Computers and Mathematics with Applications 61 (2011) 1305–1334

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

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

New progress in real and complex polynomial root-finding☆

Victor Y. Pan [a,b,*], Ai-Long Zheng [b]

[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

ARTICLE INFO

Article history:
Received 27 December 2010
Accepted 27 December 2010

Keywords:
Polynomial root-finding
Real roots
Companion matrices
DPR1 matrices
Eigenvalues
Eigenvectors
Rayleigh quotients
Secular equation
Homotopy continuation methods

ABSTRACT

Matrix methods are increasingly popular for polynomial root-finding. The idea is to approximate the roots as the eigenvalues of the companion or generalized companion matrix associated with an input polynomial. The algorithms also solve secular equation. QR algorithm is the most customary method for eigen-solving, but we explore the inverse Rayleigh quotient iteration instead, which turns out to be competitive with the most popular root-finders because of its excellence in exploiting matrix structure. To advance the iteration we preprocess the matrix and incorporate Newton's linearization, repeated squaring, homotopy continuation techniques, and some heuristics. The resulting algorithms accelerate the known numerical root-finders for univariate polynomial and secular equations, and are particularly well suited for the acceleration by using parallel processing. Furthermore, even on serial computers the acceleration is dramatic for numerical approximation of the real roots in the typical case where they are much less numerous than all complex roots.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

1.1. Background on root-finding

The solution of a univariate polynomial equation is the classical problem of mathematics and numerical mathematics, extensively studied for four millennia (since the Sumerian times) and is still a research area with highly important applications to numerical, algebraic and geometric computations (see, e.g., [1–8], and the bibliography therein).

The increasingly popular matrix methods approximate the roots as the eigenvalues of the associated companion and generalized companion matrices. Matlab's function "roots" applies the QR algorithm to companion matrices. The algorithms in [9,10] alternate the steps of Weierstrass' polynomial root-finding iteration (also called Durand–Kerner's) and of the QR algorithm applied to diagonal plus rank-one generalized companion matrices (hereafter we refer to them as *DPR1 matrices*) associated to polynomial and secular equations. (See our Theorem 7.1, the papers [11–13,8], and the bibliography therein on secular equation.)

Neither of these algorithms exploits the structure of input matrices, but nonetheless for the task of approximation of all n roots of a polynomial of degree n, the Fortune's package EIGENSOLVE [9] competes with the other current best root-finder MPSOLVE by Bini and Fiorentino [14], based on Börsch–Supan's iteration (also called Aberth's or Aberth–Ehrlich's).

☆ Supported by PSC CUNY Awards 61406-0039 and 62230-0040. Some results of this paper have been presented at the International Symposium on Symbolic and Algebraic Computation (ISSAC 2010) in Münhen, Germany, in July 2010.

* Corresponding author at: Department of Mathematics and Computer Science, Lehman College of the City University of New York, Bronx, NY 10468, USA. Tel.: +1 914 737 2637; fax: +1 718 960 8969.

E-mail addresses: v_y_pan@yahoo.com, victor.pan@lehman.cuny.edu (V.Y. Pan), azheng@yahoo.com (A.-L. Zheng).

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

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.12.070

Empirically these and various other celebrated iterative root-finders and eigen-solvers rapidly converge to the solution right from the start and with rare exceptions need a rather small constant number of iteration loops per root or eigenvalue. It follows that in practice one needs just the order of bn^2 bit operations (up to a polylog factor) to approximate all the n roots of an input polynomial of a degree n within the relative error bound 2^{-b} (cf. [3,15]).

This is a nearly optimal number of bit operations. Indeed for the worst case input one must process at least bn^2 input bits and therefore must perform at least $0.5bn^2$ bit operations to ensure the above bound on the output errors because in the worst case the input errors are magnified by the factor of n in the output. (Compare, e.g., the roots of the two polynomials $(x-4/7)^n-2^{-n}$ and $(x-4/7)^n-3^{-n}$.)

Such a magnification is not typical for random inputs, and one can decrease the computational cost on the average by tuning the precision of computing to each specific input and output. Such tuning has been incorporated in the MPSOLVE and EIGENSOLVE and can be included into most of the popular root-finders and eigen-solvers as well.

No adequate formal support has been provided so far for the empirical data on the fast convergence of the cited iterations, but this has not been a serious issue for the users, who gladly employ iterative algorithms as soon as their iteration loop is performed fast and their fast convergence has empirical support.

Nearly optimal (up to a polylog factor) upper bounds on the parallel and sequential Boolean and arithmetic time-complexity of the approximation of all roots of a polynomial have been proved based on the divide-and-conquer root-finder in [16,4,17]. These bounds, however, slightly exceed the empirical bounds supported by the other cited iterations, and since the users rely on empirical bounds, the implementation work for the algorithm in [16,4,17] has never had sufficient motivation.

1.2. Advancing the RQ iteration

We devise polynomial root-finders based on the inverse Rayleigh quotient iteration [15,18–20], which is a variant of Newton's iteration [21,22,19], but its power is enhanced because it approximates both eigenvalues and the associated eigenvectors. (Hereafter we use the abbreviation "RQ" for "Rayleigh quotient".) The iteration is a popular means for fast refinement of an approximate eigenvector and empirically has good global convergence to matrix eigenpairs.

It was first applied to polynomial root-finding in [23], then in [24,25]. In these applications the iteration exploited the input matrix structure, used linear arithmetic time $c_{\mathrm{RQ}}n$ per step (for a scalar c_{RQ}) and linear memory space, empirically converged in a few steps [15], and readily incorporated the techniques for tuning the precision of computing for each specific input and output. One can extend the iteration to approximating all eigenvalues via deflation, by applying the iteration concurrently at sufficiently many distinct initial points, or by combining these two techniques. Even for the task of the approximation of all roots the iteration competes with the Börsch–Supan and Weierstrass algorithms according to the tests in [26,23], although its strength is in approximating a single root and all roots in a fixed region. Under appropriate implementation it should become the method of choice for these tasks, and with some further advance can become such also for approximating all roots.

The QR algorithm in [11] also exploits the input matrix structure, and for companion and DPR1 matrices uses linear memory space and only $c_{\mathrm{QRR}}n$ arithmetic operations per step provided that the associated polynomial has only real roots [8]. The papers [27,28] remove this restriction and still support a linear time bound $c_{\mathrm{QR}}n$, but the constant c_{QR} noticeably exceeds c_{RQ} and c_{QRR}. Furthermore unlike the RQ iteration, the QR algorithm in the papers [11,27,28] has numerical problems in exploiting matrix structure and also restricts concurrency in the approximation of distinct eigenvalues.

In the present paper we pushed the advantages of the RQ approach further by incorporating our novel techniques, which simplify every iteration step and avoid or minimize application of deflation techniques for the approximation of all roots. Even under the sequential model of computing we yield noticeable progress versus the algorithms in [23] (see our tables in Sections 5 and 6). Furthermore our preprocessing turns the shifted companion matrices into bidiagonal matrices and turns the DPR1 matrices into diagonal ones, which allows significant parallel acceleration of our iteration steps.

To improve the chances for fast convergence one can apply the iteration to both input polynomial and its reverse and can alternate it with other root-finders such as iterative factorization algorithms in Section 10.

1.3. Real eigen-solving and root-finding

Our another achievement is a novel numerical algorithm that approximates all real roots of a polynomial with real coefficients where real roots are much less numerous than the nonreal ones. The latter case is typical both for random input polynomials [29] and in the practice of algebraic–geometric computations, but the known numerical algorithms approximate all real roots not much faster than they approximate all complex roots. This holds in terms of both theoretical estimates and the actual CPU time [30].

To yield our acceleration we combine our simplified RQ iteration with repeated squaring of the matrix functions $M^{(0)}=I+2\sqrt{-1}(M-\sqrt{-1}\,I)^{-1}$ and $(M^{(0)})^{-1}$ where M is the input matrix. On the one hand, such squaring is inexpensive in the case of companion matrices M (see [31,32]) and DPR1 matrices M (see our Theorems 2.1, 7.8 and 7.9). On the other hand, the smallest eigenvalues of the matrices $\frac{(M^{(0)})^k+(M^{(0)})^{-k}}{\|(M^{(0)})^k+(M^{(0)})^{-k}\|}$ converge to zero as k grows large, so that we can readily approximate the eigenspace associated with these eigenvalues, which (as one can easily prove) is precisely the eigenspace associated with the real eigenvalues of the input matrix M.

At that point we can deflate the input matrix M, thus reducing the original task to the approximation of the r real eigenvalues of the resulting $r \times r$ matrix. The latter task is simplified versus the original root-finding task because $r < n$ and because the eigen-solvers in [11,33] are highly effective for matrices having only real eigenvalues.

As an alternative to deflation we can direct the RQ iteration towards the approximation of the r real eigenvalues. Empirically this approach works with just a few or no squarings. This makes it more amenable to parallel acceleration, but even under the sequential model of computing we accelerate the known algorithms dramatically, by the factor n/r.

In this part of our work we were seeking real roots via real eigen-solving for the companion and DPR1 matrices, but the algorithm can be applied to approximate the real eigenvalues of any real matrix and remains highly effective as long as the matrix is structured.

We also show a promising matrix-free variation of this algorithm directed to real polynomial root-finding.

1.4. Summary of our progress, some technical aspects, and a brief discussion

In sum we apply the RQ iteration to the companion and DPR1 matrices, combine it with additive preprocessing, Newton-like linearization, homotopy continuation techniques, Newton's iterative polynomial factorization, and various heuristics. Our algorithms noticeably accelerate the known numerical root-finders for polynomial and secular equations. Parallel processing enables substantial additional speedup, but even under the model of sequential computations we yield dramatic acceleration for the important task of approximating all real roots in the typical case where they are much less numerous than all roots.

Our extensive numerical experiments (the contribution of the second author) are in good accordance with our theoretical study and demonstrate the power of our algorithms.

Our techniques can be of independent interest. Some of them can be extended to root-finding for polynomial systems of equations (see Appendix D) and to real eigen-solving for structured matrices.

There are various natural directions for advancing our study (see Section 12).

1.5. Organization of the paper

We organize our paper as follows. In the next two sections we recall some definitions and basic results on matrix computations and additive preprocessing. In Section 4 we recall and modify the RQ iteration, in particular by employing additive preprocessing. In Section 5 we describe Newton-like linearization of the modified RQ iteration. In Sections 6 and 7 we apply these techniques to the companion and DPR1 generalized companion matrices, respectively, to devise our root-finders. In Sections 8 and 9 we present our real eigen-solver and real root-finder, respectively. In Section 10 we cover some iterative techniques for numerical factorization of a polynomial, which can be applied to deflation and can alternate with the steps of RQ iteration. In Section 11 we cover our numerical experiments. We leave Section 12 for a brief discussion. In the Appendix we recall a number of successful eigen-solving techniques and outline a sample extension to solving polynomial systems of equations.

Sections 9 and 10 can be read independently from the other sections. Section 8 involves the RQ iteration, but can actually use just its rudimentary version in equations (4.1)–(4.4).

2. Some definitions

Hereafter "op" stands for "arithmetic operation".

M^T is the transpose and M^H is the Hermitian transpose of a matrix M.

$(M_1, \dots, M_k) = ((M_i^T)_{i=1}^k)^T$ is a $1 \times k$ block matrix with the blocks $M_1, M_2, \dots, M_k$.

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

$O = O_{k,l}$ is the $k \times l$ matrix filled with zeros.

$I = I_k = (\mathbf{e}_j^{(k)})_{j=1}^k$ is the $k \times k$ identity matrix with columns $\mathbf{e}_1^{(k)}, \dots, \mathbf{e}_k^{(k)}$. To simplify the notation we drop the superscripts (k) and write $\mathbf{e}_j = \mathbf{e}_j^{(k)}$ where this causes no confusion.

$J = J_k = (\mathbf{e}_k, \dots, \mathbf{e}_1)$ is the $k \times k$ reflection matrix, $J^2 = I$.

$\|M\|$ is the 2-norm of a matrix M.

Nonsingular matrices M and linear systems $M\mathbf{y} = \mathbf{f}$ are *ill conditioned* where the condition numbers $\mathrm{cond}(M) = \|M\| \, \|M^{-1}\|$ are large (in the context of the computational task and computer environment) or equivalently where the matrices M are close to singular matrices. In this case the computation of the inverse matrices M^{-1} and the solution vectors $\mathbf{y}$ is prone to magnification of the input and rounding errors and requires higher precision [15,34,35,19]. Otherwise the matrix and the linear systems are *well conditioned*.

$\mathcal{R}(M) = \{\mathbf{z} : \mathbf{z} = M\mathbf{y}$ over all vectors $\mathbf{y}\}$ is the range of a matrix M.

$\mathcal{N}(M) = \{\mathbf{x} : M\mathbf{x} = \mathbf{0}\}$ is its *null space*, made up of the *null vectors* $\mathbf{x}$.

$\rho = \dim(\mathcal{R}(M))$ is the rank of a matrix M. $\nu = \dim(\mathcal{N}(M))$ is its nullity.

$\nu + \rho = n$ for an $n \times n$ matrix M.

$\mathbb{S}$ is a right (resp. left) *invariant subspace* or *eigenspace* of a matrix M if $M\mathbb{S} \subseteq \mathbb{S}$ (resp. $\mathbb{S}M \subseteq \mathbb{S}$).

Suppose B and C are matrices of full rank, $BM = LB$, and $MC = CL$. Then $\{L, B\}$ and $\{L, C\}$ are left and right *eigenpairs* of the matrix M, respectively, and $\{L, B, C\}$ is its *eigentriple*. If $L = \lambda$ is a scalar, $B = \mathbf{b}$ and $C = \mathbf{c}$ are vectors, then $\lambda = \lambda(M)$ is an eigenvalue of the matrix M, whereas $\mathbf{b}$ and $\mathbf{c}$ are the associated left and right eigenvectors.

$\det(M - xI)$ is the *characteristic polynomial* of the matrix M. Its root of a multiplicity ν is an eigenvalue $\lambda(M)$ of algebraic multiplicity $\nu = \nu(\lambda)$.

The dimension $\nu_g = \nu_g(\lambda)$ of the space of the right (as well as left) eigenvectors associated with an eigenvalue λ is its geometric multiplicity, $\nu_g \le \nu$.

An eigenvalue is simple if its algebraic and geometric multiplicities are equal to one.

Hereafter $\lambda_j = \lambda_j(M)$ for $j = 1, 2, \dots, n$ denote the n eigenvalues repeated according to their algebraic multiplicities and listed in the nonincreasing order,

$$|\lambda_1| \ge |\lambda_2| \ge \cdots \ge |\lambda_n|. \tag{2.1}$$

$\Lambda_K(M)$ is the set $\{\lambda_j\}_{j\in K}$ for a subset K of the set $\{1, 2, \dots, n\}$. $\Lambda(M) = \Lambda_K(M)$ for $K = \{1, 2, \dots, n\}$. $\mathbb{S}_K = \mathbb{S}(M, \Lambda_K)$ and $\mathbb{T}_K = \mathbb{T}(M, \Lambda_K)$ are the two eigenspaces of all left and right eigenvectors, respectively, associated with all eigenvalues in this set. The eigenspace $\mathbb{S}_{\{1,\dots,\nu\}}$ is *dominant* and the eigenspace $\mathbb{S}_{\{\nu+1,\dots,n\}}$ is *dominated* if $|\lambda_{\nu+1}/\lambda_\nu| < 1$.

"The SMW formulae" is our abbreviation for the Sherman–Morrison–Woodbury inversion and determinantal formulae

$$(K - UV^H)^{-1} = K^{-1} + K^{-1}UG^{-1}V^HK^{-1}, \quad G = I_r - V^HK^{-1}U, \tag{2.2}$$

$$\det(K - UV^H) = (\det K)\det G \tag{2.3}$$

where $M, K \in \mathbb{C}^{n\times n}$, $U, V \in \mathbb{C}^{n\times r}$, $0 < r < n$, $K = M + UV^H$, and the matrices M and K are assumed to be nonsingular.

Banded matrices $B = (b_{i,j})_{i,j}$ have a lower bandwidth l and an upper bandwidth u if $b_{i,j} = 0$ where $i - j > l$ or $j - i > u$ (cf. [15, Section 1.2.1]). The inverse of such an $n \times n$ matrix (if it is nonsingular) is a rank structured matrix defined by $O((l + u + 1)n)$ parameters [36].

Fact 2.1. *Let an $n \times n$ banded matrix B have a lower bandwidth l and an upper bandwidth u. Then one can multiply this matrix by a vector by using $O((l + u + 1)n)$ ops. The same cost bound holds for the solution of a linear system of equations with the coefficient matrix B provided the matrix and the system are nonsingular.*

Theorem 2.1. *Let $M_i = B_i + U_iV_i^H$, $B_i \in \mathbb{C}^{n\times n}$, $U_i, V_i \in \mathbb{C}^{n\times r}$ for $1 \le r \le n$ and $i = 1, 2$. Then $M = B + UV^H$ where $M = M_1M_2$, $U = (U_1, B_1U_2)$, $V = (M_2^HV_1, V_2)$, $U, V \in \mathbb{C}^{n\times(2r)}$, and $B = B_1B_2 \in \mathbb{C}^{n\times n}$. Furthermore if B_i are lower triangular matrices with bandwidths b_i for $i = 1, 2$, then B is a lower triangular matrix with a bandwidth $b \le b_1 + b_2$.*

Corollary 2.1. *Under the assumption of Theorem 2.1, let b_i and r_i for $i = 1, 2$ denote fixed positive integers and let $n \to \infty$. Then it is sufficient to perform $O(n)$ ops to compute*

(a) *the representation $M = B + UV^H$ for the matrix $M = M_1M_2$,*
(b) *the representation of the matrix $B_1^{-1} \in \mathbb{C}^{n\times n}$ with $O(b_1n)$ parameters as a rank structured matrix if B_1 is a lower triangular and nonsingular matrix (cf. [36]) and*
(c) *the matrices $U_-, V_- \in \mathbb{C}^{n\times r}$ such that $M_1^{-1} = B_1^{-1} + U_-V_-^H$ provided that the matrices B_1 and M_1 are nonsingular.*

Proof. Part (a) is immediately verified. Part (b) is proved in [36]. Part (c) follows from the SMW formula (2.2). □

We use some results on computations with other structured matrices, e.g., Hankel, Toeplitz and Toeplitz-like (see [37] and the bibliography therein). We write

$$F_{\mathbf{p}} = \begin{pmatrix} 0 & & & & -p_0 \\ 1 & \ddots & & & -p_1 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & -p_{n-2} \\ & & & 1 & -p_{n-1} \end{pmatrix}, \quad Z_f = \begin{pmatrix} 0 & & & & f \\ 1 & \ddots & & & 0 \\ & \ddots & \ddots & & \vdots \\ & & \ddots & 0 & 0 \\ & & & 1 & 0 \end{pmatrix}. \tag{2.4}$$

Let f be a nonzero scalar. Then Z_f is the unit f-circulant matrix. $Z = Z_0$ is the $n \times n$ downshift matrix, $F_{\mathbf{p}} = Z - \mathbf{e}_n\mathbf{p}^T$ is the companion matrix of a monic polynomial $p(x) = x^n + \sum_{i=0}^{n-1} p_ix^i$, $\mathbf{p} = (p_i)_{i=0}^{n-1}$. Forward substitution supports the following result.

Lemma 2.1. *Given a vector $\mathbf{w} = (w_i)_{i=0}^{n-1}$ and a nonsingular $n \times n$ diagonal matrix D, one can compute the vector $\mathbf{y} = (y_i)_{i=0}^{n-1} = (D + Z)^{-1}\mathbf{w}$ in at most $2n - 1$ ops. $2n - 2$ ops are sufficient for computing the vector $\mathbf{z} = (I + aZ)^{-1}\mathbf{w}$ for a fixed scalar a.*

Remark 2.1. Parallel acceleration of the latter computations by the factor $n/\log n$ can be achieved based on Cyclic Reduction [38, Section 9.3], [39].

3. Computation in the null space with additive preprocessing and augmentation

Let us recall some results on additive preprocessing from [40, Theorem 3.1 and its corollaries].

Theorem 3.1. *Suppose M is an $n \times n$ matrix having a rank ρ and the nullity $\nu = n - \rho$, U and V are two matrices of size $n \times r$, and the matrix $K = M + UV^H$ is nonsingular. Then*

$$\nu \le \operatorname{rank}(U) \le r, \qquad \mathcal{N}(M) \subseteq \mathcal{R}(K^{-1}U). \tag{3.1}$$

Furthermore $\mathcal{R}(K^{-1}U) = \mathcal{N}(M)$ if $\operatorname{rank}(U) = \nu$.

Corollary 3.1. *Under the assumptions of Theorem 3.1 (except for equation $\operatorname{rank}(U) = \nu$) we have*

(a) $\mathcal{R}(K^{-1}UX) = \mathcal{N}(M)$ *if* $\mathcal{R}(X) = \mathcal{N}(MK^{-1}U)$,
(b) *the converse is true if* $\operatorname{rank}(K^{-1}U) = r$, *and*
(c) $\mathcal{N}(MK^{-1}U) = \mathcal{N}(I_\nu - V^HK^{-1}U)$ *if the matrix U has full rank.*

Recipes for computing the nullity of a matrix.

The following observations (implied by Theorem 3.1 and Corollary 3.1) can be used for computing the nullity of a matrix.

