \frac{d_i^{(\text{new})}}{s_i^{(\text{new})} - (1/\lambda)} = 1 \tag{11.7}$$

for $d_i^{(\text{new})} = u_i^{(\text{new})} v_i^{(\text{new})}$ and for all values λ satisfying Eq. (11.6). First rewrite Eq. (11.7) as $\sum_{i=1}^{n} \frac{d_i^{(\text{new})}\lambda}{s_i^{(\text{new})}\lambda - 1} = 1$. Then substitute the expressions $\frac{d_i^{(\text{new})}\lambda}{s_i^{(\text{new})}\lambda - 1} = \frac{d_i^{(\text{new})}}{s_i^{(\text{new})}}\left(1 + \frac{1}{s_i^{(\text{new})}\lambda - 1}\right)$ for $i = 1, \dots, n$ and obtain that Eq. (11.7) is equivalent to the equation $\sum_{i=1}^{n} \frac{d_i^{(\text{new})}}{s_i^{(\text{new})}} \frac{1}{s_i^{(\text{new})}\lambda - 1} = s^{(\text{new})}$ for $s^{(\text{new})} = 1 - \sum_{i=1}^{n} \frac{d_i^{(\text{new})}}{s_i^{(\text{new})}}$. Now write $s_i^{(\text{new})} = 1/s_i$, $d_i^{(\text{new})} = -s^{(\text{new})} d_i / s_i^2$ for $i = 1, \dots, n$ and observe that under this assignment we have $s^{(\text{new})} = 1/s$ and Eqs. (11.6) and (11.7) are equivalent to one another. It remains to compute $s_i^{(\text{new})} = 1/s_i$ (in n flops), $w_i = d_i/s_i$ (in n flops), $u_i^{(\text{new})} = w_i/s_i$ (in n flops) for $i = 1, \dots, n$, $-s = \sum_{i=1}^{n} w_i - 1$ (in n flops), $v_i^{(\text{new})} = -1/s$ for $i = 1, \dots, n$. □

The transition $F_{\mathbf{p}} \Longrightarrow C$ (resp. $F_{\mathbf{p}} \Longleftarrow C$) for fixed knots $s_1, \dots, s_n$ essentially amounts to multipoint evaluation of (resp. interpolation to) the polynomial $p(x)$. Generally these operations require $O(n \log^2 n)$ high precision arithmetic operations, but the bound decreases to $O(n \log n)$ in the case of the knots $s_i = a\omega_n^i + b$, $i = 1, \dots, n$, where $\omega_n = \exp\left(\frac{2\pi}{n}\sqrt{-1}\right)$ and $a \neq 0$ and b are two constants (cf., e.g., [21, Problem 2.4.3]). The same cost bounds cover the computation of the coefficients of the auxiliary polynomials $q(x)$ and $q'(x)$ and the values $q'(s_1), \dots, q'(s_n)$. The latter operations can be viewed as preprocessing for they depend only on the knots $s_1, \dots, s_n$, and not on the polynomial $p(x)$. Moreover they can be skipped in the transition $F_{\mathbf{p}} \Longrightarrow C$ where $s_i = \omega_n^i$, $i = 1, \dots, n$, $q(x) = x^n - 1$ and $q'(x) = nx^{n-1}$. In this case $D = \Omega Z_1 \Omega^{-1}$ is a diagonal matrix [7], and since $Z_1 = F_{\mathbf{p}} + (\mathbf{p} + \mathbf{e}_1)\mathbf{e}_n^T$, it follows that $\Omega^{-1} F_{\mathbf{p}} \Omega = D - \mathbf{u}\mathbf{v}^H$ where $\Omega = (\omega_n^{ij})_{i,j=0}^{n-1}$ is the $n \times n$ matrix of the discrete Fourier transform, $\mathbf{u} = \Omega^{-1}(\mathbf{p} + \mathbf{e}_1)$, and $\mathbf{v}^H = \mathbf{e}_n^T \Omega$. These FFT-based computations are known to be norm-wise numerically stable (cf., e.g., [6, Section 3.4]).

The reduction to eigen-solving leads to some of the most effective polynomial root-finders. In particular such a root-finder in [4] turned out to be competitive with the Aberth's (Börsch–Supan's) algorithm, which is the basis of the current best package MPSOLVE in [3] for approximating all roots of a polynomial. Furthermore, the root-finder in [4] has the additional power of rapidly approximating just a single root or the roots in a fixed region, and is highly effective also for solving the secular equation in Theorem 11.1. Even a relatively minor acceleration of this algorithm can give it upper hand versus the Aberth's and make it the root-finder of choice.

Next we employ A-preprocessing to use fewer flops per an iteration loop in our algorithms, derive the respective estimates, and display them in Tables 1 and 2. (In our tests the algorithms in Section 10.2 with such simplified loops compute crude approximations to the eigenvalues as fast as the RQ and SQ loops do by with no preprocessing, but unlike the latter loops cannot refine these approximations. In contrast, Algorithms 10.3 and 10.3a with such simplified loops are more vulnerable to the problems in Remark 10.4 at the initial stages, but remain powerful for the refinement task.)

Table 1
Number of flops per an iteration loop in the algorithms applied to an $n \times n$ companion matrix (cf. Remark 11.1).

Algorithm	GE	Algorithm 10.2(sq)	Algorithm 10.2(sq0)	Algorithm 10.3
Flops	$7n-3$	$2n+3$	$2n+3$	$4n+1$

Table 2
Number of flops per an iteration loop in the algorithms applied to an $n \times n$ DPR1 matrix.

Algorithm	[4]	Algorithm 10.2(sq)	Algorithm 10.2(sq0)	Algorithm 10.3
Flops	$9n$	$3n+2$	$4n$	$5n$

First recall that the algorithms in [4] rely on application of the RQ and SQ iterations (10.1)–(10.4) to the Frobenius companion matrix $F_{\mathbf{p}}$ in (11.1) or the generalized companion matrix C in (11.2).

In our estimates for the cost of our computations with the matrix $F_{\mathbf{p}}$ we employ the following simple lemma.

Lemma 11.1. (a) *A nonsingular bidiagonal linear system of n equations* $B\mathbf{x} = \mathbf{f}$ *can be solved in* $2n-1$ *flops by means of the substitution algorithm.* (b) *The algorithm is numerically stable if the system is diagonally dominant, that is if* $2|b_{ii}| \geqslant \min\left\{\sum_i |b_{ij}|, \sum_j |b_{ij}|\right\}$ *for* $B = (b_{ij})_{i,j}$, *e.g., if* $B = aI + bZ$ *and* $|a| > |b|$.

At every iteration loop of the SQ and RQ iterations, the overall computational cost is dominated at the stage of the solution of a linear system of equations with a shifted matrix $M - \mu_i I$ for $M = F_{\mathbf{p}}$ or $M = C$ and a scalar μ_i. This takes $7n-6$ flops for $M = F_{\mathbf{p}}$ (based on Gaussian elimination) and $9n$ flops in [4] for $M = C$.

Preprocessing with $\mathbf{u}\mathbf{v}^H = \mathbf{p}\mathbf{e}_n^T$ enables acceleration. In particular we decrease the overall cost to $2n+3$ flops per the entire iteration loop in Algorithm 10.2(sq) in the case where $M = F_{\mathbf{p}}$. Indeed $F_{\mathbf{p}} + \mathbf{p}\mathbf{e}_n^T = Z$, so that $F_{\mathbf{p}} - \mu_i I + \mathbf{p}\mathbf{e}_n^T = Z - \mu_i I$ is a bidiagonal (Toeplitz) matrix, and we apply Lemma 11.1. Furthermore in this case we have $\mathbf{v} = \mathbf{e}_n$, so that $g_i = 1 - \mathbf{e}_n^H \mathbf{z}_i = 1 - z_i^{(n)}$. The respective PSQ $\delta_i = g_i \frac{u_i^{(j)}}{z_i^{(j)}}$ is computed in three flops, and we update the shift value μ_i and the matrix $F_{\mathbf{p}} - \mu_i I + \mathbf{p}\mathbf{e}_n^T = Z - \mu_i I$ in single flop.

Algorithm 10.2(sq0) performs as fast, in $2n+3$ flops, because it also updates δ_i in three flops.

$4n+1$ flops are sufficient in Algorithm 10.3 applied to the matrix $M = F_{\mathbf{p}}$ and slightly rearranged. Namely we use $4n-2$ flops for computing the vectors $\mathbf{y}_i = (Z - \mu_i I)^{-1}\mathbf{p}$ and $\tilde{\mathbf{y}}_i = (Z - \mu_i I)^{-1}\mathbf{y}_i$ (cf. Lemma 11.1). Then we obtain the values $\tilde{f}_i = \mathbf{e}_n^T \mathbf{y}_i$ and $f_i = \mathbf{e}_n^T \tilde{\mathbf{y}}_i$ (cost-free), $g_i = 1 - \tilde{f}_i$, and $\delta_i = \frac{g_i}{f_i}$, and update the value λ_i in three flops overall.

We apply preprocessing $F_{\mathbf{p}} - \mu_i I \to F_{\mathbf{p}} - \mu_i I + \mathbf{p}\mathbf{e}_n^T = Z - \mu_i I$ where $\mu_i \geqslant 1$ because in this case the matrix $Z - \mu I$ is well conditioned. Approximating the eigenvalues $\lambda < 1$, we should either work with the reverse polynomial $x^n p(1/x) = \sum_{i=0}^n p_{n-i} x^i = p_0 \prod_{j=1}^n (x - 1/\lambda_j)$ (where w.l.o.g. we can assume that $p_0 \neq 0$) or apply preprocessing $F_{\mathbf{p}} - \mu_i I \to F_{\mathbf{p}} - \mu_i I + (\mathbf{p} + \mu_i \mathbf{e}_n + \mathbf{e}_1)\mathbf{e}_n^T = Z_1^T (I - \mu_i Z)$.

Remark 11.1. In all our algorithms above we can save n flops where we approximate the right eigenvector $(\lambda_h^{i-1})_{i=1}^n$ associated with a simple eigenvalue λ_h of the matrix $F_{\mathbf{p}}^T$, $h = 1, \ldots, n$, although in this case convergence can deteriorate.

We use $3n+2$ flops in Algorithm 10.2(sq) applied to DPR1 matrix $M = C$. Indeed under preprocessing $C \to D_{\mathbf{s}} = C + \mathbf{u}\mathbf{v}^H$ we deal with the diagonal matrices $D_{\mathbf{s}}$ and $K_i = D_{\mathbf{s}} - \lambda_i I$ and update the matrix K_i in n flops. We compute the vector $\mathbf{y}_i = K_i^{-1}\mathbf{u}$ in Algorithm 10.2(sq) also in n flops. We choose $\mathbf{v} = \mathbf{e}$, $\mathbf{e} = (\pm 1)_{i=0}^{n-1}$, that is the vector filled with the values -1 and 1, and obtain the value

$g_i = 1 - \mathbf{v}^H\mathbf{y}_i$ in n flops; then in two flops we obtain the SPQ $\delta_i = g_i \frac{u_i^{(j)}}{z_i^{(j)}}$. Overall this sums to $3n + 2$ flops per an iteration loop, as we claimed.

We use $4n$ flops per a loop of Algorithm 10.2(sq0) applied to the DPR1 matrix $M = C$. In this case K_i is a diagonal matrix, $M_i = K_i + \mathbf{u}\mathbf{e}^T$, and so we only need n flops to update the matrices M_i and K_i, n flops to compute the vector $K_i^{-1}\mathbf{u}$, and $2n$ flops to compute the ratio δ_i.

Algorithm 10.3 applied to a DPR1 matrix uses n flops to update the diagonal matrix K_i, $2n$ flops for computing the vectors $\mathbf{y}_i = K_i^{-1}\mathbf{p}$ and $\tilde{\mathbf{y}}_i = K_i^{-1}\mathbf{y}_i$, followed by $2n - 2$ flops for obtaining the inner products $\tilde{f}_i = \mathbf{e}^T\mathbf{y}_i$ and $f_i = \mathbf{e}^T\tilde{\mathbf{y}}_i$ and two flops for computing the values $g_i = 1 - \tilde{f}_i$ and $\delta_i = g_i/f_i$. All this is summed to $5n$ flops per iteration loop.

In the case of both companion and DPR1 input matrices, Algorithm 10.3a requires a little more flops, converges a little slower and diverges a little more readily (cf. Table 18).

Remark 11.2. The algorithms in [19,20,22] support nearly optimal Boolean complexity bounds for the classical problem of root-finding for polynomial equation

$$p(x) = 0 \quad \text{for } p(x) = \sum_{i=0}^{n} p_i x^i, \ \ p_n \neq 0, \tag{11.8}$$

but the users prefer other algorithms that show excellent practical performance, although support no competitive estimates for the computational cost.