(a) For $n \times r$ matrices U and V and $n \times n$ matrix M with a nullity ν, the matrix $K = M + UV^H$ is singular if $r < \nu$ (in virtue of bounds (3.1)) but is likely to be nonsingular if $r \ge \nu$ and if the matrices U and V are random or random structured (see [41] for specific probability estimates).

(b) Let the matrix K be nonsingular. Then $\mathcal{R}(K^{-1}U) = \mathcal{N}(M)$ for $r = \nu$, whereas $\mathcal{R}(K^{-1}UX) = \mathcal{N}(M)$ if $r > \nu$ and if $\mathcal{R}(X) = \mathcal{N}(MK^{-1}U) = \mathcal{N}(I_r - V^HK^{-1}U)$.

Additive preprocessing $A \Longrightarrow A + UV^H$ and augmentations $A \Longrightarrow K = \begin{pmatrix} A & U \\ S & W \end{pmatrix}$ and $A \Longrightarrow \tilde{K} = \begin{pmatrix} W & S \\ U & A \end{pmatrix} = \begin{pmatrix} O & I_r \\ I_n & O \end{pmatrix} K \begin{pmatrix} O & I_n \\ I_r & O \end{pmatrix}$ are closely linked to each other and have similar power. Theorem 3.1 and other respective results can be extended to the augmented matrices K either directly or based on a factorization in [40, Theorem 4.3], which reduces augmentation to additive preprocessing.

We refer the readers to the papers [42,40,24] on application of additive preprocessing and augmentation to regularization and preconditioning of matrix computations, in particular of the solution of linear systems of equations.

4. The RQ and SQ iterations with preprocessing

The RQ (that is, Rayleigh quotient) iteration has an $n \times n$ matrix M and its approximate eigenpair $\{\lambda^{(0)}, \mathbf{w}_0\}$ as an input and recursively updates the eigenpairs as follows,

$$\mathbf{y}_i = (M - \lambda^{(i)}I)^{-1}\mathbf{w}_i, \tag{4.1}$$

$$\lambda^{(i+1)} = \lambda^{(i)} + \frac{\mathbf{y}_i^H\mathbf{w}_i}{\mathbf{y}_i^H\mathbf{y}_i}, \tag{4.2}$$

$$c_i \approx 1/\sqrt{\mathbf{y}_i^H\mathbf{y}_i}, \mathbf{w}_{i+1} = c_i\mathbf{y}_i \tag{4.3}$$

for $i = 0, 1, \ldots$. It stops and outputs the eigenpair $\{\lambda^{(i)}, \mathbf{w}_i\}$ where

$$\|M\mathbf{w}_i - \lambda^{(i)}\mathbf{w}_i\| < \tau\|M\mathbf{w}_i\| \tag{4.4}$$

for a fixed tolerance τ. One can skip checking this bound where $|\lambda^{(i+1)} - \lambda^{(i)}| > \tau|\lambda^{(i)}|$.

The iteration extends the Power method

$$\mathbf{y}_{i+1} = M\mathbf{y}_i/\|M\mathbf{y}_i\|, \qquad \lambda_1^{(i)} = \frac{\mathbf{y}_i^H\mathbf{y}_{i+1}}{\mathbf{y}_i^H\mathbf{y}_i}, \quad i = 0, 1, \ldots, \tag{4.5}$$

where the pairs $\{\lambda_1^{(i)}, \mathbf{y}_i\}$ converge to the eigenpair $\{\lambda_1, \mathbf{x}_1\}$ provided $|\frac{\lambda_2}{\lambda_1}| \le \theta < 1$ and the eigenvalue λ_1 is simple [15, Section 7.3.1], [19, Section 2.1.1]. The RQ iteration rapidly converges to an eigenpair $\{\lambda_j, \mathbf{y}_j\}$ for almost any initial vector $\mathbf{w}_0$ provided $|\frac{\lambda_j - \lambda^{(0)}}{\lambda_k - \lambda^{(0)}}| \le \theta < 1$ for all $k \ne j$ and for θ not close to one. Unless a reasonably close initial approximate eigenvalue is available, it is customary to choose the initial values $\lambda^{(0)}$ on a large circle $C_{c,\gamma} = \{\lambda^{(0)} : |\lambda^{(0)} - c| = \gamma\}$ for $c = 0$ or $c = \frac{1}{n}\operatorname{trace}M$ and $\gamma \approx 10\|M\|$, say. Empirically this recipe works fine. Apart from rare cases of hard inputs, one can expect to have convergence in quite a small number of iteration loops (cf. Section 11).

Seeking all eigenvalues of an $n \times n$ matrix M one can choose hn equally spaced initial points $\lambda^{(0)}$ on the circle $C_{c,\gamma}$ for $h \ge 1$ (cf. [43]) and concurrently initialize the iteration at all of these points. Some processes can converge to the same eigenvalues

from distinct initial approximations, but typically the iteration approximates a substantial fraction of the eigenvalues, if not all of them [43]. By combining this algorithm with deflation one can recursively approximate all eigenvalues.

The ith iteration loop (4.1)–(4.3) is essentially equivalent to computing Newton's update of an approximate eigenpair $\{\lambda^{(i)}, \mathbf{y}_i\}$ (see [21,22,19]), and this implies local quadratic convergence of the iteration.

Subspace iteration converges under weaker assumptions, whereas *Rayleigh–Ritz* (*Galerkin*) *techniques* split out the eigenspaces associated with a desired number of eigenvalues and avoid convergence to the eigenpairs already approximated (see the Appendix and [44,19]). Furthermore these methods update approximate eigenvectors and eigenspaces faster by employing all vectors from the Krylov linear space defined by all previously computed approximate eigenvectors instead of just the single most recent approximate eigenvector. In application to root-finding, however, this advantage should be weighed against the incurred increase of the computational cost of an iteration step (see Remark B.3).

According to both formal and empirical study, the RQ iteration remains effective wherever instead of the pairs $\{\lambda^{(i+1)}, \mathbf{w}_{i+1}\}$ one computes approximations $\{\tilde{\lambda}^{(i+1)}, \tilde{\mathbf{w}}_{i+1}\}$ such that $(\lambda_g - \tilde{\lambda}^{(i+1)})^{-1} \gg (\lambda_j - \tilde{\lambda}^{(i+1)})^{-1}$ unless $j = g$. Thus to save some ops, we can replace the values c_i in (4.3) with their approximations and (cf. [23]) at the stage (4.2) of updating the eigenvalue replace the RQs with *simple quotients* (hereafter we refer to them as *SQs*),

$$\lambda^{(i+1)} = \lambda^{(i)} + \frac{\mathbf{e}_j^H \mathbf{w}_i}{\mathbf{e}_j^H \mathbf{y}_i}, \quad \mathbf{e}_j^H \mathbf{y}_i \neq 0. \tag{4.6}$$

We can also simplify updating the eigenvectors in (4.3) by incorporating additive preprocessing $M_i = M - \lambda^{(i)} I \to K_i = M_i + \mathbf{u}_i \mathbf{v}_i^H$ for appropriate pairs of vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ such that the linear systems with the matrix K_i can be solved more easily than the ones with the matrix M_i. Indeed the eigenvectors associated with an eigenvalue λ_j are precisely the null vectors of the matrix $M - \lambda_j I$, and our results in the previous section can be applied. Therefore we can replace stage (4.1) in the RQ and SQ iterations with the computation of the vector $\mathbf{y}_i$ equal either to $K_i^{-1}(1 + g_i^{-1}\mathbf{u}_i \mathbf{v}_i^H K_i^{-1})\mathbf{w}_i$ for $g_i = 1 - \mathbf{v}_i^H K_i^{-1}\mathbf{u}_i$ or to $K_i^{-1}\mathbf{u}_i$. We call the two resulting algorithms the *SMW* and *AP iterations*, respectively, each having the RQ and SQ variations. Hereafter we use the abbreviation "*AP*" for "additive preprocessor".

For $\lambda^{(i)}$ equal to an eigenvalue $\lambda_j(M)$, both *SMW* and *AP* iterations compute an associated eigenvector $\mathbf{y}_i$, due to the SMW formula and Corollary 3.1, respectively. For $\lambda^{(i)}$ equal to an eigenvalue $\lambda_j(M)$, the SMW iteration produces the same approximations $\mathbf{y}_i$ as the RQ or SQ iterations up to rounding errors. The AP iteration computes distinct approximations but for $\mathbf{u}_i = \theta_i \mathbf{y}_{i-1}$ and appropriate scalars θ_i preserves local quadratic convergence of the RQ and SQ iterations [41,45].

Remark 4.1. Our approach can be extended to the approximation of an eigenvalue $\lambda_j(M)$ having geometric multiplicity ν or even a cluster or any fixed set of ν simple eigenvalues for $1 \leq \nu \ll n$. One should seek eigenspaces instead of eigenvectors, employ $n \times \nu$ matrices U_i and V_i instead of the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$, and apply the Rayleigh–Ritz (Galerkin) procedure in the Appendix or [44,19] instead of RQ or SQ iteration. See some details in [45].

5. AP iteration with Newton-like linearization

Theorem 5.1. *Let $\{\lambda, X, Y\}$ be an eigentriple of an $n \times n$ matrix M where the eigenvalue λ has algebraic and geometric multiplicity ν (see the definitions in Section 2), and so X and Y are $n \times \nu$ matrices. Furthermore assume that $U, V, \tilde{X}$, and $\tilde{Y}$ are $n \times \nu$ matrices, $0 < \nu < n$, and a triple $\{\tilde{\lambda}, \tilde{X}, \tilde{Y}\}$ approximates an eigentriple $\{\lambda, X, Y\}$. Write $M(\lambda) = M - \lambda I, \tilde{M} = M - \tilde{\lambda} I, K = M + UV^H, K(\lambda) = K - \lambda I$, and $\tilde{K} = K - \tilde{\lambda} I$ and suppose $\tilde{K}^H \tilde{X} = V, \tilde{K}\tilde{Y} = U$, and the matrices $K(\lambda)$ and $\tilde{K}$ are nonsingular. Write $\delta = \lambda - \tilde{\lambda}$ and $\Delta = Y - \tilde{Y}$. Then*

(i) $X^H K(\lambda) = V^H, K(\lambda) Y = U$,
(ii) $\Delta = \delta \tilde{K}^{-1}(I - \delta \tilde{K}^{-1})^{-1}\tilde{Y} = \delta \tilde{K}^{-1}\tilde{Y} + O(|\delta|^2)$,
(iii) $\tilde{M}\tilde{Y} = \delta U T + O(|\delta|^2)$ *where* $T = V^H \tilde{K}^{-2} U$.

Proof. Part (i) follows from Theorem 3.1. Next combine the equations $K(\lambda)Y = U, \tilde{K}\tilde{Y} = U, K(\lambda) = \tilde{K} - \delta I$ and deduce that $\tilde{K}Y = U + \delta Y, Y = \tilde{K}^{-1}U + \delta \tilde{K}^{-1} Y = \tilde{Y} + \delta \tilde{K}^{-1} Y$. Consequently $\Delta = Y - \tilde{Y} = \delta \tilde{K}^{-1} Y$, and so $\Delta = \delta \tilde{K}^{-1}\tilde{Y} + \delta \tilde{K}^{-1}\Delta$, implying part (ii). Now recall that $M(\lambda) = \tilde{M} - \delta I$, and so $M(\lambda)Y = (\tilde{M} - \delta I)Y = O$. Consequently $\tilde{M}Y = \delta Y = \delta \tilde{Y} + O(|\delta|^2), \delta \tilde{Y} = \tilde{M}\tilde{Y} + \tilde{M}\Delta + O(|\delta|^2)$. Replace Δ by its expression from part (ii) and deduce that $\delta \tilde{Y} = \tilde{M}\tilde{Y} + \delta \tilde{M}\tilde{K}^{-1}\tilde{Y} + O(|\delta|^2)$. Substitute the equation $\tilde{M} = \tilde{K} - UV^H$ and obtain that $\delta \tilde{Y} = \tilde{M}\tilde{Y} + \delta \tilde{Y} - \delta U V^H \tilde{K}^{-1}\tilde{Y} + O(|\delta|^2)$. Therefore $\tilde{M}\tilde{Y} = \delta U V^H \tilde{K}^{-1}\tilde{Y} + O(|\delta|^2)$. Substitute $\tilde{Y} = \tilde{K}^{-1}U$ and $T = V^H \tilde{K}^{-2} U$ and obtain part (iii). □

Corollary 5.1. *Under the assumptions of Theorem 5.1, let the matrix U have full column rank. Then $\delta T = G + O(|\delta|^2)$ where $G = I_\nu - V^H \tilde{K}^{-1} U$ and $T = V^H \tilde{K}^{-2} U + O(|\delta|^2)$.*

Proof. First recall that $\tilde{M}\tilde{Y} = \tilde{K}\tilde{Y} - UV^H\tilde{Y} = U - UV^H\tilde{Y} = UG$. Combine this equation with part (ii) to obtain that $\delta^{(i)} U T = UG + O(|\delta|^2)$ and arrive at the corollary. □

Now assume an $n \times n$ input matrix M, fix an initial approximation $\lambda^{(0)}$ to its isolated eigenvalue λ of multiplicity ν, apply Newton-like linearization, that is recursively apply Theorem 5.1 and Corollary 5.1 deleting the terms in $O(|\delta|^2)$, and arrive at the following algorithm.

Algorithm 5.1. *AP iteration with Newton-like linearization* 1.

INITIALIZATION: Compute the matrix $M_0 = M - \lambda^{(0)}I$. Set $i = 0$.

COMPUTATIONS:
1. Generate a pair of $n \times \nu$ matrices U_i and V_i, of full rank ν. Scale them to have the ratio $\frac{\|U_iV_i^H\|}{\|M_i\|}$ neither small nor large.
2. Compute the matrix $K_i = M_i + U_iV_i^H$, expected to be nonsingular.
3. Stop and output FAILURE if the matrix K_i is singular. Otherwise compute the matrix $Y_i = K_i^{-1}U_i$.
4. If the stopping criterion $\|M_iY_i\| \le t\|M\|\,\|Y_i\|$ holds for a fixed tolerance t, output an approximate eigenpair $\{\lambda_i, Y_i\}$ and stop. Otherwise successively compute the matrices $Y_i = K_i^{-1}U_i$, $\tilde{Y}_i = K_i^{-1}Y_i$, $G_i = I_\nu - V_i^HY_i$, and $T_i = V_i^H\tilde{Y}_i$.
5. Fix a pair of integers g and h in the range $r_{\nu,\nu} = \{1 \le g \le \nu,\ 1 \le h \le \nu\}$ and compute the scalar $t_{g,h}^{(i)} = \mathbf{e}_g^TT_i\mathbf{e}_h$. If $t_{g,h}^{(i)} = 0$, then remove the pair $\{g, h\}$ from the range, that is set $r_{\nu,\nu} \leftarrow r_{\nu,\nu} - \{g, h\}$, and repeat Stage 5.
6. Otherwise compute the scalars $\delta^{(i)} = \frac{\mathbf{e}_g^TG_i\mathbf{e}_h}{t_{g,h}^{(i)}}$ and $\lambda^{(i+1)} \leftarrow \lambda^{(i)} + \delta^{(i)}$.
7. Compute the matrix $M_{i+1} = M_i - \delta^{(i)}I$, increment the integer i: $i \leftarrow i + 1$ and reapply the iteration loop, beginning with Stage 1.

Correctness of the algorithm and its local quadratic convergence follow from Theorem 5.1 and Corollary 5.1.

Remark 5.1. One can choose the matrices U_i to simplify the computations in Algorithm 5.1 (cf. the next two sections).

Remark 5.2. Assume that the matrices $X_i = K_i^{-H}V_i$, $Y_i = K_i^{-1}U_i$, and $T_i = X_i^HY_i$ are filled with random and independent entries. Then the ratio $\frac{\|T_i\|}{\|X_i\|\,\|Y_i\|}$ tends to be very large even in the case where the ratio n/ν is moderately large. In our tests, at the initial stages where the approximation errors $\delta^{(i)}$ were not small, this ratio tended to be very large indeed in the case of random matrices U_i and V_i. We avoid this problem if we ensure small approximation errors $\delta^{(i)}$ or if $M = M^H$ and if we choose $V_i = U_i$. In the latter case we have $X_i = K_i^{-H}V_i = K_i^{-1}U_i = Y_i$ and $X^HY_i = \|Y_i\|^2$. We can extend this technique heuristically to the case of non-Hermitian matrices M by first setting $V_i = U_i$ and then recursively redefining the matrix $V_i \leftarrow K_i^{-1}Y_i$ until the norm $\|Y_i\| = \|V_i^HK_i^{-1}Y_i\|$ would grow to the desired level.

Remark 5.3. Corollary 5.1 implies that the norm $\|G_i\| = O(|\delta^{(i)}|)$ is small near an eigenvalue, which leads to numerical problems at the stage of computing the matrix G_i, but one can overcome them with the techniques of iterative refinement in [42].

Remark 5.4. We can modify Algorithm 5.1 near the solution based on the representation of the matrix $K_{i+1}^{-1} = (K_i - \delta^{(i)}I)^{-1} = K_i^{-1}(I - \delta^{(i)}K_i^{-1})^{-1}$ as the formal power series $\sum_{j=0}^{\infty}(\delta^{(i)})^jK_i^{-1-j}$. If the value $|\delta^{(i)}|\,\|K_i^{-1}\|$ is small, we can truncate this series to the first two terms and reduce the computation of the matrices X_{i+1}, Y_{i+1}, G_{i+1}, and T_{i+1} to the solution of linear systems with the same matrix K_i^{-1}. Then we would only need its single factorization and would avoid factorization of the matrices K_{i+j} for $j = 1$ and possibly even for $j = 2, 3, \ldots$. An alternative of solving linear systems with the two matrices K_i^{-1} and $I - \delta^{(i)}K_i^{-1}$ can be also attractive where $\|\delta^{(i)}K_i^{-1}\| < 0.5$, say, so that the matrix $I - \delta^{(i)}K_i^{-1}$ is strongly diagonally dominant.

Let us alternatively compute the scalars $\delta = \delta^{(i)}$ and the matrices $\Delta = \Delta^{(i)}$.

Ignoring the terms in $O((|\delta| + \|\Delta\|)|\delta|)$ deduce from part (ii) of Theorem 5.1 that $\tilde{M}\Delta \approx \delta M\tilde{K}^{-1}\tilde{Y}$.

Furthermore we have $\tilde{M}Y = (M - \tilde{\lambda}I)Y = (M - \lambda I + \delta I)Y$. Consequently $\tilde{M}Y = \delta Y$ because $MY = \lambda Y$. It follows that $\tilde{M}\Delta = \tilde{M}Y - \tilde{M}\tilde{Y} = \delta Y - \tilde{M}\tilde{Y} \approx \delta\tilde{Y} - \tilde{M}\tilde{Y}$. Combine the two expressions for $\tilde{M}\Delta$ and obtain that $\delta(\tilde{Y} - \tilde{M}\tilde{K}^{-1}\tilde{Y}) \approx \tilde{M}\tilde{Y}$ and consequently

$$\delta\mathbf{e}_g^{(n)T}(\tilde{Y} - \tilde{M}\tilde{K}^{-1}\tilde{Y})\mathbf{e}_h^{(\nu)} \approx \mathbf{e}_g^{(n)T}\tilde{M}\tilde{Y}\mathbf{e}_h^{(\nu)} \quad \text{for all pairs of } g \text{ and } h. \tag{5.1}$$

Based on these equations we can modify Algorithm 5.1 as follows.

Algorithm 5.2. *AP iteration with Newton-like linearization* 2.

INITIALIZATION AND STAGES 1–4 OF COMPUTATIONS are as in Algorithm 5.1. Modify the rest of its COMPUTATIONS as follows.

5. Fix a pair of integers g and h in the range $r_{n,\nu} = \{1 \le g \le n,\ 1 \le h \le \nu\}$ and compute the scalar $t_{g,h}^{(i)} = \mathbf{e}_g^{(n)T}(Y_i - M_iK_i^{-1}Y_i)\mathbf{e}_h^{(\nu)}$. If $t_{g,h}^{(i)} = 0$, then remove the pair $\{g, h\}$ from the range, that is set $r_{n,\nu} \leftarrow r_{n,\nu} - \{g, h\}$, and repeat Stage 5.

6. Otherwise compute the scalars $\delta^{(i)} = \frac{\mathbf{e}_g^{(n)T} M_i Y_i \mathbf{e}_h^{(\nu)}}{t_{g,h}^{(i)}}$ and $\lambda^{(i+1)} = \lambda^{(i)} + \delta^{(i)}$ and the vector $\Delta_i = \delta^{(i)} \tilde{\Delta}_i$.
7. Compute the matrix $M_{i+1} = M_i - \delta^{(i)} I$, increment the integer $i : i \leftarrow i+1$ and reapply the iteration loop, beginning with Stage 1.

Correctness of the algorithm follows from part (ii) of Theorem 5.1 and Eq. (5.1).

Recall that $K_i = M_i + U_i V_i^H$, deduce that $I - M_i K_i^{-1} = U_i V_i^H K_i^{-1}$ and therefore $Y_i - M_i K_i^{-1} Y_i = U_i V_i^H K_i^{-1} Y_i$, and obtain yet another modification of the previous algorithms.

Algorithm 5.3. *AP iteration with Newton-like linearization* 3.