Remark 11.3. One can try to extend the powerful eigen-solving algorithms for DPR1 matrix to the case of general input matrices. For example, one can evaluate the characteristic polynomial $\det(M - xI)$ at the n points $x_i = \text{trace}(M) + a\omega_n^i$, $i = 0, 1, \ldots, n-1$ for a sufficiently large scalar a, e.g., $a = 2\|M - \text{trace}(M)\|$, and $\omega_n = \exp\left(\frac{2\pi}{n}\sqrt{-1}\right)$. Then a DPR1 matrix sharing the eigenvalues with the matrix M can be readily defined by Eqs. (11.2)–(11.5). Such an approach can be prone to numerical stability problems, but strong diagonal dominance of the matrices $M - (x_i - \lambda)I$ for all eigenvalues λ of the matrix M is encouraging.

12. Numerical experiments

We performed a series of numerical experiments in the Graduate Center of the City University of New York to test our algorithms of this paper. Tables 3–18 display the results of these tests.

Tables 3–12 represent the results of experimental computation of cas, nmbs and null vectors of general and Toeplitz matrices. These results demonstrate the power of the algorithms in Section 6 and are reproduced from [28,29]. The respective tests were conducted by the second author on a Dell server with a dual core 1.86 GHz Xeon processor and 2G memory running Windows Server 2003 R2. The test Fortran code was compiled with the GNU gfortran compiler within the Cygwin environment.

The other tests (supporting the results in Tables 13–18) were performed by the fourth and mostly the third authors on a Dell PC with a dual core 1.86 GHz and 2G memory. The test sofware was Matlab 7.5.0.

Table 3
CPU time (in cycles) for computing null vectors of Toeplitz matrices (cf. [29]).

Size	Rand. aug.	*QR*	SVD	*QR*/Rand. aug.	SVD/Rand. aug.
256	3.8	18.4	317.8	4.8	83.6
512	8.0	148.0	5242.1	18.5	655.3
1024	16.1	1534.2	87371.2	97.0	5522.6
2048	33.6	11750.3	—	357.7	—
4096	79.5	—	—	—	—
8192	169.5	—	—	—	—

Table 4
CPU time (in cycles) for computing null vectors of circulant matrices (cf. [28]).

Size	Rand. aug.	*QR*	SVD	*QR*/Rand. aug.	SVD/Rand. aug.
256	3.0	18.8	261.5	6.3	87.2
512	7.3	147.9	4220.9	20.3	578.2
1024	16.1	1538.3	70452.5	97.1	4445.8
2048	35.5	11748.3	—	342.1	—
4096	78.7	—	—	—	—
8192	170.4	—	—	—	—

Table 5
CPU time (in cycles) for computing null vectors of symmetric Toeplitz matrices (cf. [28]).

Size	Rand. aug.	*QR*	SVD	*QR*/Rand. aug.	SVD/Rand. aug.
256	4.7	18.0	291.5	3.8	62.0
512	6.9	148.9	4728.4	21.6	685.3
1024	15.7	1536.9	78653.3	98.6	5046.2
2048	35.3	11747.8	—	343.2	—
4096	79.4	—	—	—	—
8192	170.4	—	—	—	—

Table 6
Residual norms for 64×64 unstructured matrices (cf. [28]).

Class	Type	Min	Max	Mean	Std
1	n	9.6×10^{-16}	3.0×10^{-11}	6.6×10^{-14}	9.8×10^{-13}
1	s	8.7×10^{-16}	2.8×10^{-12}	2.1×10^{-14}	1.1×10^{-13}
2	n	3.8×10^{-15}	7.8×10^{-12}	1.0×10^{-13}	4.1×10^{-13}
2	s	3.8×10^{-15}	5.7×10^{-12}	9.7×10^{-14}	3.9×10^{-13}
3	n	1.1×10^{-13}	1.6×10^{-10}	8.5×10^{-12}	1.4×10^{-11}
3	s	1.2×10^{-14}	2.9×10^{-10}	1.6×10^{-12}	1.3×10^{-11}
4	n	9.7×10^{-14}	1.8×10^{-10}	8.9×10^{-12}	1.5×10^{-11}
4	s	1.4×10^{-14}	3.8×10^{-10}	2.0×10^{-12}	1.5×10^{-11}

Table 7
Residual norms for 128×128 unstructured matrices (cf. [28]).

Class	Type	Min	Max	Mean	Std
1	n	5.9×10^{-15}	1.2×10^{-11}	1.1×10^{-13}	5.7×10^{-13}
1	s	1.9×10^{-15}	8.1×10^{-12}	5.6×10^{-14}	3.6×10^{-13}
2	n	5.9×10^{-15}	7.5×10^{-11}	2.1×10^{-13}	2.4×10^{-12}
2	s	4.6×10^{-15}	8.0×10^{-12}	1.1×10^{-13}	4.5×10^{-13}
3	n	1.0×10^{-12}	2.4×10^{-10}	1.6×10^{-11}	1.7×10^{-11}
3	s	6.1×10^{-14}	3.0×10^{-10}	2.9×10^{-12}	1.3×10^{-11}
4	n	1.2×10^{-12}	2.4×10^{-10}	1.7×10^{-11}	1.8×10^{-11}
4	s	8.1×10^{-14}	2.9×10^{-10}	4.2×10^{-12}	1.5×10^{-11}

We generated random real numbers with the random_number intrinsic Fortran function assuming the uniform probability distribution over the range $[-1, 1) = \{x : -1 \leqslant x < 1\}$. To shift to the range $\{y : b \leqslant y \leqslant a + b\}$ for fixed real a and b, we applied the linear transform $x \to y = ax + b$.

Tables 3–5 display the CPU time averaged over 100 runs for each input size and measured in terms of the CPU cycles. They can be converted into seconds by dividing them by a constant CLOCKS_PER_SEC, which is 1000 on our platform. In the respective tests we computed *QR* factorizations and SVDs by applying the LAPACK procedures DGEQRF and DGESVD, respectively.

Table 8
Residual norms for 64 × 64 unstructured matrices (in computations with iterative refinement and extended precision) (cf. [28]).

Class	Type	Min	Max	Mean	Std
1	n	4.0×10^{-53}	5.2×10^{-49}	6.0×10^{-50}	1.6×10^{-49}
1	s	1.9×10^{-59}	6.3×10^{-47}	6.3×10^{-48}	2.0×10^{-47}
2	n	1.0×10^{-14}	1.5×10^{-13}	5.2×10^{-14}	4.6×10^{-14}
2	s	4.1×10^{-14}	3.5×10^{-12}	4.9×10^{-13}	1.0×10^{-12}
3	n	2.4×10^{-50}	8.9×10^{-43}	9.9×10^{-44}	3.0×10^{-43}
3	s	2.8×10^{-55}	3.0×10^{-43}	3.0×10^{-44}	9.4×10^{-44}
4	n	2.9×10^{-13}	1.6×10^{-12}	6.4×10^{-13}	4.0×10^{-13}
4	s	9.7×10^{-13}	9.4×10^{-11}	1.7×10^{-11}	2.9×10^{-11}

Table 9
Residual norms for 128 × 128 unstructured matrices (in computations with iterative refinement and extended precision) (cf. [28]).

Class	Type	Min	Max	Mean	Std
1	n	1.8×10^{-56}	2.3×10^{-45}	2.3×10^{-46}	7.3×10^{-46}
1	s	6.9×10^{-57}	3.9×10^{-44}	4.9×10^{-45}	1.4×10^{-44}
2	n	2.0×10^{-14}	4.2×10^{-12}	5.9×10^{-13}	1.3×10^{-12}
2	s	4.9×10^{-14}	1.8×10^{-11}	3.3×10^{-12}	6.4×10^{-12}
3	n	2.4×10^{-55}	7.9×10^{-49}	1.1×10^{-49}	2.5×10^{-49}
3	s	1.6×10^{-52}	3.9×10^{-47}	5.7×10^{-48}	1.4×10^{-47}
4	n	1.7×10^{-13}	2.0×10^{-11}	4.0×10^{-12}	6.3×10^{-12}
4	s	3.2×10^{-13}	1.3×10^{-11}	3.3×10^{-12}	4.6×10^{-12}

Table 10
Ratios $\frac{\text{cond}(M)}{\text{cond}(K)}$ (cf. [29]).

Matrix size	Min	Max	Mean	Std
64 × 64	3.29×10^{9}	1.65×10^{13}	2.49×10^{12}	2.60×10^{12}
128 × 128	8.27×10^{8}	2.56×10^{12}	5.51×10^{11}	6.44×10^{11}

Table 11
Relative residual norms in the solution tests with 64 × 64 inputs (cf. [29]).

Refinement	Mmin	Max	Mean	Std
2 iterations	7.89×10^{-48}	8.26×10^{-44}	1.40×10^{-45}	8.47×10^{-45}
No iteration	1.43×10^{-31}	7.30×10^{-28}	1.69×10^{-29}	9.12×10^{-29}

Table 12
Relative residual norms in the solution tests with 128 × 128 inputs (cf. [29]).

Refinement	Min	Max	Mean	Std
2 iterations	1.31×10^{-46}	1.37×10^{-43}	4.11×10^{-45}	1.67×10^{-44}
No iteration	8.57×10^{-31}	1.92×10^{-27}	5.12×10^{-29}	2.55×10^{-28}

Tables 6–18 display various other average data in the columns marked **mean** and also display minimums, maximums and standard deviations of the 1000 runs in the columns marked **min**, **max**, and **std**, respectively.

12.1. Solution of singular Toeplitz linear systems

We generated $n \times n$ unsymmetric Toeplitz, circulant and symmetric Toeplitz matrices of rank $n - 1$ and computed their null vectors based on our randomized augmentation, *QR* factorization, and SVD of the input matrices.

Table 13
ν-tails of the SVDs and approximation by a nearby matrix of a lower rank to an $n \times n$ matrix M having a positive numerical nullity nnul(M).

nnul(M)	cond(M) **or** r_i	n	Min	Max	Mean	Std
2	cond(M)	64	$3.17 \times 10^{+02}$	$9.13 \times 10^{+04}$	$7.28 \times 10^{+03}$	$1.43 \times 10^{+04}$
2	cond(M)	128	$1.37 \times 10^{+03}$	$3.18 \times 10^{+06}$	$7.42 \times 10^{+04}$	$3.54 \times 10^{+05}$
2	cond(M)	256	$3.20 \times 10^{+03}$	$8.38 \times 10^{+06}$	$2.65 \times 10^{+05}$	$1.04 \times 10^{+06}$
2	r_1	64	5.61×10^{-10}	2.01×10^{-08}	3.43×10^{-09}	4.23×10^{-09}
2	r_1	128	4.84×10^{-10}	5.81×10^{-07}	1.15×10^{-08}	5.81×10^{-08}
2	r_1	256	8.09×10^{-10}	4.22×10^{-07}	1.05×10^{-08}	4.22×10^{-08}
2	r_2	64	1.83×10^{-08}	9.17×10^{-07}	1.41×10^{-07}	1.94×10^{-07}
2	r_2	128	5.53×10^{-08}	3.89×10^{-05}	7.84×10^{-07}	3.89×10^{-06}
2	r_2	256	1.05×10^{-07}	6.87×10^{-05}	1.72×10^{-06}	6.91×10^{-06}
2	r_3	64	7.24×10^{-10}	3.86×10^{-08}	5.18×10^{-09}	6.92×10^{-09}
2	r_3	128	1.06×10^{-09}	6.32×10^{-07}	1.40×10^{-08}	6.32×10^{-08}
2	r_3	256	1.21×10^{-09}	6.05×10^{-07}	1.55×10^{-08}	6.08×10^{-08}
4	cond(M)	64	$9.82 \times 10^{+02}$	$4.33 \times 10^{+05}$	$2.54 \times 10^{+04}$	$6.59 \times 10^{+04}$
4	cond(M)	128	$1.69 \times 10^{+03}$	$3.93 \times 10^{+06}$	$1.54 \times 10^{+05}$	$5.14 \times 10^{+05}$
4	cond(M)	256	$7.87 \times 10^{+03}$	$5.49 \times 10^{+06}$	$2.33 \times 10^{+05}$	$6.40 \times 10^{+05}$
4	r_1	64	3.65×10^{-10}	2.59×10^{-07}	7.66×10^{-09}	2.82×10^{-08}
4	r_1	128	5.58×10^{-10}	6.31×10^{-07}	1.79×10^{-08}	7.49×10^{-08}
4	r_1	256	1.03×10^{-09}	3.30×10^{-07}	1.09×10^{-08}	3.41×10^{-08}
4	r_2	64	2.50×10^{-08}	1.14×10^{-05}	3.34×10^{-07}	1.19×10^{-07}
4	r_2	128	6.72×10^{-08}	3.61×10^{-05}	1.40×10^{-06}	4.78×10^{-06}
4	r_2	256	1.86×10^{-07}	3.11×10^{-05}	1.90×10^{-06}	3.79×10^{-06}
4	r_3	64	9.30×10^{-10}	3.84×10^{-07}	1.29×10^{-08}	4.14×10^{-08}
4	r_3	128	1.12×10^{-09}	7.90×10^{-07}	2.75×10^{-08}	9.92×10^{-08}
4	r_3	256	1.77×10^{-09}	3.42×10^{-07}	1.94×10^{-08}	4.17×10^{-08}
16	cond(M)	64	$1.96 \times 10^{+03}$	$1.61 \times 10^{+06}$	$9.41 \times 10^{+04}$	$2.32 \times 10^{+05}$
16	cond(M)	128	$7.60 \times 10^{+03}$	$9.90 \times 10^{+06}$	$4.72 \times 10^{+05}$	$1.59 \times 10^{+06}$
16	cond(M)	256	$1.97 \times 10^{+04}$	$1.80 \times 10^{+07}$	$9.20 \times 10^{+05}$	$2.65 \times 10^{+06}$
16	r_1	64	3.15×10^{-10}	1.23×10^{-07}	6.45×10^{-09}	1.64×10^{-08}
16	r_1	128	5.50×10^{-10}	5.23×10^{-07}	1.52×10^{-08}	6.44×10^{-08}
16	r_1	256	6.75×10^{-10}	1.91×10^{-07}	1.03×10^{-08}	2.50×10^{-08}
16	r_2	64	3.72×10^{-08}	9.37×10^{-06}	4.81×10^{-07}	1.24×10^{-06}
16	r_2	128	1.42×10^{-07}	4.25×10^{-05}	1.97×10^{-06}	6.21×10^{-06}
16	r_2	256	3.74×10^{-07}	8.02×10^{-05}	4.40×10^{-06}	9.81×10^{-06}
16	r_3	64	1.60×10^{-09}	6.19×10^{-07}	2.86×10^{-08}	8.38×10^{-08}
16	r_3	128	2.98×10^{-09}	1.87×10^{-06}	5.26×10^{-08}	2.10×10^{-07}
16	r_3	256	3.32×10^{-09}	5.78×10^{-07}	4.26×10^{-08}	8.54×10^{-08}