Initialization and Stages 1–4, 6 and 7 of Computations are as in Algorithm 5.2. Modify Stage 5 of its Computations as follows.

5. Fix a pair of integers g and h in the range $r_{n,\nu} = \{1 \le g \le n,\ 1 \le h \le \nu\}$ and compute the scalar $t_{g,h}^{(i)} = \mathbf{e}_g^{(n)T} U_i V_i^H K_i^{-1} Y_i \mathbf{e}_h^{(\nu)}$. If $t_{g,h}^{(i)} = 0$, then remove the pair $\{g, h\}$ from the range, that is set $r_{n,\nu} \leftarrow r_{n,\nu} - \{g, h\}$, and repeat Stage 5.

Algorithms 5.1–5.3 output the same values up to the perturbations of the order quadratic in $|\delta^{(i)}|$ and $\|\Delta_i\|$, and so our previous analysis can be extended. Our experiments have showed quite similar convergence patterns for all three algorithms, but the arithmetic cost of an iteration loop in their application to companion and generalized companion matrices a little varies (see Tables 1 and 2 in the next sections).

In all three algorithms we can choose additive preprocessors $U_i V_i^H$ for which the subsequent computations are simplified. e.g., we can turn a Hessenberg matrix M_i into a 2×2 block triangular matrix K_i having two Hessenberg diagonal blocks. In the next sections we yield more substantial simplifications where the matrices M_i are already quite simple.

6. Polynomial root-finding via eigen-solving for companion matrices

6.1. A companion matrix and its eigenspaces

The n roots $\lambda_1, \dots, \lambda_n$ of a monic polynomial $p(x) = \sum_{i=0}^n p_i x^i = \prod_{j=1}^n (x - \lambda_j)$ with the coefficient vector $\mathbf{p} = (p_i)_{i=0}^{n-1}$ are precisely the n eigenvalues of the associated companion matrix $F_{\mathbf{p}}$ in (2.4). (Here we assume that $p_n = 1$, but see Remark 6.1.) We can approximate the roots by applying the algorithms in the previous sections to the matrix $F_{\mathbf{p}}$ and by exploiting its structure. One can immediately verify the following facts and corollary.

Fact 6.1. *Any eigenvalue $\lambda_j = \lambda_j(F_{\mathbf{p}})$ has a left eigenvector $\mathbf{y}_j^T = (\lambda_j^{i-1})_{i=1}^n$.*

Corollary 6.1. *Suppose a companion matrix $F_{\mathbf{p}}$ has n distinct simple eigenvalues $\lambda_j = \lambda_j(F_{\mathbf{p}})$ for $j = 1, 2, \dots, n$. Then $Y F_{\mathbf{p}} Y^{-1} = \mathrm{diag}(\lambda_j)_{j=1}^n$ where $Y = (\mathbf{y}_j^T)_{j=1}^n = (\lambda_j^{i-1})_{j,i=1}^n$.*

Fact 6.2. *Assume a companion matrix $F_{\mathbf{p}}$ defined by the coefficient vector $\mathbf{p}$ of a monic polynomial $p(x) = \sum_{i=0}^n p_i x^i$ and let $r(x) = \frac{u(x)}{w(x)}$ be a rational function for two polynomials $u(x)$ and $w(x)$ such that the matrix $w(F_{\mathbf{p}})$ is nonsingular. Let $\lambda_j = \lambda_j(F_{\mathbf{p}})$ and $r(\lambda_j)$ for $j = 1, 2, \dots, n$ denote the eigenvalues of the matrices $F_{\mathbf{p}}$ and $r(F_{\mathbf{p}})$, respectively, which share their associated eigenvectors for every j. If $p_{\mathbf{r}}(x) = \prod_{j=1}^n (x - r(\lambda_j)) = x^n + \sum_{i=0}^{n-1} p_{\mathbf{r},i} x^i$ and $\mathbf{p}(\mathbf{r}) = (p_{\mathbf{r},i})_{i=0}^{n-1}$ is the vector of the n trailing coefficients of this polynomial, then the matrix $F_{\mathbf{p}(\mathbf{r})}$ has eigenvalues $r(\lambda_j)$ and the associated left eigenvectors $(r(\lambda_j^{i-1}))_{i=1}^n$ for $j = 1, 2, \dots, n$.*

The monic polynomials $p_{\mathrm{rev}}(x) = \frac{1}{p_0} x^n p(1/x) = \sum_{i=0}^n \frac{p_i}{p_0} x^{n-i} = \prod_{j=1}^n \left(x - \frac{1}{\lambda_j}\right)$ (where $p_0 \ne 0$), $p_a(x) = a^n p(x/a)$ (for a scalar $a \ne 0$), and $p(x - \mu) = q(x) = \sum_{i=0}^n q_i x^{n-i} = \prod_{j=1}^n (x - (\lambda_j + \mu))$ have the roots $1/\lambda_j$, $a\lambda_j$, and $\lambda_j + \mu$, respectively, for $j = 1, 2, \dots, n$. Let $\mathbf{p}_{\mathrm{rev}}$, $\mathbf{p}_a$, and $\mathbf{q}$ denote the coefficient vectors of the polynomials $p(x)$, $p_a(x)$, and $q(x)$ above. Then the matrices $F_{\mathbf{p}_{\mathrm{rev}}}$, $F_{\mathbf{p}_a}$, and $F_{\mathbf{q}}$ share their eigenvalues but not eigenspaces with the matrices $F_{\mathbf{p}}^{-1}$, $aF_{\mathbf{p}}$, and $F_{\mathbf{p}} - \mu I$, respectively. For $p_0 \ne 0$ we have

$$F_{\mathbf{p}}^{-1} = Z^T - \left(\frac{p_i}{p_0}\right)_{i=1}^n \mathbf{e}_1^T = J F_{\mathbf{p}_{\mathrm{rev}}} J. \tag{6.1}$$

Since $J = J^{-1}$, it follows that the matrices $F_{\mathbf{p}}^{-1}$ and $F_{\mathbf{p}_{\mathrm{rev}}}$ share their eigenvalues, whereas $J\mathbf{v}$ is an eigenvector of the matrix $F_{\mathbf{p}_{\mathrm{rev}}}$ if and only if $\mathbf{v}$ is a common eigenvector of the matrices $F_{\mathbf{p}}^{-1}$ and $F_{\mathbf{p}}$.

Computation of the coefficients of the polynomial $p(x - \mu)$ takes $O(n \log n)$ ops (see, e.g., [37]). Generally this may require a substantial increase of the input precision, but not for the shifts μ into the points $-\frac{p_{n-1}}{np_n} = \frac{1}{n}\mathrm{trace}(F_{\mathbf{p}})$ and $-n\frac{p_0}{p_1} = n/\mathrm{trace}(F_{\mathbf{p}}^{-1})$ (where $p_1 \ne 0$) because the scaled trace (resp. its reciprocal) is the average value of the roots of the polynomial $p(x)$ (resp. $p_{\mathrm{rev}}(x)$).

Table 1
Number of ops per an SQ loop in four algorithms applied to an $n \times n$ companion matrix.

Algorithm	SQ/GE	SQ/AP	Algorithm 5.1/SQ/AP	Algorithm 5.3/SQ/AP
ops	$7n-3$	$2n+3$	$4n+1$	$3n+4$

Remark 6.1. Scaling by $\frac{1}{p_n}$ reduces any polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$ with $p_n \neq 0$ to the case of monic polynomial. If $p_n \approx 0$, however, then one may prefer to use the recipes in [46,47] or to work with the reverse of the polynomial $q(x) = p(x-s)$ for a scalar s such that the value $|q(0)|$ is not small; in particular one can choose $s = 0$ if the value $|p(0)|$ is not small.

6.2. The RQ iteration and its acceleration

Suppose we apply the RQ iteration in (4.1)–(4.4) to the matrix $F_{\mathbf{p}}$. The iteration updates the approximations to an eigenvector in $8n$ ops based on the SMW formula (cf. (2.2) and (4.1)) and to an eigenvalue in $4n$ ops (cf. (4.2)), computes a square root and performs n ops for scaling in (4.3), and uses $9n-2$ ops for testing stopping criterion (4.4).

We can employ the Subspace iteration and the Rayleigh–Ritz (Galerkin) methods, which have some benefits cited in Section 4, but there is a research challenge of preserving both matrix structure and fast (even local) convergence (see Remark B.3).

In this subsection we advance into the opposite direction of decreasing the cost bounds per iteration loop by such means as employing the SQ iteration and additive preprocessing and modifying the stopping criterion.

1. The SQ iteration in Section 4 updates an eigenvalue in two ops and enables us to skip scaling.
2. With the *simplifying AP* $\mathbf{p}\mathbf{e}_n^T$ we update an approximate eigenvector in $2n-1$ ops per step by modifying expression (4.1) as follows,

$$\mathbf{y}_i = (F_{\mathbf{p}} - \lambda^{(i)} I + \mathbf{p}\mathbf{e}_n)^{-1}\mathbf{p} = (Z - \lambda^{(i)} I)^{-1}\mathbf{p} \quad \text{for } |\lambda^{(i)}| \geq 1. \tag{6.2}$$

Likewise with the *simplifying AP* $(\mathbf{p} + \mathbf{e}_1 + \lambda^{(i)}\mathbf{e}_n)\mathbf{e}_n^T$ we can update an eigenvector in at most $2n-2$ ops as follows,

$$\mathbf{y}_i = (F_{\mathbf{p}} - \lambda^{(i)} I + (\mathbf{p} + \mathbf{e}_1 + \lambda^{(i)}\mathbf{e}_n)\mathbf{e}_n)^{-1}(\mathbf{p} + \mathbf{e}_1 + \lambda^{(i)}\mathbf{e}_n) \quad \text{for } |\lambda^{(i)}| \leq 1 \tag{6.3}$$

where $F_{\mathbf{p}} - \lambda^{(i)} I + (\mathbf{p} + \mathbf{e}_1 + \lambda^{(i)}\mathbf{e}_n)\mathbf{e}_n = (I - \lambda^{(i)} Z)Z_1^T$. In both cases we can compute the vector $\mathbf{y}_i$ in less than $2n$ ops in virtue of Lemma 2.1 and can readily employ concurrency for further acceleration (see Remark 2.1).

The matrix $Z - \lambda^{(i)} I$ is well conditioned for $|\lambda^{(i)}| \geq 1$, whereas the matrix $I - \lambda^{(i)} Z$ is well conditioned for $|\lambda^{(i)}| \leq 1$. In fact we have more options because we can shift to the matrices $F_{\mathbf{p}_{\text{rev}}}$ or $F_{\mathbf{p}}^{-1} = JF_{\mathbf{p}_{\text{rev}}}J$ assuming w.l.o.g. that $p_0 \neq 0$.

3. We can save n ops in approximating an eigenvector if we replace the matrix $F_{\mathbf{p}}$ with its transpose $F_{\mathbf{p}}^T$, which preserves the spectrum of $F_{\mathbf{p}}$. Indeed $(Z - \mu I)^{-T}\mathbf{e}_n = (Z^T - \mu I)^{-1}\mathbf{e}_n = (\mu^{i-n})_{i=0}^{n-1}$. If μ is an eigenvalue, then this is an associated eigenvector of the matrix $F_{\mathbf{p}}^T$ (see Fact 6.1).
4. In the stopping criterion we only need about $4n$ ops if instead of bound (4.4) we test whether $|p(\lambda^{(i)})| \leq us(\lambda^{(i)})$ for the unit roundoff (machine epsilon) u (cf. [48, Section 4]) and for $s(\lambda) = \sum_{j=0}^{n}(4j+1)|p_j|\,|\lambda^{(i)}|^j$. Furthermore since $\mathbf{x}_j = (\lambda_j^h)_{h=0}^{n-1}$ is a left (resp. right) eigenvector associated with an eigenvalue λ_j of the matrix $F_{\mathbf{p}}$ (resp. of $F_{\mathbf{p}}^T$) (cf. Fact 6.1), we can skip testing unless the ratio of two fixed consecutive components of the current approximation to an eigenvector is close to $\lambda^{(i)}$ and unless the value $|\lambda^{(i)} - \lambda^{(i-1)}|$ is small enough.

In our experiments a few initial steps of our simplified iteration (which employed Eq. (6.2) and a simplifying AP) and of the original RQ or SQ iteration with no preprocessing have regularly produced approximations to an eigenvalue of about the same quality. Then our simplified iteration stopped refining these approximations any further. At this point, however, we shifted to the RQ or SQ iteration with no preprocessing. Finally, having computed an approximation that was reasonably close to an eigenvalue, we refined it by applying Algorithm 5.1 or Algorithm 5.3 with the same simplifying APs. Our tests confirm fast convergence of this three-stage iteration.

Table 1 displays the numbers of ops per step in these variations of the SQ iteration where "GE" stands for "Gaussian elimination". In the ops count for Algorithms 5.1 and 5.3 we assumed that $t_{g,h}^{(i)} \neq 0$, that is the denominators have not vanish, already for the first choices of the pairs $\{g, h\}$.

Concurrent application of our iterations to the matrices $F_{\mathbf{p}_{\text{rev}}}$, $F_{\mathbf{p}_a}$, $F_{\mathbf{q}}$, $F_{\mathbf{p}}^{-1}$, $aF_{\mathbf{p}}$, and $F_{\mathbf{p}} - \mu I$ (see Section 6.1) would increase the chances to approximate a sufficiently large fraction of all roots between the successive deflations.

6.3. Initialization and continuous scaling

One can apply the standard initialization recipes for polynomial root-finding, in particular Bini's effective heuristic algorithm in [48, Section 2], which involves $O(n \log n)$ ops. According to Bini's tests in [48] the algorithm produces reasonable

approximations to all root radii, that is to the distances $r_j = |\lambda_j|$, $j = 1, \dots, n$ from the roots to the origin. Then, according to Bini's recipe, one should uniformly distribute hn initial approximations for a fixed $h \geq 1$ (e.g., $h = 3\log_2 n$) in the respective narrow annuli lying about the circles $\{x : |x| = r_j\}$, $j = 1, \dots, n$. Some sets of circles can lie close to each other and be covered by the same annuli. According to the tests in [48,23], this initialization policy enables quite fast convergence and decreases the chances for recomputing the roots already computed. For some input polynomials some roots can still be missing, but one can obtain more roots by applying the iteration concurrently to the polynomials $p(x)$, $p_{\rm rev}(x)$, and possibly $p(x-s)$ and $p_{\rm rev}(x-s_1)$ for some selected shifts s and s_1, e.g., for the shifts $s = -\frac{p_{n-1}}{np_n}$ and $s_1 = -\frac{p_1}{np_0}$ into the average values of the roots.

Standard support for initialization also comes from homotopy continuation techniques. One can first choose a family of polynomials $p(u, x)$ continuously depending on a real parameter u in a fixed range $[s, t]$ where $s < t$ and the roots of a polynomial $p(s, x)$ are easy to approximate, whereas $p(t, x) = p(x)$. Then one can choose a sequence of values $u_0 = s, u_1, \dots, u_q = t$ with sufficiently small step sizes $|u_{k+1} - u_k|$ for all k and recursively approximate the roots of the polynomial $p(u_{k+1}, x)$ by using the initial approximations by the computed roots of the polynomial $p(u_k, x)$ for $k = 0, 1, \dots, q-1$.

In the most customary variant of this process (cf., e.g., [49]), one chooses $s = 0$, $t = 1$, and $p(u, x) = p(x) + (1-u)a^n$ where the value $|a|$ is large enough so that the values $a\omega_n^j$ (for $j = 0, 1, \dots, n-1$ and ω_n denoting a primitive nth root of one) are reasonable initial approximations to the roots $\lambda_j^{(0)}$ of the polynomial $p(0, x)$.

Apparently there are many other effective variations of these techniques. Our tests show the efficiency of the heuristic policies where $p(u, x) = \text{fl}\left(u^n p\left(\frac{x}{u}\right)\right)$, $t = 1$, $|s|$ is small, and $\text{fl}(f)$ denotes a polynomial f whose coefficients are represented with floating point and are rounded to double precision.

6.4. Deflation

Deflation is a reliable way of decreasing the problem size and avoiding convergence to the same eigenvalue. Suppose we have computed the eigenvalues $\lambda_{k(1)}, \dots, \lambda_{k(h)}$ of the companion matrix $F_{\mathbf{p}}$. Then we can deflate the matrix by applying the Rayleigh–Ritz (Galerkin) methods. These customary techniques, however, are too costly in our case because they do not preserve the structure of the matrix $F_{\mathbf{p}}$. Instead we can divide (with no remainder) the polynomial $p(x)$ by the product $d(x) = \prod_{j=1}^{h}(x - \lambda_{k(j)})$ by applying the classical polynomial division (which uses $(2n-h)h$ ops). Alternatively we can apply Toom's approach in [50], that is, first evaluate both polynomials $p(x)$ and $d(x)$ at the 2^lth roots of unity $\omega_j = \exp(2\pi j\sqrt{-1}/2^l)$ for $j = 0, 1, \dots, 2^l - 1$ and $l = 1 + \lfloor \log_2(n-h) \rfloor$, then concurrently compute the 2^l quotients $q(\omega_j) = p(\omega_i)/d(\omega_j)$ for $j = 0, 1, \dots, 2^l - 1$, and finally interpolate to the quotient polynomial $q(x)$. This takes $O(n \log n)$ ops if we apply FFT-based fast evaluation and interpolation algorithms amenable to parallel acceleration.

We can refine the output as follows. Write $p(x) = d(x)(q(x) + \Delta(x))$ where $q(x)$ is the computed quotient and $\Delta(x)$ is the error polynomial. Then we can compute the error polynomial $\Delta(x) = (p(x) - d(x)q(x))/d(x)$ and the refined quotient $q(x) + \Delta(x)$. Such Newton-like steps can be repeated recursively and can be expressed in terms of operations with the associated structured matrices (see Section 10).

6.5. Repeated squaring techniques for a companion matrix

In this section we recall *repeated squaring* of a (shifted) companion matrix $F_{\mathbf{p}}$. The algorithm quite rapidly approximates its complex eigenvalues and has solid formal support in [31,32], but in our tests with random companion matrices was still outperformed by the RQ iteration and its modifications. Application to approximating real roots of a polynomial in Section 8 may give repeated squaring new life.

Write $F^{(0)} = F_{\mathbf{p}}$ and recursively compute the matrices $F^{(i+1)} = (F^{(i)})^2$ for $i = 0, 1, \dots$. The impact of i steps of repeated squaring amounts to the impact of 2^i steps of the Power Iteration (4.5), whose convergence therefore is dramatically accelerated.

Furthermore, squaring and pairwise multiplication of rational matrix functions $r(F_{\mathbf{p}})$ can be reduced essentially to a small number of FFTs and performed in $O(n \log n)$ ops in numerically stable way (see [31], [32, Section 6]). Every matrix $r(F_{\mathbf{p}})$ has Toeplitz-like structure, has displacement rank at most two, and can be inverted in $O(n \log^2 n)$ ops if it is nonsingular (cf., [31,32], [37, Chapter 5]).

Remark 6.2. Repeated squaring is effective for the class $r(F_{\mathbf{p}})$, but tends to destroys other matrix structures quite rapidly. E.g., in about $\log_2 n$ repeated squarings, an $n \times n$ tridiagonal, Toeplitz, Hankel, Vandermonde and Cauchy matrices generally become unstructured, and since then their squaring requires the order of n^3 ops, although one can obtain some practical acceleration by using block matrix algorithms on multiprocessors [15, Chapter 6].

The h initial squarings of the matrix $F_{\mathbf{p}} - \mu I$ as well as its inverse (where it is nonsingular) are less costly. They use $O(hn)$ ops in virtue of Theorem 2.1 applied to the matrices $F_{\mathbf{p}} - \mu I = B + UV^H$ for $B = Z - \mu I$, $U = -\mathbf{e}_n$, and $V = \mathbf{p}$.

Seeking approximations to other roots of the polynomial $p(x)$, we can reapply repeated squaring by using explicit deflation in Section 6.4 or implicit deflation in [31], [32, Section 6].

7. Polynomial root-finding via DPR1 eigen-solving

7.1. DPR1 matrix, its eigenspaces, and back and forth transforms into companion matrices

Companion matrix $F_{\mathbf{p}}$ and its transpose are the best known examples of generalized companion matrices whose eigenvalues are precisely the roots of a polynomial $p(x) = \sum_{i=0}^{n} p_i x^i = \prod_{j=1}^{n}(x - \lambda_j)$. Among the other important classes [51,8], we choose the DPR1 (that is diagonal + rank-one) matrices

$$C = C_{\mathbf{s},\mathbf{u},\mathbf{v}} = D_{\mathbf{s}} - \mathbf{u}\mathbf{v}^H \tag{7.1}$$

for $\mathbf{s} = (s_i)_{i=1}^n$, $\mathbf{u} = (u_i)_{i=1}^n$, $\mathbf{v} = (v_i)_{i=1}^n$,

$$D_{\mathbf{s}} = \operatorname{diag}(s_i)_{i=1}^n, \qquad \prod_i s_i \neq 0, \tag{7.2}$$

$$d_i = u_i v_i = \frac{p(s_i)}{q_i(s_i)} \neq 0, \qquad q_i(x) = \prod_{j\neq i}(x - s_i), \quad i = 1, \ldots, n, \tag{7.3}$$

$$q_i(s_i) = q'(s_i), \quad i = 1, \ldots, n, \qquad q(x) = \prod_{j=1}^{n}(x - s_i). \tag{7.4}$$

Note that $C - \mu I$ for a scalar μ is also a DPR1 matrix. Furthermore, unlike the companion matrices, DPR1 matrices are defined by the values of the associated polynomial on a fixed set of points rather than by the coefficients.

Theorem 7.1. *The eigenvalues of the matrix C in (7.1) are precisely the roots of the polynomial $p(x)$ as well as of the associated* secular equation

$$\sum_{i=1}^{n} \frac{u_i v_i}{s_i - \lambda} = 1. \tag{7.5}$$

Proof. See, e.g., [11,9], [23, Theorem 4.4]. □

Theorem 7.2. *Assume n distinct scalars $s_1, \ldots, s_n$ and let the DPR1 matrix C in Eq. (7.1) have n distinct eigenvalues $\lambda_1, \ldots, \lambda_n$. Then it has the eigendecomposition $C = W^{-1} D_{\Lambda} W$ where $D_{\Lambda} = \operatorname{diag}(\lambda_j)_{j=1}^n$, $W = \left(\frac{u_i}{s_i - \lambda_j}\right)_{i,j=1}^n$, and $W^{-1} = \left(\frac{v_j}{s_i - \lambda_j}\right)_{i,j=1}^n$.*

Proof. The theorem follows from Theorem 3.1. □

For two fixed sets of distinct knots $s_1, \ldots, s_n$ and values $d_1, \ldots, d_n$, we can define infinite number of DPR1 matrices $D_{\mathbf{s}} - \mathbf{u}\mathbf{v}^H$ with $u_i v_i = d_i$ for all i. All of them share their eigenvalues (but not eigenspaces).

DPR1 and companion matrices for the same polynomial can be transformed into one another based on polynomial interpolation and multipoint evaluation [37, Sections 3.1 and 3.3] that take almost linear arithmetic time, but these transformations generally require using extended precision to counter numerical stability problems.

Theorem 7.3. *$O(n \log^2 n)$ ops are sufficient for the transition from the companion matrix $F_{\mathbf{p}}$ to the DPR1 matrix C in (7.1) associated with the same polynomial $p(x)$. The bound decreases to $O(n \log n)$ in the case of DPR1 matrices with the knots $s_i = a\omega_k^{i-1}$ for $i = 1, \ldots, n$, where a is a nonzero scalar, $\omega_q = \exp(2\pi\sqrt{-1}/q)$ denotes a primitive q-th root of unity, $k \geq n$ and $k = O(n)$.*

7.2. Some basic operations with DPR1 matrices

Next, for a given DPR1 input matrix C in (7.1), we estimate the arithmetic cost of computing the DPR1 matrices $C - \mu I$, C^{-1}, and C_{rev} (associated with the reverse polynomial $p_{\text{rev}}(x)$).

Theorem 7.4. *Suppose $3n+1$ scalars μ, u_i, v_i, and s_i for $i = 1, \ldots, n$ define a DPR1 generalized companion matrix C in Eq. (7.1). Write $s = 1 - \sum_{i=1}^{n} \frac{u_i v_i}{s_i}$ and let $s \neq 0$. (For $s = 0$ Eq. (7.5) has the root $\lambda = 0$.) Then we can compute $3n$ parameters $u_i^{(\text{new})}$, $v_i^{(\text{new})}$, and $s_i^{(\text{new})}$, $i = 1, \ldots, n$, that define a DPR1 generalized companion matrices (a) $C - \mu I$ in n ops, (b) C^{-1} in $6n$ ops (provided the matrix C is nonsingular), and (c) C_{rev} associated with the polynomial $p_{\text{rev}}(x)$ in $4n + 1$ ops.*

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

Algorithm	[23]	SQ/AP	Algorithm 5.1/SQ/AP	Algorithm 5.3/SQ/AP
ops	$9n$	$3n+2$	$5n+1$	$4n+2$

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) Apply the SMW formula to obtain that $C^{-1} = (D - \mathbf{u}\mathbf{v}^H)^{-1} = D^{-1} + g^{-1}D^{-1}\mathbf{u}\mathbf{v}^H D^{-1} = D_- + \mathbf{u}_-\mathbf{v}_-^H$ where $D_- = D^{-1}$ (n ops), $\mathbf{w} = D^{-1}\mathbf{u}$ (n ops), $g = 1 - \mathbf{v}^H\mathbf{w}$ ($2n$ ops), $\mathbf{u}_- = g\mathbf{w}$ (n ops), and $\mathbf{v}_-^H = \mathbf{v}^H D^{-1}$ (n ops).

(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

$$\sum_{i=1}^{n} \frac{d_i^{(\text{new})}}{s_i^{(\text{new})} - (1/\lambda)} = 1 \tag{7.6}$$

for $d_i^{(\text{new})} = u_i^{(\text{new})} v_i^{(\text{new})}$ and for all values λ satisfying Eq. (7.5). First rewrite Eq. (7.6) 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 deduce that Eq. (7.6) 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 deduce that $s^{(\text{new})} = 1/s$ and Eqs. (7.5) and (7.6) are equivalent to one another. It remains to compute $s_i^{(\text{new})} = 1/s_i$ (in n ops), $w_i = d_i/s_i$ (in n ops), $u_i^{(\text{new})} = w_i/s_i$ (in n ops) for $i = 1, \dots, n$, $-s = \sum_{i=1}^{n} w_i - 1$ (in n ops), $v_i^{(\text{new})} = -1/s$ for $i = 1, \dots, n$ (in single division). □

7.3. The RQ and SQ iterations for DPR1 matrices and its modification

Assume the SQ iteration applied to a DPR1 matrix C in Eq. (7.1). It updates an approximate eigenvalue as in Sections 4 and 6.2. To update an eigenvector apply the SMW formula (2.2) for $r = 1$.

Theorem 7.5. *For the vector $\mathbf{u} = (\pm 1, \pm 1, \dots, \pm 1)^T$, a scalar μ, a vector $\mathbf{w}$, and the DPR1 matrix C in Eq. (7.1), we can compute the vector $(C - \mu I)^{-1}\mathbf{w}$ by performing $9n$ ops provided the matrix $C - \mu I$ is nonsingular.*

Proof. See [23, Theorem 5.1]. □

With the *simplifying AP* $\mathbf{u}\mathbf{v}^H$ we reduce updating an eigenvector in the SQ iteration to computing the vector $(D - \mu I)^{-1}\mathbf{w}$; this takes $2n$ ops. Moreover we can perform these ops in $\lceil 2n/s \rceil$ arithmetic parallel steps if we can distribute them among s processors for any s, $2 \le s \le n$.

Theorem 7.6. *For a scalar μ, four vectors $\mathbf{s}, \mathbf{u}, \mathbf{v}$, and $\mathbf{w}$ of dimension n, and the DPR1 matrix C in Eq. (7.1), let the matrix $K = C + \mathbf{u}\mathbf{v}^H - \mu I = D_{\mathbf{s}} - \mu I$ be nonsingular. Then we can compute the vector $K^{-1}\mathbf{w}$ by performing $2n$ ops.*

As a stopping criterion we can just check whether secular Eq. (7.5) is satisfied for a fixed scalar λ within a fixed tolerance bound. This takes $2n$ ops assuming that the products $d_i = u_i v_i$ have been given to us for all i. Moreover these ops can be reused when we update an approximate eigenvalue $\lambda^{(k)}$ in Algorithm 5.1 based on the formula

$$\lambda^{(k+1)} = \lambda^{(k)} + \frac{1 - \beta_k}{\gamma_k}, \quad \beta_k = \sum_{j=1}^{n} \frac{d_j}{s_j - \lambda^{(k)}}, \quad \gamma_k = \sum_{j=1}^{n} \frac{d_j}{(s_j - \lambda^{(k)})^2} \tag{7.7}$$

provided $s_j \ne \lambda^{(k)}$ for all pairs $\{j, k\}$. This updating takes $5n + 1$ ops.

In Table 2 we display the number of ops per step of our SQ eigen-solving iterations applied to a DPR1 matrix (under the same assumptions as in Table 1 for companion matrices).

Here is the specification of Theorem 5.1 to a DPR1 matrix C.

Theorem 7.7. *We have $|\lambda^{(k+1)} - \lambda| = O(|\lambda^{(k)} - \lambda|^2)$ as $|\lambda^{(k)} - \lambda| \to 0$ for $\lambda^{(k+1)}$ in Eq. (7.7) and an eigenvalue λ of the matrix C in (7.1).*

7.4. Initialization and continuous scaling

The initialization and deflation recipes in Section 6.3 can be applied in the case of DPR1 matrices as well, including concurrent application of the eigenvalues of DPR1 matrices associated with the polynomials $p(x)$, $p_{\rm rev}(x)$, and possibly $p(x-s)$ and $p_{\rm rev}(x-s_1)$ for some selected shifts s and s_1 (cf. Theorem 7.4). Moreover, versus the companion matrices, we have numerous options of employing the DPR1 matrices $C = D + \mathbf{u}\mathbf{v}^T$ that share their eigenvalues for distinct pairs of the vectors $\mathbf{u}$ and $\mathbf{v}$ such that $u_i v_i = d_i$ for a fixed set $\{d_1, \dots, d_n\}$ and for all i.

Bini's heuristic initialization in [48, Section 2] involves the coefficients $p_0, \dots, p_n$ not available in the DPR1 case, but our experiments show that random choice of the initial approximations to the eigenvalues on the unit circle (for random DPR1 inputs in the range $\{0, 1\}$) serves as effectively. For various other inputs one can try initial approximations on a sufficiently large circle centered near the origin or the point $\frac{1}{n}\text{trace}(C)$, the average of the eigenvalues, and can employ deflation where the iteration converges to the same root from distinct initial points. Then again we can apply the iteration concurrently to the polynomials $p(x)$, $p_{\rm rev}(x)$, and possibly $p(x-s)$ and $p_{\rm rev}(x-s_1)$ for some selected shifts s and s_1, to approximate more roots.

Continuous scaling can be easily applied to DPR1 matrices, and the standard homotopy continuation process, defined by the equation $p(u, x) = p(x) + (1-u)a^n$, can be readily extended to the DPR1 inputs. We only need to perform n divisions by $q_1(s_1), \dots, q_n(s_n)$ per homotopic step provided all divisors $q_i(s_i)$ have been precomputed. One can also explore other policies, e.g., continuous scaling of the parameters s_i and $d_i = u_i v_i$ for all i.

7.5. Deflation

We can deflate the $n \times n$ DPR1 matrix in $4n-4$ ops in a numerically stable algorithm, to arrive at an $(n-1) \times (n-1)$ DPR1 matrix (see [23, Section 6]).

Suppose for some $l < n$ we have computed l eigenvalues $\lambda_{n-l+1}, \dots, \lambda_n$ of a DPR1 matrix C in Eq. (7.1). Then we can compute the remaining eigenvalues $\lambda_1, \dots, \lambda_{n-l}$ as the eigenvalues of an $(n-l) \times (n-l)$ DPR1 matrix $\tilde{C}$ defined by the subvector $(s_i)_{i=1}^{n-l}$ of the vector $\mathbf{s} = (s_i)_{i=1}^n$ and by the vector $\tilde{\mathbf{d}} = (\tilde{d}_i)_{i=1}^{n-l}$ where $\tilde{d}_i = d_i \prod_{j=1}^{n-l} \frac{s_i - s_{n+1-j}}{s_i - \lambda_{n+1-j}}$, $i = 1, \dots, n-l$. Overall the computation of the matrix $\tilde{C}$ takes $4(n-l)l$ ops and can be readily distributed among up to $n-l$ processors assigned to computing the vector $\tilde{\mathbf{d}}$.

7.6. Updating a DPR1 matrix

For the task of the computation of all eigenvalues of a DPR1 matrix we have an additional resource, not available for companion matrices, but successfully exploited in [10,9] for convergence acceleration in the DPR1 case. Namely, we can begin with crude approximations $s_1, \dots, s_n$ to the eigenvalues and then recursively update them (and respectively update the DPR1 matrix based on Eqs. (7.1)–(7.4)) as we improve the approximations. This updating takes $9n-8$ ops when we change a single approximation s_i into a new one $\tilde{s}_i$. Indeed besides n divisions in (7.3), we need four ops to compute the value $\tilde{q}'(s_j) = q'(s_j)\frac{s_j - \tilde{s}_i}{s_j - s_i}$ for any integer j, $j \neq i$, $2n-3$ ops to compute the value $\tilde{q}'(s_i) = \prod_{j \neq i}(\tilde{s}_i - s_j)$, and $2n-1$ ops to compute the value $p(\tilde{s}_i)$. Using Taylor expansion of the polynomial $p(x)$ at the point s_i would stabilize numerical computation of the latter value provided the ratio $|\tilde{s}_i - s_i|/|s_i|$ is noticeably less than one. Likewise we can update l approximate eigenvalues by performing $(9n-8)l$ ops and can readily achieve parallel acceleration provided that $l > 1$ and that we update a number of approximations s_i simultaneously.

7.7. Repeated squaring techniques for a DPR1 matrix

Unlike the companion matrix structure, the DPR1 structure deteriorates in squaring, so that the order of $\log n$ squaring steps can completely destroy the structure of a DPR1 matrix. Let us show some remedies for the latter disadvantage. First of all in $O(n)$ ops we can perform the first h squarings of an $n \times n$ DPR1 matrix and of its inverse where the matrix is nonsingular and h is a small constant. Here are some specific estimates.

Theorem 7.8 (*Cf.* Theorem 2.1). *The ith successive squaring of the shifted DPR1 matrix $C_0 = C - \mu I = D + \mathbf{u}\mathbf{v}^T$ takes at most $2^{2i+1}n$ multiplications and $4^i(2n-1)$ additions for $i \leq \log_2 n$.*

Proof. Write $U_0 = \mathbf{u}$, $V_0 = \mathbf{v}$, $C_{i+1} = D^{2^{i+1}} + D^{2^i}U_iV_i^T + U_iV_i^TD^{2^i} + U_iV_i^TU_iV_i^T$, $i = 0, 1, \dots$, represent the output matrix as $C_{i+1} = D^{2^{i+1}} + U_{i+1}V_{i+1}^T$ where $U_{i+1} = (D^{2^i}U_i, U_i)$, $V_{i+1}^T = (V_i^T, V_i^TD^{2^i} + V_i^TU_iV_i^T)$, and count ops used in the squaring $C_i \to C_{i+1} = C_i^2$. □

Furthermore our next theorem applied to $r = 2$ and DPR2 matrix $(D - \mathbf{u}\mathbf{v}^H)^2$ enables us to square a DPR1 matrix recursively in $O(n \log n)$ ops per squaring, although this squaring is implicit, does not preserve the eigenvectors, and has numerical deficiency of employing the characteristic polynomial rather than just the eigenvectors and eigenspaces. We state this theorem in a more general form than we need in this paper (see [52] on its more narrow version).

First we define $n \times n$ DPRr matrices as diagonal + rank-r matrices of the form $C = D - UV^H$ where D is an $n \times n$ diagonal matrix and U and V are $n \times r$ matrices. If both matrices C and D are nonsingular we can apply the SMW formula and deduce that C^{-1} is a DPRr matrix as well. Furthermore we immediately observe that $C_1C_2 = D_1D_2 - U_1\tilde{V}_1 - \tilde{U}_2V_2$ where $C_j = D_j - U_jV_j^H$ are DPRr_j matrices for $j = 1, 2$, $\tilde{V}_1 = V_1^H(D_2 - U_2V_2^H)$ and $\tilde{U}_2 = D_1U_2$, so that C_1C_2 is a DPR$(r_1 + r_2)$ matrix.

Theorem 7.9. (a) *For $2n$ distinct scalars $s_1, \dots, s_n, \mu_1, \dots, \mu_n$ define the diagonal matrix $D = \mathrm{diag}(s_i)_{i=1}^n$, a pair of $n \times r$ matrices U and V, and the* DPRr *matrix $C = D - UV^H$. Then it is sufficient to use $O((r^3 + \log^2 n)n)$ ops to compute the values $\det(C - \mu_hI)$ for $h = 1, \dots, n$.*

(b) *Furthermore $O((r^3 + \log n)n)$ ops are sufficient if $s_i = a\omega_k^{i-1}$ and $\mu_h = b\omega_l^{h-1}$ for $h,\ i = 1, \dots, n$ and two nonzero scalars a and b, where ω_q denotes a primitive q-th root of one, $k \geq n$, $l \geq n$, and $k + l = O(n)$.*

Proof. We have $\det(C - \mu_hI) = (\det(D - \mu_hI))\det(I_r - V^H(D - \mu_hI)^{-1}U)$ (cf. (2.3)). Within the claimed cost bounds we compute at first the coefficients and then the values $\det(D - \mu_hI) = \prod_{j=1}^n(s_j - \mu_h)$ of the polynomial $\prod_{j=1}^n(s_j - x)$ at the n points $x = \mu_h,\ h = 1, \dots, n$ (cf. [37, Section 3.1]). It remains to compute the $r \times r$ matrices $I_r - G_h = V^H(D - \mu_hI)^{-1}U = \left(\sum_{j=1}^n \frac{v_{ij}u_{jk}}{s_j - \mu_h}\right)_{i,k=0}^{r-1}$, $h = 1, \dots, n$ (in $O(nr)^2$) ops), then the $r \times r$ matrices G_h for $h = 1, \dots, n$ (in rn ops), and finally the n values of their determinants (in $(4r^3 - 15r^2 + 23r - 6)n/6$ ops by means of Gaussian elimination). □

We can readily extend the theorem to the cases where the diagonal matrix D is replaced with a bidiagonal matrix $B = (b_{ij})_{i,j}$, $b_{ij} = 0$ for $i > j+1$ and $i < j$, or a tridiagonal matrix $T = (t_{ij})_{i,j}$, $t_{ij} = 0$ for $|i-j| > 1$. Indeed the matrix $(P - \mu_hI)^{-1}U$ can be computed in $O(nr)$ ops for $P = B$ (see Lemma 2.1) and $P = T$ [15, Section 4.36]. Furthermore $\det(B - \mu_hI) = \prod_{j=1}^n(b_{jj} - \mu_h)$, whereas one can compute the coefficients of the characteristic polynomial $\det(xI - T)$ in $O(n\log^2 n)$ ops based on [15, Eq. (8.5.2)].

Theorem 7.9 bounds the cost of computing the values of the characteristic polynomial $\det(\mu_hI - C)$ of the matrix C at n points $\mu_1, \dots, \mu_n$. Within the same cost bound we can compute the coefficients of the polynomial $q(x) = \prod_{h=1}^n(x - \mu_h)$ and the values $q'(\mu_h)$ for all h, thus defining a DPR1 matrix that shares its eigenvalues with the matrix C (cf. (7.1)–(7.4)). In particular we can apply the algorithm supporting this theorem for $r = 2$ to compute (in $O(n\log^g n)$ ops for $g = 1$ or $g = 2$) a DPR1 matrix sharing the eigenvalues with the squares of the eigenvalues of a given DPR1 matrix. This enables us to extend the repeated squaring techniques in Section 6.5 to DPR1 matrices.

Throughout the process of squaring we can choose the values s_i and μ_h to our advantage. We can choose $s_i = \omega_{3^k}^{i-1}$ for $i = 1, 2, \dots, n$ and an integer k such that $n \leq 3^k < 3n$. Then in all squarings we would have $s_i^{2^g} = \omega_{3^k}^{(i-1)j}$ for $i = 1, 2, \dots, n,\ j = 2^g \mod 3^k,\ g = 1, 2, \dots$, and so we can apply part (b) of Theorem 7.9. One can readily modify the squaring stages in our algorithm to compute cubic powers rather than squares, and then we can choose $s_i^{2^g} = \omega_{2^k}^{(i-1)j}$ for $i = 1, 2, \dots, n,\ j = 2^g \mod 2^k,\ g = 1, 2, \dots$, and employ the FFT subroutines.

8. Real eigen-solving

In this section we approximate the real eigenvalues of a real non-Hermitian matrix M, which may also have nonreal eigenvalues. We are motivated by the two special cases where M is a companion matrix $F_{\mathbf{p}}$ or a DPR1 matrix C, but our algorithm can be applied to any matrix M.

We assume that the matrix $M^2 + I$ is nonsingular for otherwise $\lambda = \pm\sqrt{-1}$ are the eigenvalues of the matrix M, and we can deflate it. Alternatively we can shift to the matrix aM for a random real $a \neq 0$, and then the matrix $a^2M^2 + I$ is nonsingular with a probability close to one. We begin with the following simple observation.

Fact 8.1. *The transition $M \to M^{(0)} = I + 2\sqrt{-1}(M - \sqrt{-1}I)^{-1}$ maps the real eigenvalues of a matrix M onto the unit circle $C_1 = \{x : |x| = 1\}$.*

It remains to approximate the eigenvalues of the matrix $M^{(0)}$ lying on the circle C_1.