We use abbreviation "**Rand. aug.**", "**QR**", and "**SVD**" as pointers to the respective algorithms. Tables 3–5 cover our computation of null vectors for general Toeplitz, circulant, and symmetric Toeplitz input matrices, respectively. The tables show the CPU time of this computation for each of the three methods as well as the ratios of these data for the *QR*-based and SVD-based solutions versus the algorithm based on randomized augmentation. The ratios are displayed in the last two columns of the table.

In all our tests the computed approximate null vectors $\mathbf{y}$ had relative residual norms $\frac{\|M\mathbf{y}\|}{\|M\|\,\|\mathbf{y}\|}$ of the order of 10^{-17}.

The input size (dimension) 2^k ranged from 256 to 8192. The table entries are marked by a hyphen "-" where the tests required too long runtime and were not completed.

12.2. Generation of unstructured input matrices and additive preprocessors

For $n = 64$ and $n = 128$, we computed the $n \times n$ unstructured input matrices M numerically, with double precision, as the products $S\Sigma T^T$ (cf. [13, Section 28.3]). Here we generated random real

Table 14
Numbers of RQ and SQ iteration loops in Algorithms 10.1(rq) and (sq) until convergence.

Iteration	Matrix	n	Min	Max	Mean	Std
RQ	Frobenius	64	4.00	12.00	6.10	1.65
RQ	Frobenius	128	4.00	11.00	6.21	1.48
RQ	Frobenius	256	4.00	13.00	6.18	1.50
SQ	Frobenius	64	4.00	16.00	7.75	2.27
SQ	Frobenius	128	5.00	17.00	8.37	2.49
SQ	Frobenius	256	4.00	19.00	7.65	2.86
RQ	DPR1	64	5.00	12.00	7.67	1.61
RQ	DPR1	128	5.00	14.00	7.97	1.95
RQ	DPR1	256	5.00	14.00	7.88	1.69
SQ	DPR1	64	5.00	21.00	9.34	2.72
SQ	DPR1	128	5.00	21.00	9.80	2.94
SQ	DPR1	256	5.00	17.00	9.12	2.54
RQ	Random	64	3.00	3.00	3.00	0.00
RQ	Random	128	3.00	3.00	3.00	0.00
RQ	Random	256	3.00	3.00	3.00	0.00
SQ	Random	64	3.00	4.00	3.92	0.27
SQ	Random	128	3.00	4.00	3.78	0.42
SQ	Random	256	3.00	4.00	3.57	0.50

orthonormal matrices S and T, being the Q-factors in the QR factorization of matrices with random integer entries from the range $[-10^4, 10^4)$ and with positive diagonal entries of the R-factors. We defined diagonal matrices $\Sigma = \mathrm{diag}(\sigma_i)_{i=1}^n$ with the diagonal entries $\sigma_1, \dots, \sigma_1$ from one of the four following classes.

Class 1. $\sigma_i = \frac{1}{i}$ for $i = 1, \dots, n-k$, $\sigma_i = 0$ for $i > n-k$,

Class 2. $\sigma_i = \frac{1}{i}$ for $i = 1, \dots, n-k$, $\sigma_i = \frac{10^{-14}}{i-n+k}$ for $i > n-k$,

Class 3. $\sigma_i = \frac{1}{i}$ for $i = 1, \dots, n-k-l$, $\sigma_i = \frac{10^{-9}}{i-n+k+l}$ for $i = n-k-l+1, \dots, n-k$, $\sigma_i = 0$ for $i > n-k$,

Class 4. $\sigma_i = \frac{1}{i}$ for $i = 1, \dots, n-k-l$, $\sigma_i = \frac{10^{-9}}{i-n+k+l}$ for $i = n-k-l+1, \dots, n-k$, $\sigma_i = \frac{10^{-14}}{i-n+k}$ for $i > n-k$.

For each of these classes, besides generating random orthonormal matrices T independently of the matrices S, we defined T by setting $T = S$. Respectively we defined Classes 1n, 1s, 2n, 2s, 3n, 3s, 4n, and 4s where "n" stood for "nonsymmetric" and "s" for "symmetric".

In our tests we selected $k = 24$ and $l = 20$ for $n = 64$ and selected $k = 48$ and $l = 40$ for $n = 128$.

For every instance of the input matrix M we computed the A-modification matrix $K = M + UV^T$ for random orthonormal $n \times r$ generators U and for $V = U$ where $r = k$ for Classes 1 and 2 and $r = k + l$ for Classes 3 and 4.