By squaring a matrix we square its eigenvalues. Therefore repeated squaring of the matrices $M^{(0)}$ and $(M^{(0)})^{-1}$ keeps the eigenvalues on the circle C_1, so that the respective eigenvalues of the matrices $\frac{1}{2}((M^{(0)})^{2^k} + (M^{(0)})^{-2^k})$ lie in the unit disc $D_1 = \{x : |x| \leq 1\}$ for all integers k, whereas the absolute values of all the other eigenvalues of these matrices converge to ∞ as $k \to \infty$. Thus the eigenspace of such a matrix associated with the former eigenvalues is dominated as $k \to \infty$ unless $M^{(0)}$ is a unitary matrix, that is unless $M^{(0)}(M^{(0)})^H = I$. This gives us a chance to approximate such eigenspaces of the matrices $\frac{1}{2}((M^{(0)})^{2^k} + (M^{(0)})^{-2^k})$ already for moderate integers k, which is precisely the eigenspace of the matrix M associated with its real eigenvalues. Then we can readily approximate the real eigenvalues themselves. In this algorithm we compute the matrices $(M^{(0)})^{2^k}$ and $(M^{(0)})^{-2^k}$ for $k = 2^i,\ i = 0, 1, \dots$ by means of repeated squaring.

Let us supply some results supporting this outline. Write $M^{(i+1)} = (M^{(i)})^2$, $M_i = \frac{1}{2}(M^{(i)} + (M^{(i)})^{-1}) = \frac{1}{2}(I + M^{(i+1)})(M^{(i)})^{-1}$, $\lambda_j^{(i)} = \lambda_j(M^{(i)})$, and $\lambda_{j,i} = \lambda_j(M_i)$ for $j = 1, \dots, n$; $i = 1, 2, \dots$, and observe the following simple facts. The first of them enables the computation of all matrices M_i in real arithmetic where M is a real matrix.

Fact 8.2. $M_i = \frac{1}{2}((M+\sqrt{-1}I)^{2^{i+1}} + (M-\sqrt{-1}I)^{2^{i+1}})(M^2+I)^{-2^i}$ *for* $i = 0, 1, \ldots$ *In particular,* $M_0 = (M^2-I)(M^2+I)^{-1}$, $M_1 = (M^4 - 6M^2 + I)(M^2+I)^{-2}$.

Fact 8.3. *The eigenvalues of the matrices* M_i *are given by* $\frac{1}{2}(\lambda_j^{(i)} + (\lambda_j^{(i)})^{-1}) = \frac{1}{2}((\lambda_j^{(0)})^{2^i} + (\lambda_j^{(0)})^{-2^i})$ *for* $j = 1, \ldots, n$ *and all* i.

Fact 8.4. *For every positive integer* i *the matrix* M_i *shares its eigenspaces with the matrix* M.

Fact 8.5. *For every positive integer* i *the map* $M \to M_i$ *moves the real eigenvalues into the unit disc* $D_1 = \{x : |x| \le 1\}$.

Fact 8.6. *Either* $|\lambda_j^{(i)}| = 1$ *for all* i *and* j *(or equivalently* $M^{(0)H}M^{(0)} = M^{(0)}M^{(0)H} = I$*) or the eigenspace* $\mathbb{S}_{\mathbb{R}}$ *associated with all the real eigenvalues of the matrix* M *is a dominated eigenspace of the matrices* M_i *for all sufficiently large integers* i. *Furthermore the domination increases infinitely as* $i \to \infty$.

Facts 8.2–8.6 together with Theorem 3.1 imply the following corollary.

Corollary 8.1. *Unless* $M^{(0)H}M^{(0)} = M^{(0)}M^{(0)H} = I$ *(or equivalently unless* $|\lambda_j^{(i)}| = 1$ *for all* i *and* j*), the spaces* $\mathcal{R}(K_i^{-1}U_i)$ *converge to the eigenspace* $\mathbb{S}_{\mathbb{R}}$ *provided* r *is the dimension of the eigenspace,* $K_i = M_i + U_iV_i^H$, U_i *and* V_i *are* $n \times r$ *matrices,* K_i *and* $V_i^HU_i$ *are nonsingular and well conditioned matrices, and* $i \to \infty$.

The corollary shows that for large integers i the spaces $\mathcal{R}(K_i^{-1}U)$ closely approximate the eigenspace $\mathbb{S}_{\mathbb{R}}$.

We can obtain the integer r from the Sturm sequence for the characteristic polynomial of the matrix M or by means of binary search based on the recipes for computing the nullity in Section 3 applied to the matrices M_i for sufficiently large integers i.

Suppose that we have computed a matrix B_i (e.g., $B_i = K_i^{-1}U_i$ in Corollary 8.1) whose range $\mathcal{R}(B_i)$ approximates the eigenspace $\mathbb{S}_{\mathbb{R}}$ much closer than the eigenspaces associated with the remaining eigenvalues. Such an approximation is obtained where the number

$$\theta_i = \min_{j:\lambda_j|(M_i)|>1} \max\{|\lambda_j^{(i)}|, 1/|\lambda_j^{(i)}|\} = \theta_0^{2^i}$$

becomes sufficiently large. (Here we assume that not all eigenvalues of the matrix M are real, and so $\theta_0 > 1$.)

We can refine the computed approximation to the eigenspace $\mathbb{S}_{\mathbb{R}}$ by applying the inverse Rayleigh–Ritz (Galerkin) iteration to the matrix M_i; then we can recall Fact 8.4 and deflate the matrix M by decoupling its $r \times r$ block L, whose r eigenvalues are precisely the r real eigenvalues of the matrix M (see the Appendix and [44,19] on the inverse Rayleigh–Ritz (Galerkin) iteration and deflation). Besides decreasing the size of the original problem from n to r, we get rid of all nonreal roots and can reduce the matrix to the rank structured (semiseparable or quasiseparable) form and then apply the structured QR algorithms in [11] or [33]. Hereafter we refer to the above procedure for the approximation of the real eigenvalues as **Algorithm 8.1**.

In our tests for $M = F_{\mathbf{p}}$, however, we observed rapid convergence of the RQ or SQ iterations even where we initialized them near the origin or near the point $\frac{1}{n}\text{trace}(M)$ and applied to the matrices M_0, M_1 and M_2. More precisely we continued the iteration until we satisfied our stopping criterion with the tolerance 10^{-2}. We used the computed eigenvector (shared by the matrices M_i and $F_{\mathbf{p}}$) to initialize the second stage, where we applied the RQ or SQ iteration to the matrix $F_{\mathbf{p}}$ with the tolerance 10^{-6}. Hereafter we refer to this algorithm as **Algorithm 8.2**. Whenever the process converged to a nonreal eigenvalue (this occurred in less than 20% of runs in our tests), we deflated it together with its complex conjugate eigenvalue and reapplied the same algorithm to the deflated matrix of dimension $n-2$.

Estimating the arithmetic cost of the computations in Algorithms 8.1 and 8.2 we can incorporate our estimates for the arithmetic cost of repeated squaring in the two previous sections in both cases where $M = F_{\mathbf{p}}$ is the companion matrix and $M = C$ is a DPR1 matrix. We also note that for $M = F_{\mathbf{p}}$ the computation of each of the matrices $(M \pm \sqrt{-1}I)^{-1}$ takes $4n-1$ ops. (We only need the first column of the inverse [31,32] and compute it by applying Gaussian elimination.) For $M = C$ such computation takes $6n$ ops (see Theorem 7.4), and $M \pm \sqrt{-1}I$ and $(M \pm \sqrt{-1}I)^{-1}$ are DPR1 matrices.

Remark 8.1. We can stop our iteration where the relative residual norms for our approximations are within a fixed tolerance bound. To be sure that the approximated eigenvalues are real, we can choose the tolerance based on the gap estimates for polynomial roots (see [53] and the references therein). To refine the computed approximations and to handle more rare hard inputs, we can shift to the well developed symbolic methods for the isolation of the real roots of a polynomial (see [30] and the references therein).

9. Matrix-free real root-finding

Let us describe a matrix-free variant of the latter approach to real root-finding. Remark 8.1 can be applied to this variant as well.

Algorithm 9.1. *Real root-finding*

INPUT: a small positive tolerance τ, a positive integer n, and $n+1$ real values $p_0, p_1, \dots, p_n, p_n \neq 0$, such that the polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$ has an unknown number r of real roots. (One can shift the variable x to ensure that $p_{n-1} = 0$.)

OUTPUT: the integer r and the approximations to the r real roots of the polynomial $p(x)$ within the required precision.

INITIALIZATION: If $p(\sqrt{-1}) = p(-\sqrt{-1}) = 0$, set $n \leftarrow n-2$ and $p(x) \leftarrow \frac{p(x)}{x^2+1}$. If $p(1) = 0$, set $n \leftarrow n-1$ and $p(x) \leftarrow \frac{p(x)}{x-1}$. Repeat until $p(\sqrt{-1})p(1) \neq 0$.

COMPUTATIONS:

1. Compute the polynomial $p_0(x) = \frac{(x-\sqrt{-1})^n}{p(1)} p\left(1 + \frac{2\sqrt{-1}}{x-\sqrt{-1}}\right)$. ($p_0(x) = \prod_{j=1}^{n}(x-\lambda_j)$ for n unknown roots $\lambda_1, \dots, \lambda_n$. The real roots of the polynomial $p(x)$ are mapped into the roots of the monic polynomial $p_0(x)$ lying on the unit circle $C_1 = \{x : |x| = 1\}$.)
2. Fix a reasonably large integer k and apply k steps of the Dandelin's (Lobachevsky's, Gräffe's) root-squaring iteration $p_{i+1}(x) = (-1)^n p_i(\sqrt{x}) p_i(\sqrt{-x})$, $i = 0, 1, \dots, k$. (We have $p_i(x) = \prod_{j=1}^{n}(x - \lambda_j^{2^i})$, so that the i-th iteration step squares the roots of the polynomial $p_{i-1}(x)$ for all i. The map $p(x) \to p_i(x) + x^n p_i(1/x)$ moves all real roots of the input polynomial $p(x)$ into the disc $D_2 = \{x : |x| \le 2\}$ and for large integers i moves all its other roots far away from this disc.)
3. Having performed k squaring steps, apply the algorithm in [54] (cf. [55,17]) to estimate the root radii of the polynomial $p_{k+1}(x)$, that is the distances $|\lambda_j|^{2^{k+1}}$ of all its roots from the origin. Allow relative errors within a fixed tolerance δ and output the number r of the roots that lie in the annulus $1-\delta \le |\lambda| \le 1+\delta$. (Count the roots with their multiplicities.) If $r = n$, output $v(x) = p(x)$ and stop.
4. Otherwise, as soon as the roots of the polynomial $p_k(x)$ lying on the unit circle C_1 become sufficiently well separated from the other roots, apply [56, Algorithm 2.1] to the polynomial $p_k(x) = \prod_{j=1}^{n}(x-\lambda_j^{(k)})$, which should replace $p(x)$ in [56]. The algorithm outputs polynomial $\widehat{p}(x) = \sum_{i=0}^{n} \widehat{p}_i x^i = \prod_{j=1}^{n}(x - \frac{1}{2}(\lambda_j^{(k)} + 1/\lambda_j^{(k)}))$ whose r absolutely smallest roots lie in the unit disc $D_1 = \{x : |x| \le 1\}$, whereas all other roots lie far from this disc.
5. Apply the algorithms in [54,57,17] to compute an approximate factor $\widehat{v}(x) \approx \prod_{h=1}^{r}(x - z_h^{(k)})$ of degree r sharing these r roots with the polynomial $\widehat{p}(x)$.
6. Apply a selected root-finder to compute the r roots $z_1^{(k)}, \dots, z_r^{(k)}$ of the latter factor.
7. Observe that $\lambda_{j_h}^{(k)}$ for $h = 1, \dots, r$ equals either $z_h^{(k)} + \sqrt{z_h^{(k)} - 1}$ or $1/(z_h^{(k)} + \sqrt{z_h^{(k)} - 1}) = z_h^{(k)} - \sqrt{z_h^{(k)} - 1}$. For every h select one of the two expressions for $\lambda_{j_h}^{(k)}$ for which $p_k(\lambda_{j_h}^{(k)}) = 0$.
8. For $l = k, k-1, \dots, 1$ recursively descend from the r roots $\lambda_{j_1}^{(l)}, \dots, \lambda_{j_r}^{(l)}$ of the polynomial $p_l(x)$ to the r roots $\lambda_{j_1}^{(l-1)}, \dots, \lambda_{j_r}^{(l-1)}$ of the polynomial $p_{l-1}(x)$ by recalling that $(\lambda_{j_h}^{(l-1)})^2 = \lambda_{j_h}^{(l)}$ and $p_{l-1}(\lambda_{j_h}^{(l-1)}) = 0$ for all pairs of h and l. (Cf. [16,4,17].)
9. Compute and output the approximate real roots $\lambda_{j_h} = \sqrt{-1}\frac{\lambda_{j_h}^{(0)}+1}{\lambda_{j_h}^{(0)}-1}$ of the polynomial $p(x)$ for $h = 1, \dots, r$.

Every squaring step as well as the root radii estimation takes $O(n \log n)$ ops, and so do Stage 1 (reduced to two variable shifts and the transition to the reverse polynomial between them (cf. [37, Problem 2.4.3])), Stage 3 (see [54,55,17]), Stage 4 (see [56,58]), and Stage 5 provided that the roots in the unit disc D_1 are well separated from the other roots. Stages 7 and 8 involve $O(rn)$ ops, whereas Stage 9 involves $3r$ ops. The cost of performing Stage 6 is dominated where $r \ll n$, and we can further accelerate the computations at Stage 6 as follows.

(a) Use [56, Eqs. (12)–(14)] to compute the coefficients $q_0, \dots, q_r$ defining the representation $v(x) = \sum_{i=0}^{r} q_i(y^i + y^{-i})$, $x = \frac{y+y^{-1}}{2}$, of the polynomial $\widehat{v}(x)$ with the roots $z_1^{(k)}, \dots, z_r^{(k)}$. The computation can be performed by interpolating to the polynomial $q(y) = y^r v(x)$ from its values at the 2^lth roots of unity $y_j = \exp(2\pi j\sqrt{-1}/2^l)$, $j = 0, 1, \dots, 2^l - 1$, $l = 1 + \lfloor \log_2 2r \rfloor$. More precisely we can compute the values of the polynomial $v(x)$ at the Chebyshev points $x_j = \frac{1}{2}(y_j + y_j^{-1})$ and then multiply these values by y_j^r and y_j^{-r}. With FFT we only need $O(r \log r)$ ops at both stages of evaluation and interpolation [58].

(b) Recall from [56, Section 2] that the polynomial $q(y) = y^r v(x) = \sum_{i=0}^{r} q_i(y^{r+i} + y^{r-i})$ has $2r$ roots $\lambda_{j_h}^{(k)}$ and $1/\lambda_{j_h}^{(k)}$ for $h = 1, \dots, r$. Note that these roots lie on the unit circle C_1. Transform them into real values by applying the substitution $y \to \sqrt{-1}\frac{1+w}{1-w} = -\sqrt{-1}\left(1 + \frac{2}{w-1}\right)$. Then again $O(r \log r)$ ops are sufficient at this stage.

(c) Apply the Laguerre or quasi-Laguerre algorithms in [59–63] to approximate the $2r$ real roots $w_1, \dots, w_{2r}$ of the polynomial $(1-w)^r q(\sqrt{-1}\frac{1+w}{1-w})$ of degree $2r$.

(d) Obtain the $2r$ roots $\lambda_{j_h}^{(k)}$ and $1/\lambda_{j_h}^{(k)}$ for $h = 1, \dots, r$ of the polynomial $q(y)$ by applying the inverse transform $w \to \frac{1+y\sqrt{-1}}{1-y\sqrt{-1}}$.

(e) Continue as at Stages 7–9 of Algorithm 9.1.

This modification of Algorithm 9.1 is said to be **Algorithm 9.2**. The main benefit of using it is the application of the Laguerre or quasi-Laguerre root-finders, which are proved to be highly effective where all roots are real. These proofs can be extended to the case where all roots lie on the unit circle C_1, and we can compress Steps (b)–(d) above into the direct application of the respective extension of the Laguerre or quasi-Laguerre algorithm to the polynomial $q(y)$. This modification of Algorithm 9.1 is said to be **Algorithm 9.3**.

Algorithms 9.1–9.3 can face numerical problems at Stage 2 because the required computational precision rapidly increases in root squaring, due to the uneven growth of the absolute values of the polynomial coefficients. One can safely perform a squaring step numerically by using the order of n^2 ops provided the computation of a logarithm as well as an exponential is also counted as an op [64], although computations with extended precision would still be required at Stage 3.

We can, however, reuse the remedy from the previous section, that is we can stop Stage 2 at a smaller integer k, say at $k \le 2$, and instead of performing Stage 5 seek the roots of the polynomial $\widehat{v}(x)$ by applying to the polynomial $\widehat{p}(x)$ Müller's or Newton's iteration initiated near the origin. We can expect that it converges to a root of the polynomial $\widehat{p}(x)$ lying in the unit disc D_1 because such roots tend to be closest to the origin among all roots. Having approximated such a root $z^{(k)}$ of the polynomial $\widehat{p}(x)$, we proceed as in Stages 7–9 to approximate the respective root λ of the input polynomial $p(x)$ and output it if this is a real root. Otherwise, we would have $\lambda = r + s\sqrt{-1}$ for real r and $s \neq 0$, and then we would deflate the polynomial $p(x)$ by dividing it by $x^2 - 2rx + r^2 + s^2$ and would reapply the algorithm to the quotient polynomial. This modification of Algorithm 9.1 is said to be **Algorithm 9.4**.

Correctness verification for Algorithms 9.1–9.4 is rather straightforward, and we omit it.

10. Numerical factorization of a polynomial

Factorization $p(x) = u(x)v(x)$ of a polynomial $p(x)$ of a degree n into the product of two factors $u(x) = \sum_{i=0}^{k} u_i x^i$ and $v(x) = \sum_{i=0}^{l} v_i x^i$ of degrees k and $l = n - k$, respectively, served as a basis for effective root-finding algorithms in [54,16, 17], but it also represents deflation, that is polynomial division with no remainder and is of independent interest due to its applications to the time series analysis, Weiner filtering, noise variance estimation, covariance matrix computation, and the study of multi-channel systems [65–70].

The factorization can be equivalently expressed by any of the two following vector equations (cf., e.g., [73,74]),

$$C_l(\mathbf{u})\mathbf{v} = \mathbf{p} \tag{10.1}$$

or

$$C_k(\mathbf{v})\mathbf{u} = \mathbf{p}. \tag{10.2}$$

Here $\mathbf{u} = (u_i)_{i=0}^{k}$, $\mathbf{v} = (v_i)_{i=0}^{l}$, and $\mathbf{p} = (p_i)_{i=0}^{n}$ are the coefficient vectors of the polynomials $u(x)$, $v(x)$, and $p(x)$, respectively, whereas $C_l(\mathbf{u}) \in \mathbb{C}^{(n+1)\times(l+1)}$ and $C_k(\mathbf{v}) \in \mathbb{C}^{(n+1)\times(k+1)}$ are the convolution matrices associated with the product $u(x)v(x)$. They are lower trapezoidal Toeplitz matrices (with all the superdiagonal and subdiagonal entries zero) defined by their first columns $C_l(\mathbf{u})\mathbf{e}_1 = (\mathbf{u}^T, \mathbf{0}^T)^T$ and $C_k(\mathbf{v})\mathbf{e}_1 = (\mathbf{v}^T, \mathbf{0}^T)^T$, respectively. Eqs. (10.1) and (10.2) provide an equivalent representation via structured linear system of equations. We assume that the polynomial $u(x)$ is monic, so that $u_0 = 1$, $v_0 = p_0$.

Now suppose we are given approximate factors $u_0(x) \approx u(x)$ (monic) and $v_0(x) \approx v(x)$ and wish to refine them. We can write $r_0(x) = p(x) - u_0(x)v_0(x)$ or equivalently $\mathbf{r}_0 = \mathbf{p} - C_l(\mathbf{u}_0)\mathbf{v}_0 = \mathbf{p} - C_v(\mathbf{v}_0)\mathbf{u}_0$ and define a fixed point iteration with Newton's updates as follows, $\begin{pmatrix}\bar{\mathbf{u}}_{i+1}\\ \mathbf{v}_{i+1}\end{pmatrix} = \begin{pmatrix}\bar{\mathbf{u}}_i\\ \mathbf{v}_i\end{pmatrix} + \Delta_i$. Here $\mathbf{u}_h$ and $\mathbf{v}_h$ for all h denote the coefficient vectors of the polynomials $u_h(x)$ (monic) and $v_h(x)$, each of the vectors $\bar{\mathbf{u}}_h$ is obtained by deleting the first (unit) coordinate of the vector $\mathbf{u}_h$, $-J_i\Delta_i = \mathbf{r}_i, J_i = -(C(\mathbf{v}_i), C_l(\mathbf{u}_i)) \in \mathbb{C}^{(n+1)\times(n+1)}$ are the Jacobians for $i = 0, 1, \dots$, and the matrix $C(\mathbf{v}_i) \in \mathbb{C}^{(n+1)\times k}$ is obtained by deleting the first column from the matrix $C_k(\mathbf{v}_i) \in \mathbb{C}^{(n+1)\times(k+1)}$. (We delete the column because $u_0 = 1$ is not a variable in the expression $\mathbf{r}_0 = \mathbf{p} - C_v(\mathbf{v}_0)\mathbf{u}_0$.)

Clearly the resulting Newton iteration algorithm (we refer to it as **Algorithm 10.1**) has local quadratic convergence. Its ith step is essentially the solution of a linear system of equations with the $(n+1) \times (n+1)$ Sylvester matrix $-J_i$.