12.3. Computation and approximation of complete annihilators with additive preprocessing

For each pair $\{n, r\}$, $n = 64$ and $n = 128$, we tested 1000 instances of the input matrices M, U and V defined in the previous subsection.

In these tests we computed approximate complete annihilators $K^{-1}U$ for Classes 1 and 2 and approximate complete annihilators $K^{-1}UX$ for $X = \mathrm{ca}(G)$ and $G = I_r - V^T K^{-1} U$ for Classes 3 and 4. In the latter case we successively computed the matrices $K^{-1}U$, $G = I_r - V^T K^{-1} U$ for $r = k + l$, an approximate complete annihilator X for the matrix G, and finally the approximate complete annihilator $K^{-1}UX = \mathrm{ca}(M)$.

In all cases we estimated the ratios $\frac{\|MK^{-1}U\|}{\|M\|\|K^{-1}U\|}$ and $\frac{\|MK^{-1}UX\|}{\|M\|\|K^{-1}UX\|}$, which are the relative residual norms for the matrices M in Classes 1 and 2 and in Classes 3 and 4, respectively. We output their

Table 15
Numbers of PRQ and PSQ iteration loops in Algorithm 10.2(rq0) and (sq0) until convergence.

Iteration	Matrix	n	Min	Max	Mean	Std
RPQ	Frobenius	64	5.00	13.00	8.52	1.48
RPQ	Frobenius	128	5.00	14.00	9.38	1.56
RPQ	Frobenius	256	7.00	14.00	10.24	1.36
SPQ	Frobenius	64	5.00	21.00	10.39	2.89
SPQ	Frobenius	128	4.00	18.00	11.40	3.00
SPQ	Frobenius	256	5.00	19.00	12.24	3.65
RPQ	DPR1	64	4.00	15.00	7.74	2.03
RPQ	DPR1	128	5.00	13.00	7.72	2.13
RPQ	DPR1	256	5.00	15.00	7.70	2.29
SPQ	DPR1	64	6.00	21.00	9.83	2.67
SPQ	DPR1	128	5.00	17.00	9.59	2.72
SPQ	DPR1	256	5.00	19.00	9.54	2.87
RPQ	Random	64	3.00	3.00	3.00	0.00
RPQ	Random	128	3.00	3.00	3.00	0.00
RPQ	Random	256	3.00	3.00	3.00	0.00
SPQ	Random	64	3.00	4.00	3.74	0.44
SPQ	Random	128	3.00	4.00	3.79	0.41
SPQ	Random	256	3.00	4.00	3.65	0.50

maximum, minimum, and average values as well as the standard deviations for each algorithm and each case. Tables 6 and 7 show the results of our tests performed with double precision and without using the iterative refinement.

We have also run 100 tests for each of $n = 64$ and $n = 128$ and for the input matrices M where we computed these matrices as the error-free products $M = S\Sigma T^T$ and applied the extended iterative refinement from [23] at the stage of computing the matrices $K^{-1}U$ and G^{-1}. Tables 8 and 9 display the results of these tests. As we expected, in the case of matrices M of Classes 2 and 4, the residual norms decrease only to the level of the smallest positive singular value σ_n, whereas in the case of matrices M of Classes 1 and 3 these norms immediately went below the level achieved with the costly SVD-based algorithms and then kept rapidly decreasing towards zero. (We stopped the iterative refinement process with the ratios at the levels well below 10^{-40}.)

12.4. Solution of unstructured nonhomogeneous linear systems via augmentation

(a) Generation of input matrices

We first fixed pairs of n and k for $n = 64, 128$ and $k = 7$. Then for every pair $\{n, k\}$ we generated $m = 100$ instances of matrices M, P_{01}, and $P_{10} = P_{01}^T$ and vectors $\mathbf{b}$ as follows.

The matrices M have been computed as the error-free products $S\Sigma T^H$ where S and T were $n \times n$ random unitary matrices (generated with double precision) and $\Sigma = \mathrm{diag}(\sigma_j)_{j=1}^n$, $\sigma_{n-j} = 10^{j-17}$ for $j = 1, \dots, k$, and $\sigma_{n-j} = 1/(n-j)$ for $j = k+1, \dots, n-1$ (cf. [13, Section 28.3]).

P_{01} was random $n \times k$ matrix with $\|P_{01}\| = \|M\|$.

For every choice of these matrices we performed preconditioning tests and the solution tests as follows.

(b) Preconditioning tests

We computed m ratios $\frac{\mathrm{cond}(M)}{\mathrm{cond}(K)}$ for $K = \begin{pmatrix} M & P_{01} \\ P_{01}^T & I_\nu \end{pmatrix}$.

Table 10 displays the average (mean), minimum, maximum, and standard deviation for the m ratios for $n = 64$ and $n = 128$.

(c) The solution tests

In the solution tests we solved nonsingular linear systems $M\mathbf{y} = \mathbf{b}$ where M was the matrix generated above, $\mathbf{b}$ was a random vector scaled so that $\|\mathbf{b}\| = \|M\| = 1$. We first computed the null vector

Table 16
Numbers of PRQ and PSQ iteration loops in Algorithm 10.2(rq) and (sq) until convergence.

Iteration	Matrix	n	Min	Max	Mean	Std	Diverged
RPQ	Frobenius	64	5.00	12.00	8.43	1.37	0
RPQ	Frobenius	128	5.00	39.00	9.78	3.45	0
RPQ	Frobenius	256	6.00	15.00	10.24	1.92	1
RSQ	Frobenius	64	4.00	19.00	10.79	3.34	0
RSQ	Frobenius	128	4.00	59.00	12.04	5.63	0
RSQ	Frobenius	256	4.00	21.00	11.68	3.79	1
RPQ	DPR1	64	4.00	14.00	7.95	2.28	0
RPQ	DPR1	128	5.00	15.00	7.53	1.9	0
RPQ	DPR1	256	4.00	14.00	8.42	2.14	0
RSQ	DPR1	64	5.00	21.00	9.44	3.26	0
RSQ	DPR1	128	5.00	20.00	9.33	2.95	0
RSQ	DPR1	256	5.00	20.00	9.71	3.08	0
RPQ	Random	64	3.00	3.00	3.00	0.00	0
RPQ	Random	128	3.00	3.00	3.00	0.00	0
RPQ	Random	256	3.00	3.00	3.00	0.00	0
RSQ	Random	64	3.00	4.00	3.9	0.30	0
RSQ	Random	128	3.00	4.00	3.8	0.40	0
RSQ	Random	256	3.00	4.00	3.58	0.50	0

Table 17
Number of iteration loops in Algorithm 10.3 until convergence.