If we are given just a single approximate factor $u_0(x)$, we can initialize the Newton process by computing the coefficient vector of the second factor $\mathbf{v}_0$ as an approximate solution of the overdetermined linear system (10.1) of n equations with l unknowns. We can compute this approximate solution as the solution of the lower (resp. upper) triangular Toeplitz linear system formed by the l first (resp. last) equations of the system, but its least squares solution generally gives a little better fitting. We can obtain such a solution from the normal Hermitian linear system of l equations $(C_l(\mathbf{u}_0))^H C_l(\mathbf{u}_0)\mathbf{v}_0 = (C_l(\mathbf{u}_0))^H\mathbf{p}$, whose Toeplitz matrix has the lower and upper bandwidth l [71]. The algorithm in [72, Section 2.14] reduces the

solution of such a system to the solution of a lower triangular Toeplitz linear system whose $n \times n$ matrix has a bandwidth at most $2l$ and of a $k \times k$ Toeplitz linear system. We refer to the resulting algorithm for numerical deflation (or equivalently for polynomial division with no remainder) as **Algorithm 10.2**.

Instead of Newton's process we can define iteration by alternating the application of Algorithm 10.2 to updating the factors $u_i(x)$ and $v_i(x)$ recursively, so that a sequence of approximations $v_0(x)$, $u_1(x)$, $v_1(x)$, $u_2(x)$, ... to the factors $u(x)$ and $v(x)$ is recursively computed as least squares solutions of the linear systems (10.1) and (10.2) with alternating inputs $\mathbf{u} = \mathbf{u}_i$, $\mathbf{v} = \mathbf{v}_i$ for $i = 0, 1, \ldots$. We refer to this factorization algorithm as Algorithm 10.3.

For every integer i, $i = 0, 1, \ldots$ we can refine an approximate factor $u_i(x)$ (resp. $v_i(x)$) of a polynomial $p(x)$ as an approximate gcd $g(x)$ of the two polynomials $u_i(x)$ (resp. $v_i(x)$) and $ap(x)$ where a is a scalar having a large absolute value $|a|$. This choice should suppress the perturbation of the polynomial $p(x)$, so that up to scaling the approximate gcd $g(x)$ would be close to a divisor of the gcd, and we can readily compute it from a subresultant matrix (cf., e.g., [73, Section 6.10], [37, Section 2.10], [74]).

We can apply such a refinement of both factors $u_i(x)$ and $v_i(x)$ at every ith iteration step of Algorithm 10.3 or only at some selected steps according to a fixed policy. In both cases we refer to this refined version of Algorithm 10.3 as Algorithm 10.4.

Furthermore one can alternate the steps of Algorithms 10.1–10.4 to enhance the power of the refinement of the initial factorization and ensure fast convergence to the factors $u(x)$ and $v(x)$.

One can apply the above algorithms to the approximation of a single root λ (resp. a pair of roots λ_1 and λ_2) by choosing its (their) possibly crude initial approximation $\tilde{\lambda}$ (resp. approximations $\tilde{\lambda}_1$ and $\tilde{\lambda}_2$) and setting $u_0(x) = x - \tilde{\lambda}$ (resp. $u_0(x) = (x - \tilde{\lambda}_1)(x - \tilde{\lambda}_2)$), but in our tests we consistently observed substantially faster convergence of the RQ iteration versus this factorization algorithm in the case of $k = 1$ (that is for approximating a single root for the same input polynomials). Perhaps the greater power of the RQ iteration comes because, unlike the factorization algorithm, it approximates eigenpairs of the associated matrix (rather than just its eigenvalues). If, however, the task is the refinement of a crude initial approximation and if both integers k and l are not small (that is if we wish to split a polynomial into two factors of larger degrees), the factorization algorithm may become superior because it has simpler iteration steps.

11. Numerical tests

We performed a series of numerical experiments in the Graduate Center of the City University of New York using a Dell server with a dual core 1.86 GHz Xeon processor and 2 G memory running Windows Server 2003 R2. The test Fortran code was compiled with the GNU gfortran compiler within the Cygwin environment.

We generated random numbers with the random_number intrinsic Fortran function assuming the uniform probability distribution over the range $\{x : 0 \le x < 1\}$. To shift to the range $\{y : b \le y \le a + b\}$ for fixed real a and b, we applied the linear transform $x \to y = ax + b$.

We tested our algorithms in Sections 4, 5 and 8 for the approximation of real and complex eigenvalues of random general matrices (only in Table 3), random companion matrices $F_{\mathbf{p}}$ (defined by random vectors $\mathbf{p}$), and random DPR1 matrices (defined by random vectors $\mathbf{s}$ and $\mathbf{d} = \mathbf{v}$ for $\mathbf{u} = (1)_{i=1}^{n}$), of sizes $n \times n$ for $n = 64, 128, 256$. For each input size and each iterative algorithm we generated 100 input instances and run 100 tests. Our tables show the minimum, maximum, and average (mean) numbers of iteration loops until convergence in these runs as well as the standard deviations in the columns marked by "**min**", "**max**", "**mean**", and "**std**", respectively.

For the initialization of the RQ and SQ iterations we used equally spaced points on the unit circle $\{x : |x| = 1\}$ or "large" circle $\{x : |x| = a\|M\|\}$ for the input matrices M and for $a = 2$, except for testing convergence to distinct eigenvalues (see Tables 18 and 19) where we set $a = 10$.

Tables 3 and 4 display the data on the approximation of the complex eigenvalues by the RQ and SQ iterations in (4.1)–(4.6) and the AP iteration in Section 4, respectively, assuming the initial values λ_0 chosen at random on a large circle and using the tolerance $\tau = 10^{-6}$ in the stopping criterion (4.4). For testing the AP iteration (in both RQ and SQ versions) we generated APs $\mathbf{u}_i^T\mathbf{v}_i$ for $\mathbf{u}_i = \mathbf{y}_{i-1}$ and $\mathbf{v}_i = \mathbf{e}_n$ for all integers i, except that we have chosen the simplifying APs for $i = 0$, that is the APs $\mathbf{p}\mathbf{e}_n^T$ or $(\mathbf{p} + \mathbf{e}_1 + \mu\mathbf{e}_n)\mathbf{e}_n^T$ in the case of companion matrices $F_{\mathbf{p}}$ and the APs $\mathbf{u}\mathbf{v}^H$ in the case of DPR1 matrices C.

Tables 5–8 display the test results where three eigen-solving algorithms were combined for the companion matrices $F_{\mathbf{p}}$ and DPR1 matrices C. At Stage 1 we set tolerance τ to 10^{-1} in (4.4) and applied the RQ version of the AP iteration with the simplifying APs. Then at Stage 2 we set tolerance to 10^{-4} and refined the computed crude approximations by applying the RQ iteration with no preprocessing. Finally at Stage 3 we set tolerance to 10^{-6} and applied Algorithms 5.1 or 5.3 where again we used the simplifying APs. All tables display the respective numbers of steps at every stage, and Tables 7 and 8 also show the percent (number) of the cases of divergence in 100 tests.

Tables 9–14 display the results of our tests of the real eigen-solving Algorithm 8.2 in Section 8 assuming the companion input matrix and using 0, 1, and 2 squarings. Tables 9, 11 and 13 show the number of iteration loops at Stage 1, at which the iteration stopped as soon as the error decreased below the tolerance 10^{-2}. Tables 10, 12 and 14 cover Stage 2 of refinement of these computed approximations until they decreased the error below the tolerance 10^{-6}. The rightmost columns in Tables 10, 12 and 14 display the numbers of real roots computed in 100 test runs.

Tables 15–17 display the results of our tests for initialization via continuous scaling. Table 15 shows the number of iterations at Stage 1 where we chose the initial approximate eigenvalues on a large circle, applied the AP iteration with simplifying APs to the companion and DPR1 matrices associated with the polynomial $p(1.02x)$, and set the output tolerance to 10^{-1}.

Tables 16 and 17 show the numbers of iterations at Stages 2 and 3 where we applied the RQ version of Algorithm 5.1 for $\nu = 1$ and with simplifying APs for all integers i to the same polynomial $p(1.02x)$ and to the polynomial $p(x)$, respectively, and set the output tolerance to 10^{-6} in both cases. We initialized Stages 2 and 3 by using the eigenvalue approximations output in the preceding stage.

Tables 18–20 display the numbers of iteration loops in our tests for approximating distinct eigenvalues of the companion and DPR1 matrices M associated with a random polynomial $p(x)$ of degree n, with the reverse polynomial $p_{\text{rev}}(x) = x^n p(1/x)$, or with both of them, as we specify in the first column of each table. The two last columns of each table show the percents

Table 3
Numbers of RQ and SQ iteration loops in algorithms (4.1)–(4.6).

Iteration	Matrix	n	Min	Max	Mean	Std
RQ	Companion	64	4.00	12.00	6.10	1.65
RQ	Companion	128	4.00	11.00	6.21	1.48
RQ	Companion	256	4.00	13.00	6.18	1.50
SQ	Companion	64	4.00	16.00	7.75	2.27
SQ	Companion	128	5.00	17.00	8.37	2.49
SQ	Companion	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

Table 4
Numbers of RQ and SQ iteration loops in the AP iteration.

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

Table 5
Numbers of RQ loops at Stage 1 (the AP iteration with the simplifying APs) in the combination of three eigen-solving algorithms.

Matrix	n	Mean	Std
Companion	64	3	0
Companion	128	3	0
Companion	256	3	0
DPR1	64	2.18	0.48
DPR1	128	2.04	0.50
DPR1	256	2.04	0.38

Table 6
Numbers of RQ loops at Stage 2 (the RQ iteration) in the combination of three eigen-solving algorithms.

Matrix	n	Mean	Std
Companion	64	4.69	0.65
Companion	128	4.86	0.43
Companion	256	4.78	0.57
DPR1	64	4.86	0.48
DPR1	128	4.88	0.39
DPR1	256	4.90	0.30

Table 7
Numbers of RQ loops at Stage 3 (Algorithm 5.1 with the simplifying APs) in the combination of three eigen-solving algorithms.

Matrix	n	Mean	Std	% of divergent tests
Companion	64	3.16	6.02	3
Companion	128	5.55	12.42	6
Companion	256	3.79	3.25	6
DPR1	64	1.71	1.97	6
DPR1	128	2.28	2.52	3
DPR1	256	2.16	2.33	2

Table 8
Numbers of RQ loops at Stage 3 (Algorithm 5.3 with the simplifying APs) in the combination of three eigen-solving algorithms.

Matrix	n	Mean	Std	% of divergent tests
Companion	64	3.00	4.39	2
Companion	128	4.22	5.58	7
Companion	256	5.87	12.29	6
DPR1	64	2.06	2.56	1
DPR1	128	1.85	2.18	2
DPR1	256	1.72	2.39	5

Table 9
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (no squaring, Stage 1).

Iteration	n	Min	Max	Mean	Std
RQ	64	2	17	3.86	2.19
RQ	128	2	15	4.12	2.27
RQ	256	2	15	3.72	1.76
SQ	64	2	18	4.22	2.76
SQ	128	2	45	5.18	5.68
SQ	256	2	33	4.57	4.02

Table 10
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (no squaring, Stage 2).

Iteration	n	Min	Max	Mean	Std	% of real roots
RQ	64	0	2	0.99	0.32	90
RQ	128	0	2	0.9	0.43	89
RQ	256	0	2	0.82	0.42	82
SQ	64	0	2	1	0.37	91
SQ	128	0	2	0.97	0.38	91
SQ	256	0	2	0.88	0.45	90

of distinct eigenvalues computed in our tests among all the n eigenvalues. In these tests we applied the RQ and SQ iterations to the matrices M at $3n \log_2 n$ initial equally spaced points on the selected circles, namely $\{x : |x| = 3\|M\|\}$ in Table 18, $\{x : |x| = 3\|M\|\}$ in Table 19, and Bini's circles in Table 20.

Table 21 shows the average numbers of iteration loops per eigenvalue in our tests where we applied RQ iteration with recursive deflation to DPR1 matrices.

Table 11
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (one squaring, Stage 1).

Iteration	n	Min	Max	Mean	Std
RQ	64	2	9	3.9	1.18
RQ	128	2	6	3.75	0.86
RQ	256	2	13	3.7	1.4
SQ	64	2	10	4.23	1.5
SQ	128	2	9	4.19	1.29
SQ	256	2	8	4.25	1.35

Table 12
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (one squaring, Stage 2).

Iteration	n	Min	Max	Mean	Std	% of real roots
RQ	64	0	17	2.44	2.27	89
RQ	128	0	6	2.39	1.28	89
RQ	256	0	14	2.51	1.79	83
SQ	64	0	14	2.3	2.01	86
SQ	128	0	10	2.46	1.81	90
SQ	256	0	12	2.9	2.32	83

Table 13
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (two squarings, Stage 1).

Iteration	n	Min	Max	Mean	Std
RQ	64	2	12	4.01	1.53
RQ	128	2	9	3.89	1.12
RQ	256	3	10	4.05	1.23
SQ	64	2	24	4.07	2.39
SQ	128	3	9	3.92	1.12
SQ	256	3	10	4.07	1.27

Table 14
Numbers of RQ and SQ iteration loops in Algorithm 8.2 (two squarings, Stage 2).

Iteration	n	Min	Max	Mean	Std	% of real roots
RQ	64	0	4	1.51	0.72	91
RQ	128	0	6	1.71	1	91
RQ	256	0	8	2.06	1.43	88
SQ	64	0	4	1.62	0.79	90
SQ	128	0	16	2.33	2.49	90
SQ	256	0	12	2.27	1.88	83

Table 15
Numbers of RQ iteration loops in the algorithm with simplifying APs and continuous scaling (Stage 1: the AP iteration, $t = 1.02$).

Matrix	n	Mean	Std
Companion	64	2	0
Companion	128	2	0
Companion	256	1.9	0.10
DPR1	64	1.94	0.24
DPR1	128	1.96	0.20
DPR1	256	1.97	0.17

Table 16
Numbers of RQ iteration loops in the algorithm with simplifying APs and continuous scaling (Stage 2: Algorithm 5.1, $t = 1.02$).

Matrix	n	Mean	Std
Companion	64	5.91	2.72
Companion	128	6.38	3.68
Companion	256	6.80	9.80
DPR1	64	5.55	0.24
DPR1	128	5.79	2.97
DPR1	256	6.20	3.61

Table 17
Numbers of RQ iteration loops in the algorithm with simplifying APs and continuous scaling (Stage 3: Algorithm 5.1, $t = 1$).

Matrix	n	Mean	Std
Companion	64	2.72	1.01
Companion	128	2.78	1.17
Companion	256	3.07	1.50
DPR1	64	2.50	0.87
DPR1	128	2.64	1.23
DPR1	256	2.66	1.03

Table 18
Percent of distinct computed roots with $3n \log_2 n$ initial eigenvalues on the unit circle.

Polynomials	Iteration	Matrix	n	Mean	Std
$p(x)$	RQ	Companion	32	73.16	10.53
$p(x)$	RQ	Companion	64	67.58	8.24
$p(x)$	SQ	Companion	32	91.00	6.75
$p(x)$	SQ	Companion	64	91.67	5.64
$p(x)$	RQ	DPR1	32	96.50	3.30
$p(x)$	RQ	DPR1	64	95.56	2.57
$p(x)$	SQ	DPR1	32	90.63	7.37
$p(x)$	SQ	DPR1	64	91.22	5.85
$p_{\text{rev}}(x)$	RQ	Companion	32	75.06	8.44
$p_{\text{rev}}(x)$	RQ	Companion	64	74.78	5.89
$p_{\text{rev}}(x)$	SQ	Companion	32	95.06	4.89
$p_{\text{rev}}(x)$	SQ	Companion	64	97.19	2.08
$p_{\text{rev}}(x)$	RQ	DPR1	32	97.47	4.21
$p_{\text{rev}}(x)$	RQ	DPR1	64	98.33	1.82
$p_{\text{rev}}(x)$	SQ	DPR1	32	95.38	4.34
$p_{\text{rev}}(x)$	SQ	DPR1	64	97.27	2.38
Both	RQ	Companion	32	94.56	5.99
Both	RQ	Companion	64	93.84	5.07
Both	SQ	Companion	32	99.00	2.03
Both	SQ	Companion	64	99.38	1.11
Both	RQ	DPR1	32	98.25	2.14
Both	RQ	DPR1	64	98.56	1.63
Both	SQ	DPR1	32	98.91	2.32
Both	SQ	DPR1	64	99.20	1.36

Table 19
Percent of distinct computed roots with $3n \log_2 n$ initial eigenvalues on a large circle.

Polynomials	Iteration	Matrix	n	Mean	Std
$p(x)$	RQ	Companion	32	4.63	2.45
$p(x)$	RQ	Companion	64	2.20	1.00
$p(x)$	SQ	Companion	32	11.72	6.16
$p(x)$	SQ	Companion	64	7.81	3.28
$p(x)$	RQ	DPR1	32	7.84	6.92
$p(x)$	RQ	DPR1	64	5.02	7.21
$p(x)$	SQ	DPR1	32	84.72	13.09
$p(x)$	SQ	DPR1	64	79.39	14.25
$p_{\text{rev}}(x)$	RQ	Companion	32	4.97	6.48
$p_{\text{rev}}(x)$	RQ	Companion	64	2.80	3.29
$p_{\text{rev}}(x)$	SQ	Companion	32	18.31	12.14
$p_{\text{rev}}(x)$	SQ	Companion	64	15.70	9.77
$p_{\text{rev}}(x)$	RQ	DPR1	32	7.78	7.71
$p_{\text{rev}}(x)$	RQ	DPR1	64	4.16	7.13
$p_{\text{rev}}(x)$	SQ	DPR1	32	70.28	34.50
$p_{\text{rev}}(x)$	SQ	DPR1	64	54.44	36.43
Both	RQ	Companion	32	5.81	6.97
Both	RQ	Companion	64	3.16	3.08
Both	SQ	Companion	32	20.63	12.69
Both	SQ	Companion	64	16.97	8.84
Both	RQ	DPR1	32	14.28	10.62
Both	RQ	DPR1	64	8.50	11.39
Both	SQ	DPR1	32	93.41	11.20
Both	SQ	DPR1	64	88.42	13.52

Table 20
Percent of distinct computed roots with $3n \log_2 n$ initial eigenvalues on Bini's circle.

Polynomials	Iteration	Matrix	n	Mean	Std
$p(x)$	RQ	Companion	32	24.03	7.62
$p(x)$	RQ	Companion	64	22.31	4.35
$p(x)$	SQ	Companion	32	33.66	7.84
$p(x)$	SQ	Companion	64	34.03	4.68
$p(x)$	RQ	DPR1	32	47.31	9.66
$p(x)$	RQ	DPR1	64	44.33	7.53
$p(x)$	SQ	DPR1	32	47.91	7.31
$p(x)$	SQ	DPR1	64	44.31	5.51
$p_{\text{rev}}(x)$	RQ	Companion	32	20.41	7.79
$p_{\text{rev}}(x)$	RQ	Companion	64	22.42	5.79
$p_{\text{rev}}(x)$	SQ	Companion	32	32.66	7.85
$p_{\text{rev}}(x)$	SQ	Companion	64	35.38	5.95
$p_{\text{rev}}(x)$	RQ	DPR1	32	53.97	7.43
$p_{\text{rev}}(x)$	RQ	DPR1	64	54.08	5.35
$p_{\text{rev}}(x)$	SQ	DPR1	32	50.13	9.58
$p_{\text{rev}}(x)$	SQ	DPR1	64	43.70	6.71
Both	RQ	Companion	32	40.03	8.98
Both	RQ	Companion	64	40.75	6.06
Both	SQ	Companion	32	51.56	7.94
Both	SQ	Companion	64	52.73	6.04
Both	RQ	DPR1	32	86.53	7.93
Both	RQ	DPR1	64	84.69	6.77
Both	SQ	DPR1	32	68.78	9.86
Both	SQ	DPR1	64	61.31	7.32

Table 21
Numbers of RQ iteration loops per eigenvalue of a DPR1 matrix using deflation.

Matrix	n	Mean	Std
DPR1	64	7.85	2.00
DPR1	128	7.65	1.87
DPR1	256	7.98	2.05

12. Discussion

We covered a number of approaches to complex and real root-finding and eigen-solving and polynomial factorization. Can we enhance their power by alternating their steps and possibly the steps of some known iterative root-finders? If so, what is the best policy of such an alternation? Further refinement of the algorithms is another natural challenge. The greatest promise comes from the recursive improvement of the DPR1 matrices in Section 7.6, which can be based on the RQ, SQ, SMW, AP iterations in Section 4, Algorithms 5.1–5.3, 10.1, 10.2, or iterative algorithms in [3]. Here are some sample directions to promising modifications.

(a) Convergence of our iterative algorithms applied to a DPR1 matrix associated with a given polynomial as well as convergence rate depend on the choice of the parameters s_i and d_i or s_i, u_i, and v_i, $i = 1, 2, \dots, n$, that define the matrix. How can we optimize the choice of these parameters?

(b) For separation of real eigenvalues one can modify the expressions in Fact 8.2, shift and scale the input matrix M to have its trace vanished, or move all its eigenvalues into a small circle near one or -1 (keeping the real eigenvalues real) and then apply our techniques in Section 8 to the resulting matrix. By using this recursive process one can incorporate more squarings overall under a fixed bound on the matrix norms.