Matrix	n	Min	Max	Mean	Std	Diverged
Frobenius	64	3.00	34.00	7.23	4.67	1
Frobenius	128	3.00	27.00	7.00	3.66	2
Frobenius	256	4.00	25.00	7.21	4.31	0
DPR1	64	4.00	47.00	10.48	6.48	1
DPR1	128	3.00	25.00	10.02	4.76	0
DPR1	256	5.00	25.00	10.19	4.88	0
Random	64	4.00	6.00	4.47	0.58	0
Random	128	3.00	6.00	4.48	0.56	0
Random	256	3.00	5.00	4.48	0.54	0
Hermitian	64	4.00	6.00	4.83	0.45	0
Hermitian	128	4.00	5.00	4.7	0.46	0
Hermitian	256	4.00	5.00	4.57	0.50	0

$\mathbf{z}$ of the matrix $(-\mathbf{b}, M)$, then scaled it to obtain the vector $(1, \mathbf{y})^H$, and finally output the solution vector $\mathbf{y}$.

Tables 11 and 12 display the average (mean), minimum, maximum, and standard deviation for the relative residual norms $\frac{\|M\mathbf{y}-\mathbf{b}\|}{\|\mathbf{y}\|}$ in our tests for $n = 64$ and $n = 128$, respectively. For each input instance we computed the solution in two ways, that is by performing two iteration loops of the extended iterative refinement and with no such iteration.

12.5. Approximation of the tails of the SVDs

We followed the recipes in Section 8 to compute approximations WX to the ν-tails T_ν of the SVDs of nearly rank deficient $n \times n$ input matrices M having numerical nullity ν for $n = 64, 128, 256$ and $\nu = 2, 4, 16$. For $W = K^{-1}U$ and $K = M + UV^H$ we let the matrices X minimize the norms $\|WX - T_\nu\|$, and we output the relative residual norms $r_1 = \frac{\|WX-T_\nu\|}{\|WX\|}$, $r_2 = \frac{\|MW\|}{\|M\|\|W\|}$, and $r_3 = \frac{\|MQQ^H\|}{\|M\|}$. Here $\|MQQ^H\|$ is the residual norm of the approximation to the matrix M by the rank-ν matrix $M(I - QQ^H)$ for $Q = Q(W)$.

Table 18
Number of iteration loops in Algorithm 10.3a until convergence.

Matrix	n	Min	Max	Mean	Std	Diverged
Frobenius	64	4.00	34.00	10.37	6.01	2
Frobenius	128	4.00	40.00	11.36	5.75	13
Frobenius	256	5.00	86.00	15.49	12.58	14
DPR1	64	4.00	30.00	12.99	5.49	2
DPR1	128	6.00	52.00	13.49	6.49	4
DPR1	256	4.00	44.00	13.14	7.11	2
Random	64	3.00	6.00	4.48	0.75	0
Random	128	3.00	6.00	4.44	0.54	0
Random	256	3.00	5.00	4.45	0.52	0

We defined the $n \times n$ input matrices M by their SVDs $M = S\Sigma T^T$ where we chose random unitary matrices S and T and a diagonal matrix $\Sigma = \mathrm{diag}(\sigma_j)_{j=1}^n$ such that $\sigma_j = 1/j, j = 1, \dots, n-\nu, \sigma_j = 10^{-10}, j = n-\nu+1, \dots, n$, and $\mathrm{cond}(M) = 10^{10}$. We generated $n \times \nu$ random matrices U and V and then scaled them to have the ratios $\|UV^H\|/\|M\|$ neither large nor small.

Table 13 displays the minimum, maximum and average values $\mathrm{cond}(K)$, r_1, r_2, and r_3 as well as the standard deviations in 100 runs of our tests.

12.6. Eigen-solving and root-finding tests

We counted the numbers of iteration loops until convergence in the RQ and SQ inverse iterations with and without additive preprocessing in Algorithms (a) 10.1(sq) and (rq), (b) 10.2(sq0) and (sq0), (c) 10.2(rq) and (sq), and (e) 10.3.

We applied these algorithms to (i) random general matrices, (ii) random Frobenius companion matrices, and (iii) random generalized companion DPR1 matrices, all of sizes $n \times n$ for $n = 64, 128, 256$. In some tests we used random complex values $x + y\sqrt{-1}$ defined by random parameters x and y from the real line interval $[-1, 1)$. We used additive preprocessors $\mathbf{u}_i\mathbf{v}_i^H = \mathbf{y}_{i-1}\mathbf{y}_{i-1}^H$, as in [33], except for Algorithms 10.3 and 10.3a, where we chose a random vector $\mathbf{u}$ and then set $\mathbf{u}_i = \mathbf{v}_i = \mathbf{u}$ for all i.

We initialized the iterations with the values λ_0 chosen at random on a large circle according to the recipes in Section 10.5. Tables 14–18 display the numbers of iteration loops until convergence in these runs.

We stopped the iterations, by applying the stopping criteria in (10.4), (10.8), and Remark 10.3 with the tolerance values $t = 10^{-6}$, $\tau = 10^{-6}$ and $\bar{\tau} = 10^{-6}$.

In each test run we allowed at most 100 iteration loops. If this bound has been exceeded, we stopped iteration. In our tests of Algorithms 10.1, 10.2, 10.3, and 10.3a we observed this never, at most in 1%, 2%, and 14% of the runs, respectively. We displayed the number of such cases (if they occurred) in the last column of the tables, marked as "diverged" and filled the rest of the tables based on the data from the other iterations. The bound of 100 loops was never exceeded in our tests with random matrices.

References

[1] R.R. Bitmead, B.D.O. Anderson, Asymptotically fast solution of Toeplitz and related systems of linear equations, Linear Algebra Appl. 34 (1980) 103–116.
[2] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, H. van der Vorst (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000.
[3] D.A. Bini, G. Fiorentino, Design, analysis, and implementation of a multiprecision polynomial rootfinder, Numer. Algorithms 23 (2000) 127–173.
[4] D.A. Bini, L. Gemignani, V.Y. Pan, Inverse power and Durand/Kerner Iteration for univariate polynomial root-finding, Comput. Math. (with Appl.) 47(2/3) (2004) 447–459 (also Technical Report TR 2002 020, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2002).
[5] D.A. Bini, L. Gemignani, V.Y. Pan, Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equation, Numer. Math. 3 (2005) 373–408 (also Technical Report 1470, *Department of Math., University of Pisa*, Pisa, Italy, July 2003).
[6] D. Bini, V.Y. Pan, Polynomial and Matrix Computations, Fundamental Algorithms, vol. 1, Birkhäuser, Boston, 1994.