(c) The initialization policies are highly important for various aspects of convergence, including its rate and avoiding convergence to the same eigenvalues from distinct initial points. Currently these policies are essentially heuristic. Experiments with various classes of input polynomials, concurrent choices of the matrices associated with the same polynomial, and various homotopy continuation processes may suggest further improvements.

(d) Improvement of global convergence to complex roots could possibly come from alternating the steps of our DPR1 eigen-solving with eigen-free root-finding (e.g., based on Newton's, Müller's, Börsch–Supan's, or Weierstrass' iterations, or other iterative algorithms in [3]) (cf. [10,9]).

(e) Another recipe for yielding convergence in the case of hard inputs is to employ the approximation of eigenspaces of small dimensions (rather than just eigenvectors), based on the Rayleigh–Ritz (Galerkin) procedure (see Remark 4.1 and Appendix A.4). The latter procedure also enables eigen-solving deflation, which is more efficient than the known methods for splitting polynomials into factors, but destroys the matrix structure, so that one should only apply it on a limited scale.

(f) A number of the customary eigen-solving techniques such as the subspace and Jacobi–Davidson iterations as well as Rayleigh–Ritz (Galerkin) procedure and Arnoldi and non-Hermitian Lanczos algorithms with restarting (cf. [44,19]) incorporate the shift-and-invert techniques and could benefit from incorporating our modifications of these techniques.

(g) Successful DPR1 eigen-solving could prompt effort for the reduction to it of eigen-solving for nonderogatory matrices.

(h) One can try to combine additive preprocessing and Newton's linearization for the solution of a polynomial systems of equations (see Appendix D).

Appendix A. Deflation/extraction techniques and Rayleigh–Ritz (Galerkin) procedure

A.1. Deflation/extraction techniques

For a pair of nonsingular matrices $W = (B, C)$ and $W^{-1} = \begin{pmatrix} B_{\text{left}}^H \\ C_{\text{left}}^H \end{pmatrix}$ such that

$$B_{\text{left}}^H C = 0, \qquad C_{\text{left}}^H B = 0, \qquad B_{\text{left}}^H B = I, \qquad C_{\text{left}}^H C = I, \tag{A.1}$$

suppose

$$C_{\text{left}}^H MB = 0, \qquad L = B_{\text{left}}^H MB, \qquad H = C_{\text{left}}^H MC, \tag{A.2}$$

so that the matrix

$$W^{-1}MW = \begin{pmatrix} L & G \\ O & H \end{pmatrix}, \tag{A.3}$$

is block triangular [19, Section 4.1]. (Note that $C_{\text{left}}^H MB = 0$ if $MB = 0$ or $C_{\text{left}}^H M = 0$.) Then, clearly, an eigenpair $(\bar{L}, \bar{B})$ (resp. $(\bar{H}, \bar{C})$) of the matrix L (resp. H) defines an eigenpair $(\bar{L}, B\bar{B})$ (resp. $(\bar{H}, C\bar{C})$) of the matrix M. The following converse result is also easy to verify [19, Theorem 4.4.1].

Theorem A.1. *Assume that $\mathbb{X}$ is an eigenspace of a matrix M and let B, B_{left}, and L be three matrices such that $B_{\text{left}}^H B = I$, $L = B_{\text{left}}^H MB$, and $\mathbb{X} \in \mathcal{R}(B)$. Then (a) $(\bar{L}, B\bar{B})$ is an eigenpair of the matrix M if $(\bar{L}, \bar{B})$ is an eigenpair $(\bar{L}, \bar{B})$ of the matrix L and (b) there exists an eigenpair $(\bar{L}, \bar{B})$ of the matrix L such that $\mathbb{X} = \mathcal{R}(B\bar{B})$.*

The above reduction of eigen-solving for a matrix M to eigen-solving for two matrices L and H of smaller sizes is called *deflation* or decoupling (see some alternative deflation techniques in [20, pages 584–602] and [75, Section IV.2]). Eq. (A.1)–(A.3) and Theorem A.1 also support *extraction of an eigenspace* $\mathbb{X}$ of M.

A.2. Orthogonal and structured deflation/extraction

We can rely on Eqs. (A.1) and (A.2) for any left inverse B_{left} of a matrix B, but if B is a unitary matrix, then we can choose $B_{\text{left}} = B$ and compute the RQ matrix

$$L = B^H MB. \tag{A.4}$$

Likewise if C is a unitary matrix, then we can choose $C_{\text{left}} = C$ and compute the RQ matrix

$$H = C^H MC. \tag{A.5}$$

With nonunitary matrices B and C one would face numerical problems in the deflation and extraction in Appendix A.1 but can yield the matrices W with desired structures.

A.3. Recursive deflation

Assume a structured matrix M and suppose we extend deflation based on matrix Eq. (A.3) by employing some nonsingular matrices W_L and W_H, such that the matrices $W_L^{-1}LW_L$ and $W_H^{-1}HW_H$ are 2×2 block triangular. Then the matrix $V = \text{diag}(W_L^{-1}, W_H^{-1})W^{-1}MW\,\text{diag}(W_L, W_H)$ is 4×4 block triangular. By choosing the matrices W, W_L, and W_H with appropriate structures, we can yield structure also for the 4×4 block matrix V and its blocks, although generally in a little deteriorated form. The same comments can be extended recursively. Quantitatively the input structure can be maintained and utilized in a small number of recursive deflation steps but is likely to be completely lost already in $O(\log n)$ steps. Generally the latter problem cannot be fixed because the transformation matrices S and Q in the eigendecomposition $MS = S\Lambda$ and in the Schur triangulation $M = Q^H TQ$ (for a unitary matrix Q and a triangular matrix T) are generally unstructured. The same comments apply to other popular and effective recipes of eigenspace extraction (cf. [76]).

We can stay with structured computation of all eigenpairs by working with the original matrix M where we use no or a limited number of deflations, e.g., where we decouple all the real roots of a polynomial by means of the recipes in Section 8 or apply the RQ or SQ iterations concurrently at a large number of the initial points.

In the two special cases in Sections 6 and 7 the left and right eigenvectors do form structured matrices (namely Vandermonde and Cauchy matrices and their inverses, respectively), but employing these structures for the acceleration of eigen-solving remains a research challenge. In these two cases, however, we only seek eigenvalues (roots). So we can deflate the associated companion and DPR1 matrices in linear time and continue again with matrices of the same class of a smaller size.

A.4. Rayleigh–Ritz (Galerkin) procedure

Suppose we have a matrix B whose range contains an approximation to an eigenspace of a matrix M, the left inverse B^H_{left}, the matrix $L = B^H_{\text{left}}MB$, and its eigenpair $(\bar{L}, \bar{B})$. Then Theorem A.1 implies that an eigenpair of the matrix M is approximated by the pair $(\bar{L}, B\bar{B})$, whose computation is called *Rayleigh–Ritz* (*Galerkin*) *procedure*. The auxiliary matrices L and $B\bar{B}$ are called *Ritz blocks* and *Ritz bases*, respectively, and the pairs $(L, B\bar{B})$ are *Ritz pairs*. The scalar Ritz blocks are called *Ritz values*; they are associated with *Ritz vectors* $B\bar{\mathbf{b}}$ [19, Section 4.4.1], [44].

The approximation errors of the procedure can be bounded in terms of the norm $\|C^H_{\text{left}}MB\|$ for the matrix C_{left} in Appendix A.1 (cf. [19, Section 4.4.2]), and the procedure is numerically stable if this norm is small.

The Ritz values μ_j for j in a fixed subset $J \subseteq \{1, \dots, n\}$ are the eigenvalues of the matrix L (which approximate the eigenvalues of the matrix M). We can compute them by computing the Schur decomposition of the matrix L, which is more stable numerically than its eigendecomposition. Given these values we can obtain the so called *refined Ritz vectors* $\mathbf{y}_j$, being the solutions of the minimization problem

$$\begin{aligned}&\text{minimize } \|M\mathbf{y}_j - \mu_j\mathbf{y}_j\|\\ &\text{subject to } \mathbf{y}_j \in \mathcal{R}(B),\ \|\mathbf{y}_j\| = 1 \quad \text{for } j \in J\end{aligned}$$

[19, Section 4.4.3]. For μ_j lying near the eigenvalues, the vector $\mathbf{y}_j$ lies near a null vector of the matrix $M - \mu_j I$, and so incorporation of our algorithms in Sections 4 and 5 can be effective.

A popular alternative is the *harmonic Ritz vectors* $\mathbf{y}$ [77, Section 3.2], [19, Section 4.4.4], [44] obtained by solving the generalized eigenproblem $V^HM^HMV\mathbf{y} = \lambda V^HM^HV\mathbf{y}$ for a given approximate matrix basis V for an eigenspace. By orthogonalizing the matrix MV we arrive at the standard eigenproblem $\frac{1}{\lambda}\mathbf{y} = V^HM^HV\mathbf{y}$. The vectors $\mathbf{y}$ converge to the null space of the matrix $V^HM^HMV - \lambda V^HM^HV$ as the values λ converge to an eigenvalue of M, and again we can employ our algorithms in Sections 4 and 5. Having harmonic Ritz vector $\mathbf{y}$ approximated, we can approximate the associated eigenvalue by the RQ $\frac{\mathbf{y}^HV^HMV\mathbf{y}}{\mathbf{y}^HV^HV\mathbf{y}}$ or the SQ $\frac{\mathbf{e}_j^HMV\mathbf{y}}{\mathbf{e}_j^HV\mathbf{y}}$ where $\mathbf{e}_j^HV\mathbf{y} \neq 0$.

Appendix B. Subspace iteration, extensions and expansions

B.1. Subspace iteration

The *Subspace iteration* $B_{i+1} = MB_i,\ i = 0, 1, \dots$, is defined for an $n \times \nu$ matrix B_0 and for $\nu < n$ (typically for $\nu \ll n$). It turns into the Power iteration $\mathbf{y}_{i+1} = M\mathbf{y}_i/\|M\mathbf{y}_i\|,\ i = 0, 1, \dots$, for $\nu = 1$ and $B_0 = \mathbf{y}_0$. In virtue of the next theorem, $\mathcal{R}(B_i) \to \mathbb{S}_{\{1,\dots,\nu\}}$ as $i \to \infty$ for generic $n \times \nu$ matrix B_0, that is the Subspace iteration simultaneously converges to all dominant ones among the eigenspaces $\mathbb{S}_{\{1,\dots,j\}}$ for $j = 1, \dots, \nu$.

Theorem B.1 (*[19, Theorem 6.1.1]*). *Suppose* $M = \sum_{i=1}^{3} X_iL_iY_i^H$ *is a spectral representation where* $(X_1, X_2, X_3)(Y_1, Y_2, Y_3)^H = I_n$, L_i *are* $l_i \times l_i$ *matrices,* $\Lambda(L_i) = \text{diag}(\lambda_j)_{j=h_{i-1}+1}^{h_i},\ i = 1, 2, 3, h_0 = 0, h_1 = l_1, h_2 = l_1 + l_2, h_3 = n, \lambda_j = \lambda_j(M)$, *inequalities (2.1) hold,* $|\lambda_{l_1}| > |\lambda_{l_1+1}|$, *and* $|\lambda_{l_1+l_2}| > |\lambda_{l_1+l_2+1}|$. *Suppose* $B_0 = X_1C_1 + X_2C_2 + X_3C_3$, $C_i = Y_i^HB_0$ *for* $i = 1, 2, 3$, *the matrix* (C_1^H, C_2^H) *is nonsingular, and* θ_k *is the largest canonical angle between the linear spaces* $\mathcal{R}(M^kB_0)$ *and* $\mathcal{R}(X_1)$. *Then* $\theta_k = O((\epsilon + |\lambda_{l_1+l_2+1}/\lambda_{l_1}|)^k)$ *for any positive* ϵ.

One can relax the assumption that $|\lambda_{l_1+l_2}| > |\lambda_{l_1+l_2+1}|$ by shifting to the matrices $\text{diag}(M, 0)$ if the matrix M is nonsingular or $\text{diag}(M - \gamma I, 0)$ for a small $|\gamma|$ otherwise.

Theorem B.1 implies fast convergence of the iteration to the dominant eigenspaces $\mathbb{S}_{\{1,\dots,j\}}$ for a small integer j, so that the matrix bases B_i tend to become close to some smaller rank matrices generating this dominant subspace as i grows large. We can avoid the resulting numerical problems by means of periodic orthogonalization of the bases B_i (clearly this does not affect convergence to the eigenspaces) and deflation (extraction) of the dominant eigenspaces, which can rely on the Rayleigh–Ritz (Galerkin) procedure in [75,19,44]. Extraction is followed by locking those parts of the bases B_i whose ranges become nearly invariant in multiplication by M. We summarize this algorithm below and refer the reader to [75, Algorithm 5.4] and [44, Section 7.4, Algorithm 7.2] on its analysis, further details and variations.

Algorithm B.1. *Subspace iteration with eigenspace extraction and locking.*

INPUT: three positive integers COUNTER, n, and ν, $\nu \le n$, an $n \times n$ matrix M, and a positive tolerance τ.

OUTPUT: a nonnegative integer $k \le \nu$ and k approximate eigenpairs $(\lambda_j, \mathbf{x}_j)$ of the matrix M, $j = 1, \dots, k$ such that

$$\|M\mathbf{x}_j - \lambda_j \mathbf{x}_j\| \le \tau |\lambda_j| \|\mathbf{x}_j\|. \tag{B.1}$$

INITIALIZATION: Fix a positive integer iter (usually in the range from 3 to 5) and an $n \times \nu$ matrix $Y = (\mathbf{y}_j)_{j=1}^{\nu}$. Set $\bar{i} \longleftarrow 0$, $k \leftarrow 0$ and denote a pair of $n \times k$ empty matrices by X and Λ.

COMPUTATIONS: Recursively, for $i = 1, 2, \dots$ perform the following steps.

1. Compute $n \times (\nu - k)$ matrix $M^{\text{iter}} Y$.
2. Compute the $n \times (\nu - k)$ matrix $Q(X, MY) = (X, B)$, keeping the block X of the first k columns intact.
3. Compute the Rayleigh quotient $L = B^H MB$.
4. Apply the Rayleigh–Ritz (Galerkin) procedure in Appendix A.4 (with orthogonal projections) to the matrices B and M, that is, compute the Schur decomposition $L = Z^H T Z$ where $Z = (\mathbf{z}_j)_{j=1}^{\nu-k}$, $Z^H Z = I_{\nu-k}$, and T is an upper triangular matrix with the diagonal given by the matrix $\text{diag}(\lambda_j)_{j=1}^{\nu-k}$.
5. Compute the matrix $Y = BZ = (\mathbf{y}_j)_{j=1}^{\nu-k}$.
6. Test the relative residuals for convergence for $j = 1, \dots, \nu - k$: if inequality (B.1) holds for the vector $\mathbf{y}_j$ replacing $\mathbf{x}_j$, then delete this column vector from the matrix Y, append it to the matrix X, and extend the diagonal matrix Λ of eigenvalues by appending the new diagonal entry λ_j. Keep denoting the resulting updated matrices by X, Y, and Λ. Let the updated matrix X have the size $n \times k'$. Then set $k \leftarrow k'$.
7. If $k = \nu$ or if $\bar{i} > \mathit{COUNTER}$, stop and output the integer k and the eigenpairs $(\lambda_j, \mathbf{x}_j)$, $j = 1, \dots, k$. Otherwise set $\bar{i} \leftarrow \bar{i} + i$, update the positive integer iter, and go to Stage 1.

Remark B.1. By employing the eigendecomposition of the matrix L (instead of its Schur decomposition), one can simplify Stage 4 (cf. [75, page 157], [44, Section 7.4, Algorithm 4.5]) at the expense of some deterioration of numerical stability of the computations.

Remark B.2. The algorithm uses the standard stopping criterion (B.1), which can be generalized to the bound

$$\|MX - X\Lambda\| \le \tau \|B\Lambda\| \tag{B.2}$$

for a candidate approximate eigenpair (Λ, X). Such bounds on the residual norm are readily verified, although they only guarantee that (Λ, X) is an eigenpair of a nearby matrix but may have no eigenpairs of the matrix M nearby.

Remark B.3. Assume a companion or generalized companion matrix M in Algorithm B.1. Then in principle we can modify its Stage 5 of computing the matrix Y (that defines the eigenspace). We can correct the equation $Y = BZ$ by taking into account Fact 6.1 and Theorem 7.2, respectively. By relying entirely on Fact 6.1 and Theorem 7.2 we would yield a matrix Y with a structure of Vandermonde or Cauchy type, respectively, but could hardly ensure even good local convergence. Is it possible to ensure convergence for a structured matrix Y and if so, can we extend the structure to the matrices H and L in (A.1)–(A.3)?

B.2. The inverse iteration and Rayleigh–Ritz (Galerkin) procedure

The subspace iteration approximates the dominant eigenspaces $\mathbb{S}_{\{1,\dots,j\}}$ for $j \le \nu$. The *Inverse iteration* redirects the process to approximating the eigenspaces $\mathbb{S}_K$ for a set of the eigenvalues lying near a fixed set $\Lambda^{(0)} = \{\lambda_1^{(0)}, \dots, \lambda_\nu^{(0)}\}$ on the complex plane. We just need to apply the Subspace iteration to the matrix $r(M)$ for $r(x) = 1/\prod_{j=1}^{\nu}(x - \lambda_j^{(0)})$. For a singleton $K = \{\lambda_h^{(0)}\}$ for a fixed integer h, $1 \le h \le n$, this is the RQ iteration in Section 4. Our previous study can be extended assuming the eigenvalues $\lambda_j(r(M)) = r(\lambda_j(M))$ enumerated in the nonincreasing order of their absolute values.

The iteration and its analysis can be extended to any rational function $r(x)$ for which the matrix $r(M)$ is defined. Most popular is the extension where this function $r_i(x) = 1/\prod_{j=1}^{\nu}(x - \lambda_j^{(i)})$ is modified in the ith iteration step, which updates the approximations $\lambda_j^{(i)}$ to the eigenvalues. These updates are by-products of the Rayleigh–Ritz (Galerkin) procedure applied for the subspace extraction, and we call the resulting algorithm the *Inverse Rayleigh–Ritz* (*Galerkin*) *iteration* [75], [44,19, Section 7.4].

B.3. Expanding subspace iterations

The Subspace and Inverse iterations can be modified so that the dimension of the subspace can vary dynamically. e.g., one begins with a subspace defined by a single vector, but the iteration expands the subspace by including new vectors to its basis (with periodic orthogonalization) until convergence to a single eigenvector or to an eigenspace of a dimension bounded by a fixed ν or until further subspace extension becomes too costly. In the latter case one can restart the process (explicitly or implicitly). Based on the current subspace information, one can utilize the progress achieved by the cutoff time.

This flowchart is implemented in the Jacobi–Davidson algorithm [76], [19, Section 6.2], [44, Section 7.12] and in the Krylov Sequence (Krylov Subspace) processes such as Arnoldi and non-Hermitian Lanczos iterations, highly effective for large sparse input matrices [15, Chapter 9], [19, Chapter 5], [44, Sections 7.5–7.11], [78, Chapter 9].

The kth stage of the iteration involves multiplication of the matrix M (resp. the matrices M and M^H) by some vector $\mathbf{w}$ in Arnoldi (resp. vectors $\mathbf{u}$ and $\mathbf{w}$ in non-Hermitian Lanczos) algorithm and additional $O(kn)$ (resp. $O(n)$) ops and units of memory for storage. (The latter ops and need for storage space can be a hurdle if Arnoldi algorithm is applied to a structured matrix M.) Instead of the vectors $M\mathbf{w}$ (resp. vectors $M\mathbf{u}$ and $M^H\mathbf{w}$) one computes the vectors $(\mu I - M)^{-1}\mathbf{w}$ (resp. the vectors $(\mu I - M)^{-1}\mathbf{u}$ and $(\mu I - M^H)^{-1}\ \mathbf{w}$) in the shift-and-invert version of the algorithm. The non-Hermitian Lanczos algorithm outputs both left and right eigenspaces and uses less ops and memory space than Arnoldi's but has greater risk of bad breakdown and numerical instability.

The Jacobi–Davidson algorithm applies the following modification of steps (4.1) of the RQ and SQ iterations, $(I-\mathbf{y}_i\mathbf{y}_i^H)(M-\lambda^{(i)}I)(I-\mathbf{y}_i\mathbf{y}_i^H)(\mathbf{y}_{i+1}-\mathbf{y}_i) = \lambda^{(i)}\mathbf{y}_i - M\mathbf{y}_i,\ i = 0, 1, \ldots$ Here the shift value $\lambda^{(i)}$ is defined by the Rayleigh–Ritz (Galerkin) procedure applied to the matrix M and the current correction space generated by the initial approximation $\mathbf{y}_0$ and all correction vectors $\mathbf{y}_{j+1} - \mathbf{y}_j,\ j = 0, 1, \ldots, i-1$.

The shift values employed in all these algorithms can be computed approximately, and the linear systems defining approximate null vectors of the shifted matrix can be solved by means of the algorithms in Sections 4 and 5.

The Jacobi–Davidson algorithm supersedes the subspace iteration in practice for approximating a small number of eigenvalues (at the extreme of the spectrum or near the shift) together with their associated eigenspaces, but its global convergence (with and without restarting) is not well understood theoretically unless it approximates the eigenvalues that are well separated from the other eigenvalues.