[7] R.E. Cline, R.J. Plemmons, G. Worm, Generalized inverses of certain Toeplitz matrices, Linear Algebra Appl. 8 (1974) 25–33.
[8] J. Demmel, Y. Hida, Accurate and efficient floating point summation, SIAM J. Sci. Comput. 25 (2003) 1214–1248.
[9] R.A. Demillo, R.J. Lipton, A probabilistic remark on algebraic program testing, Inform. Process. Lett. 7 (4) (1978) 193–195.
[10] G.H. Golub, Some modified matrix eigenvalue problems, SIAM Rev. 15 (1973) 318–334.
[11] I. Gohberg, T. Kailath, V. Olshevsky, Fast Gaussian elimination with partial pivoting for matrices with displacement structure, Math. Comput. 64 (1995) 1557–1576.
[12] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., The Johns Hopkins University Press, Baltimore, Maryland, 1996.
[13] N.J. Higham, Accuracy and Stability in Numerical Analysis, second ed., SIAM, Philadelphia, 2002.
[14] X. Li, J. Demmel, D. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, M. Martin, B. Thompson, T. Tung, D. Yoo, Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. Math. Software 28 (2002) 152–205. Available at <http//crd.lbl.gov/∼xiaoye/XBLAS/>.
[15] M. Morf, Doubling Algorithms for Toeplitz and Related Equations, Proceedings of IEEE International Conference on ASSP, IEEE Press, Piscataway, New Jersey, 1980, pp. 954–959.
[16] A. Melman, A unifying convergence analysis of second-order methods for secular equations, Math. Comput. 66 (1997) 333–344.
[17] W.L. Miranker, V.Y. Pan, Methods of aggregations, Linear Algebra Appl. 29 (1980) 231–257.
[18] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput. 26 (6) (2005) 1955–1988.
[19] V.Y. Pan, Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros, in: Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM Press, New York, 1995, pp. 741–750.
[20] V.Y. Pan, Solving a polynomial equation: some history and recent progress, SIAM Rev. 39 (2) (1997) 187–220.
[21] V.Y. Pan, Structured Matrices and Polynomials: Unified Superfast Algorithms, Birkhäuser/Springer, Boston/New York, 2001.
[22] V.Y. Pan, Univariate polynomials: nearly optimal algorithms for factorization and rootfinding, J. Symbolic Comput. 33 (5) (2002) 701–733 (Proc. version in *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC 01)*, ACM Press, New York, 2001, pp. 253–267).
[23] V.Y. Pan, D. Grady, B. Murphy, G. Qian, R.E. Rosholt, A. Ruslanov, Schur aggregation for linear systems and determinants, Theoret. Comput. Sci., Special Issue on Symbolic–Numerical Algorithms, D.A. Bini, V.Y. Pan, and J. Verschelde (Eds.), 409 (2008) 255–268.
[24] V.Y. Pan, D. Ivolgin, B. Murphy, R.E. Rosholt, Y. Tang, X. Wang, X. Yan, Root-finding with eigen-solving, in: Dongming Wang, Li-Hong Zhi (Eds.), Symbolic-Numeric Computation, Birkhäuser, Basel/Boston, 2007, pp. 219–245.
[25] V.Y. Pan, D. Ivolgin, B. Murphy, R.E. Rosholt, Y. Tang, X. Yan, Additive preconditioning for matrix computations, Linear Algebra Appl. 432 (2010) 1070–1089 (Proc. version in Proceedings of the Third International Computer Science Symposium in Russia (CSR 2008), Lecture Notes in Computer Science (LNCS), vol. 5010, 2008, pp. 372–383).
[26] V.Y. Pan, M. Kunin, B. Murphy, R.E. Rosholt, Y. Tang, X. Yan, W. Cao, Linking the TPR1, DPR1 and arrow-head matrix structures, Comput. Math. Appl. 52 (10–11) (2006) 1603–1608.
[27] V.Y. Pan, B. Murphy, R.E. Rosholt, Y. Tang, X. Wang, A. Zheng, Eigen-solving via reduction to DPR1 matrices, Comput. Math. Appl. 56 (2008) 166–171.
[28] V.Y. Pan, G. Qian, Randomized preprocessing of homogeneous linear systems of equations, Linear Algebra Appl. 432 (2010) 3272–3318.
[29] V.Y. Pan, G. Qian, On Solving Linear System with Randomized Augmentation, Tech. Report TR 2009009, Ph.D. Program in Computer Science, Graduate Center, the City University of New York, 2009. Available at <http://www.cs.gc.cuny.edu/tr/techreport.php?id=352>.
[30] V.Y. Pan, G. Qian, A. Zheng, Randomized Preconditioning versus Pivoting, Tech. Report TR 2009010, Ph.D. Program in Computer Science, Graduate Center, the City University of New York, 2009. Available at <http://www.cs.gc.cuny.edu/tr/techreport.php?id=352>.
[31] G. Peters, J.H. Wilkinson, Inverse iteration, ill-conditioned equations and Newton's method, SIAM Rev. 21 (1979) 339–360.
[32] V.Y. Pan, X. Yan, Null space and eigenspace computations with additive preprocessing, in: Jan Verschelde and Stephen Watt (Eds.), Proceedings of the Third International Workshop on Symbolic–Numeric Computation (SNC 2007), July 2007, London, Ontario, Canada. ACM Press, New York, 2007, pp. 152–160.
[33] V.Y. Pan, X. Yan, Additive preconditioning, eigenspaces, and the inverse iteration, Linear Algebra Appl. 430 (2009) 186–203.
[34] J.T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, J. ACM 27 (4) (1980) 701–717.
[35] G.W. Stewart, Matrix Algorithms, Basic Decompositions, vol. I, SIAM, Philadelphia, 2001.
[36] G.W. Stewart, Matrix Algorithms, Eigensystems, vol. II, SIAM, Philadelphia, 1998.
[37] H. Unger, Nichtlinear behandlung von eigenwetaufgaben, ZAMM Z. Angew. Math. Mech. 30 (1950) 281–282.
[38] R.E. Zippel, Probabilistic algorithms for sparse polynomials, in: Proceedings of EUROSAM'79, Lecture Notes in Computer Science, vol. 72, Springer, Berlin, 1979, pp. 216–226.