B.4. GR and QR iterations

The GR iteration begins with setting $M = M_0$. Its ith step

$$M^{(i)} = G^{(i)}R^{(i)}, \qquad M^{(i+1)} = R^{(i)}G^{(i)} = (G^{(i)})^{-1}M^{(i)}G^{(i)}, \quad i = 0, 1, \ldots \tag{B.3}$$

computes and interchanges the GR factors of the current iterate M_i for an upper triangular matrix R and a nonsingular matrix G of a fixed form, e.g., the GR factors are the QR or PLU factors. Each iteration step can be viewed as the space iteration step, applied to the whole space $\mathcal{R}(I_n)$ of dimension n and followed by moving the updated space back to the space $\mathcal{R}(I)$ [78, page 158]. Assume generic input matrix M with the eigenvalues $\lambda_1, \ldots, \lambda_n$ such that $|\lambda_1| < |\lambda_2| < \cdots < |\lambda_n|$. Then the iteration produces matrices M_i converging to a triangular matrix, and the cumulative transform matrices

$$\widehat{G}^{(i)} = G^{(1)}G^{(2)}\cdots G^{(i)} \tag{B.4}$$

are well conditioned. More precisely, we have the following result [78, Theorem 5.2.3].

Theorem B.2. *Assume an $n \times n$ matrix $M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$ with $k \times k$ leading block M_{00} and the eigenvalues λ_j ordered to satisfy (2.1). Also assume the inequality*

$$\theta_k = |\lambda_{k+1}/\lambda_k| < 1. \tag{B.5}$$

Let the two eigenspaces of M associated with λ_j, for $j = 1, \ldots, k$ and for $j = k+1, \ldots, n$ have the only common vector $\mathbf{0}$. Then $\|M_{10}^{(i)}\| \le c(\theta+\epsilon)^i \mathrm{cond}(\widehat{G}^{(i)})$ for the matrix $\widehat{G}^{(i)}$ in (B.4), any positive ϵ, and some constant c.

We can request that the matrices $G^{(i)} = Q^{(i)}$ be unitary for all i, thus ensuring that $\mathrm{cond}(\widehat{G}^{(i)}) = 1$ for all i. This defines the QR celebrated algorithm [15, Section 7.5], [19, Chapter 2], [44, Section 7.3]. It is customary to apply it to matrices $M = M^{(0)}$ reduced to Hessenberg form (in $O(n^3)$ ops). Then every iteration step takes $O(n^2)$ ops, which also cover the cost of bulge chasing that recovers the Hessenberg form for every computed matrix $M^{(i)}$. Multishifts by the scalars computed via Rayleigh–Ritz (Galerkin) procedure dramatically accelerate convergence. They modify Eq. (B.3) as follows,

$$r_i(M^{(i)}) = G^{(i)}R^{(i)}, \qquad M^{(i+1)} = R^{(i)}G^{(i)} = (G^{(i)})^{-1}M^{(i)}G^{(i)}, \quad i = 0, 1, \ldots, \tag{B.6}$$

where $r_i(x) = \prod_{j=1}^{\nu}(x - \lambda_j^{(i)})$ and the vector $\Lambda^{(i)} = (\lambda_j^{(i)})_{j=1}^{\nu}$ of scalar shifts $\lambda_j^{(i)}$ is updated at the ith step based on Rayleigh–Ritz (Galerkin) procedure. The actual implementation includes the policies of deflation and explicit or implicit shifting.

Appendix C. Matrix iterations for root-finding

One can try to accelerate global convergence of iterative matrix algorithms for root-finding by applying various advanced eigen-solvers, such as the Subspace iteration and the Inverse Rayleigh–Ritz (Galerkin) iteration in the previous section or the non-Hermitian Lanczos, Arnoldi and Jacobi–Davidson algorithms, but this can only be advisable in the case of hard inputs for which the iterative algorithms in Sections 6 and 7 stumble or diverge. Otherwise the ops count tends to favor the SQ iteration. In particular, convergence of the Subspace and the Inverse Rayleigh–Ritz (Galerkin) iterations is accelerated with the increase of the dimension ν of the basic subspace, but so does the arithmetic cost per step as well. For example, we need the order of $2\nu^2 n$ ops for orthogonalization of the basis and about as many ops for computing RQs, which is a substantial cost increase even for $\nu = 2$ and even if we simplify the iteration by weakening its numerical stability [75, page 157], [44, Section 7.4, Algorithm 4.5]. Deflation in Appendix A becomes substantially more costly as the matrices B, C, H, and L in (A.1)–(A.3) grow in size and lose structure. The structure is destroyed in orthogonalization and in the matrix transition $B, C \to H, L$. One can yield structured nonunitary matrices B and C based on Fact 6.1 and Theorem 7.2, but this leads to convergence problems (see Remark B.3).

Other directions to potential convergence acceleration include combining eigen-solving approach with Newton's, Müller's, Aberth's, Durand–Kerner's, and other polynomial root-finders (cf. [9]) and various heuristics for computing good initial approximations μ.

Appendix D. Solving a polynomial system of equations

Consider a system of two quadratic polynomials equations with two variables x and y,

$$p(x, y) = p_{0,0} + p_{0,1}x + p_{1,0}y + p_{0,2}x^2 + p_{1,1}xy + p_{2,0}y^2 = 0, \tag{D.1}$$

$$q(x, y) = q_{0,0} + q_{0,1}x + q_{1,0}y + q_{0,2}x^2 + q_{1,1}xy + q_{2,0}y^2 = 0. \tag{D.2}$$

Define the resultant equation $R(x, y)\mathbf{z}(x, y) = \mathbf{0}$ for the vector $\mathbf{z}(x, y) = (1, x, y, x^2, xy, y^2)^T$ and the following 6×7 resultant matrix of a rank at least five,

$$R(x, y) = \begin{pmatrix} -x & 1 & & & & & \\ -y & & 1 & & & & \\ & -x & & 1 & & & \\ & & -x & & 1 & & \\ & & -y & & & 1 & \\ p_{0,0} & p_{0,1} & p_{1,0} & p_{0,2} & p_{1,1} & p_{2,0} & \\ q_{0,0} & q_{0,1} & q_{1,0} & q_{0,2} & q_{1,1} & q_{2,0} & \end{pmatrix}.$$

This matrix has a null vector $\mathbf{z}(x, y)$ if and only if the pair (x, y) satisfies the system of Eqs. (D.1) and (D.2). The same property holds for a number of variations of the matrix. e.g., we can replace its fourth row vector $(0, 0, -x, 0, 1, 0)$ with $(0, -y, 0, 0, 1, 0)$. We can remove any of the first five rows still preserving the resultant property of the matrix, although generally not the lower bound of five on its rank. This bound is preserved, however, where $p_{0,2}q_{0,2} \neq 0$ and we remove the third row, where $p_{1,1}q_{1,1} \neq 0$ and we remove the fourth row, as well as where $p_{2,0}q_{2,0} \neq 0$ and we remove the fifth row.

Now let a pair (x_0, y_0) approximate a solution pair $(\tilde{x}, \tilde{y})$ to the polynomial system above, such that $R(\tilde{x}, \tilde{y})\mathbf{z}(\tilde{x}, \tilde{y}) = \mathbf{0}$, and combine additive preprocessing with Newton's linearization to generate a sequence of new approximations (x_i, y_i), $i = 1, 2, \ldots$.

Recursively define pairs of properly scaled random vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ and write

$$R_i = R(x_i, y_i), \qquad C_i = R_i + \mathbf{u}_i\mathbf{v}_i^H, \qquad \mathbf{z}_i = \mathbf{z}(x_i, y_i),$$

$$\delta\mathbf{z}_i = (0, \delta x_i, \delta y_i, 2(\delta x_i)x_i, (\delta x_i)y_i + (\delta y_i)x_i, 2(\delta y_i)y_i)^T,$$

$$\delta C_i = C_{i+1} - C_i = R_{i+1} - R_i = \begin{pmatrix} -\delta x_i & & & & & \\ -\delta y_i & & & & & \\ & -\delta x_i & & & & \\ & & -\delta x_i & & & \\ & & -\delta y_i & & & \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

for the scalars $\delta x_i = x_{i+1} - x_i$ and $\delta y_i = y_{i+1} - y_i$ defined from the following linear system of equations, $C_i\mathbf{z}_i + (\delta C_i)\mathbf{z}_i + C_i(\delta\mathbf{z}_i) = (\mathbf{v}_{i+1}^H\mathbf{z}_i)\mathbf{u}_{i+1} + (\mathbf{v}_{i+1}^H(\delta\mathbf{z}_i))\mathbf{u}_{i+1}$, $i = 0, 1, \ldots$. Due to randomization we can expect that $\mathbf{v}^H\mathbf{z}(\tilde{x}, \tilde{y}) \neq 0$, $\mathbf{v}_i^H\mathbf{z}_i \neq 0$ for all i, and that the matrices $C(\tilde{x}, \tilde{y}) = R(\tilde{x}, \tilde{y}) + \mathbf{u}_i\mathbf{v}_i^H$ and C_i for all i have full rank (cf. [79–81]).

We obtain this linear system in δx_i and δy_i by ignoring the terms of higher orders in $\delta_i = \max\{|\delta x_i|, |\delta y_i|\}$ in the polynomial system of equations $C_{i+1}\mathbf{z}_{i+1} = (\mathbf{v}_{i+1}^H\mathbf{z}_{i+1})\mathbf{u}_{i+1}$, which extends the polynomial system $C(\tilde{x}, \tilde{y})\mathbf{z}(\tilde{x}, \tilde{y}) = (\mathbf{v}_{i+1}^H\mathbf{z}(\tilde{x}, \tilde{y}))\mathbf{u}_{i+1}$ implied by Theorem 3.1 (for $C(\tilde{x}, \tilde{y}) = R(\tilde{x}, \tilde{y}) + \mathbf{u}_{i+1}\mathbf{v}_{i+1}^H$), and we readily observe that $\delta\mathbf{z}_i = \mathbf{z}_{i+1} - \mathbf{z}_i + O(\delta_i^2)$.

By setting $\mathbf{u}_i = \mathbf{0}$ for all i, we arrive at a Newton-like extension of the Inverse Iteration for eigen-solving, but the option of varying the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ for all i gives us some additional power for devising effective algorithms. The matrix $R(x, y)$ is structured (it can be multiplied by a vector in nearly linear time [82,83]). In typical applications to algebraic and geometric computations this matrix is also sparse. We can choose the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ to have such properties for the matrix $C(x, y)$ as well. If so, we can effectively solve the linear systems with this matrix by applying the Conjugate Gradient algorithms provided the matrix is well conditioned under our preprocessing.

In all cases Newton's linearization implies local quadratic convergence, although for the iteration in its present form convergence can be readily destroyed by rounding errors.

One can extrapolate this demonstration to polynomial systems with any number of variables, equations and terms and to resultant matrices with any positive nullity, associated with multiple roots of systems of polynomials. Furthermore we can modify our approach by using augmentation instead of additive preprocessing.

References

[1] E.T. Bell, The Development of Mathematics, McGraw-Hill, New York, 1940.
[2] C.A. Boyer, A History of Mathematics, Wiley, New York, 1968.
[3] J.M. McNamee, Numerical Methods for Roots of Polynomials, Part 1, in: Studies in Computational Math., vol. 14, Elsevier Science, Cambridge, MA, 2007.
[4] V.Y. Pan, Solving a polynomial equation: some history and recent progress, SIAM Rev. 39 (2) (1997) 187–220.
[5] V.Y. Pan, Some recent algebraic/numerical algorithms, in: Electronic Proceedings of IMACS/ACA'98, 1998. Available at: http://www.troja.fjfi.cvut.cz/aca98/sessions/approximate/pan.
[6] V.Y. Pan, Solving polynomials with computers, Am. Sci. 86 (1998) 62–69.
[7] V.Y. Pan, Numerical computation of a polynomial GCD and extensions, Inform. and Comput. 167 (2) (2001) 71–85 (Proc. version: Approximate polynomial GCDs, Padé approximation, polynomial zeros, and bipartite graphs, in; Proc. 9th Ann. ACM–SIAM Symp. on Discrete Algorithms, SODA'98, ACM Press, New York, and SIAM Publications, Philadelphia, 1998, 68–77).
[8] V.Y. Pan, D. Ivolgin, B. Murphy, R.E. Rosholt, Y. Tang, X. Wang, X. Yan, Root-finding with eigen-solving, in: Dongming Wang, Lihong Zhi (Eds.), Symbolic–Numeric Computation, Birkhäuser, Basel, Boston, 2007, pp. 185–210.
[9] S. Fortune, An iterated eigenvalue algorithm for approximating roots of univariate polynomials, J. Symbolic Comput. 33 (5) (2002) 627–646 (Proc. version in: Proc. Intern. Symp. on Symbolic and Algebraic Computation, ISSAC'01, ACM Press, New York, 2001, 121–128).
[10] F. Malek, R. Vaillancourt, Polynomial zerofinding iterative matrix algorithms, Comput. Math. Appl. 29 (1) (1995) 1–13.
[11] 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).
[12] G.H. Golub, Some modified matrix eigenvalue problems, SIAM Rev. 15 (1973) 318–334.
[13] A. Melman, A unifying convergence analysis of second-order methods for secular equations, Math. Comp. 66 (1997) 333–344.
[14] D.A. Bini, G. Fiorentino, Design, analysis, and implementation of a multiprecision polynomial rootfinder, Numer. Algorithms 23 (2000) 127–173.
[15] G.H. Golub, C.F. Van Loan, Matrix Computations, 3rd ed., The Johns Hopkins University Press, Baltimore, Maryland, 1996.
[16] V.Y. Pan, Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros, in: Proc. 27th Ann. ACM Symp. on Theory of Computing, ACM Press, New York, 1995, pp. 741–750.
[17] V.Y. Pan, Univariate polynomials: nearly optimal algorithms for factorization and rootfinding, J. Symbolic Comput. 33 (5) (2002) 701–733 (Proc. version in: Proc. International Symp. on Symbolic and Algebraic Computation, ISSAC'01, ACM Press, New York, 2001, 253–267).
[18] B.N. Parlett, The Symmetric Eigenvalue Problem, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.
[19] G.W. Stewart, Matrix Algorithms, Volume II: Eigensystems, SIAM, Philadelphia, 1998.
[20] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, England, 1965.
[21] H. Unger, Nichtlinear Behandlung von eigenwetaufgaben, ZAMM Z. Angew. Math. Mech. 30 (1950) 281–282.
[22] G. Peters, J.H. Wilkinson, Inverse iteration, ill-conditioned equations and Newton's method, SIAM Rev. 21 (1979) 339–360.
[23] D.A. Bini, L. Gemignani, V.Y. Pan, Inverse power and Durand/Kerner iteration for univariate polynomial root-finding, Comput. Math. Appl. 47 (2–3) (2004) 447–459 (Also Technical Report TR 2002 020, CUNY Ph.D. Program in Computer Science, Graduate Center, City University of New York, 2002).
[24] V.Y. Pan, G. Qian, A. Zheng, Z. Chen, Matrix computations and polynomial root-finding with preprocessing, Linear Algebra Appl. 434 (2011) 854–879.
[25] V.Y. Pan, A. Zheng, Real and complex polynomial root-finding with eigen-solving and preprocessing, in: Proc. International Symp. on Symbolic and Algebraic Computation, ISSAC 2010, ACM Press, New York, 2010, pp. 219–226.
[26] Amirhossein Amiraslani, P. Lancaster, Rayleigh quotient algorithms for nonsymmetric matrix pencils, Numer. Algorithms 51 (1) (2009) 5–22.
[27] D.A. Bini, P. Boito, Y. Eidelman, L. Gemignani, I. Gohberg, A fast implicit QR algorithm for companion matrices, Linear Algebra Appl. 432 (2010) 2006–2031.
[28] D.A. Bini, Y. Eidelman, L. Gemignani, I. Gohberg, Fast QR eigenvalue algorithms for Hessenberg matrices which are rank-one perturbations of unitary matrices, SIAM J. Matrix Anal. Appl. 29 (2) (2007) 566–585.
[29] I.Z. Emiris, A. Galligo, E. Tsigaridas, Random polynomials and expected complexity of bisection method for real solving, in: Proc. Intern. Symp. on Symbolic and Algebraic Computation, ISSAC'2010, ACM Press, New York, 2010, pp. 235–242.
[30] M. Hemmer, E.P. Tsigaridas, Z. Zafeirakopoulos, I.Z. Emiris, M.I. Karavelas, B. Mourrain, Experimental evaluation and cross-benchmarking of univariate real solvers, in: Hiroshi Kai, Hiroshi Sekigawa (Eds.), Proc. International Symposium on Symbolic–Numerical Computations, Kyoto, Japan, August 2009, ACM Press, New York, 2009, pp. 105–113.
[31] J.P. Cardinal, On two iterative methods for approximating the roots of a polynomial, in: Lectures in Applied Mathematics, vol. 32, AMS, Providence, RI, 1996, pp. 165–188.
[32] V.Y. Pan, Amended DSeSC power method for polynomial root-finding, Comput. Math. Appl. 49 (9–10) (2005) 1515–1524.
[33] R. Vandebril, M. Van Barel, N. Mastronardi, Matrix Computations and Semiseparable Matrices: Eigenvalue and Singular Value Methods, vol. 2, The Johns Hopkins University Press, Baltimore, Maryland, 2008.
[34] N.J. Higham, Accuracy and Stability in Numerical Analysis, second ed., SIAM, Philadelphia, 2002.
[35] G.W. Stewart, Matrix Algorithms, Vol I: Basic Decompositions, SIAM, Philadelphia, 1998.
[36] R. Vandebril, M. Van Barel, N. Mastronardi, Matrix Computations and Semiseparable Matrices: Linear Systems, vol. 1, The Johns Hopkins University Press, Baltimore, Maryland, 2007.
[37] V.Y. Pan, Structured Matrices and Polynomials: Unified Superfast Algorithms, Birkhäuser, Boston and Springer-Verlag, New York, 2001.
[38] M.J. Quinn, Parallel Computing: Theory and Practice, McGraw-Hill, New York, 1994.
[39] R.A. Sweet, Parallel and vector variant of the cyclic reduction algorithm, SIAM J. Sci. Stat. Comput. 9 (4) (1988) 761–765.
[40] V.Y. Pan, G. Qian, Randomized preprocessing of homogeneous linear systems, Linear Algebra Appl. 432 (2010) 3272–3318.
[41] 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 (Proceedings version in: Proc. of the Third International Computer Science Symposium in Russia, CSR 2008, Lecture Notes in Computer Science (LNCS), Springer, Berlin, vol. 5010, 2008, 372–383).

[42] V.Y. Pan, D. Grady, B. Murphy, G. Qian, R.E. Rosholt, A. Ruslanov, Schur aggregation for linear systems and determinants, in: D.A. Bini, V.Y. Pan, J. Verschelde (Eds.), Symbolic–Numerical Algorithms, in: Theoretical Computer Science, vol. 409, 2008, pp. 255–268 (special issue).
[43] J. Habbard, D. Schleicher, S. Sutherland, How to find all roots of complex polynomials by Newton's method, Invent. Math. 146 (2001) 1–33.
[44] 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.
[45] V.Y. Pan, X. Yan, Additive preconditioning, eigenspaces, and the inverse iteration, Linear Algebra Appl. 430 (2009) 186–203 (Proc. version in: SNC 2007).
[46] J.F. Jónsson, S. Vavasis, Solving polynomials with small leading coefficients, SIAM J. Matrix Anal. Appl. 26 (2) (2004) 400–412.
[47] R.M. Corless, On a generalized companion matrix pencil for matrix polynomials expressed in the Lagrange basis, in: Dongming Wang, Lihong Zhi (Eds.), Symbolic–Numeric Computation, Birkhäuser, Basel, Boston, 2007, pp. 1–15.
[48] D.A. Bini, Numerical computation of polynomial zeros by means of Aberth's method, Numer. Algorithms 13 (1996) 179–200.
[49] M.-H. Kim, S. Sutherland, Polynomial root-finding algorithms and branched covers, SIAM J. Comput. 23 (1994) 415–436.
[50] A.L. Toom, The complexity of a scheme of functional elements realizing the multiplication of integers, Sov. Math. Dokl. 3 (1963) 714–716.
[51] C. Carstensen, Linear construction of companion matrices, Linear Algebra Appl. 149 (1991) 191–214.
[52] V.Y. Pan, Root squaring with DPR1 matrices, in: N.N. Vasiliev, A.M. Vershik (Eds.), Zapiski Nauchnykh Seminarov POMI, vol. 373, 2009, pp. 189–193 (in English).
[53] I.Z. Emiris, B. Mourrain, E. Tsigaridas, The DMM bound: multivariate (aggregate) separation bound, in: Proc. Intern. Symp. on Symbolic and Algebraic Computation, ISSAC'2010, ACM Press, New York, 2010, pp. 243–250.
[54] A. Schönhage, The fundamental theorem of algebra in terms of computational complexity, Mathematics Department, University of Tübingen, Germany, 1982.
[55] V.Y. Pan, Approximating complex polynomial zeros: modified quadtree (Weyl's) construction and improved Newton's iteration, J. Complexity 16 (1) (2000) 213–264.
[56] D. Bini, V.Y. Pan, Graeffe's, Chebyshev, and Cardinal's processes for splitting a polynomial into factors, J. Complexity 12 (1996) 492–511.
[57] P. Kirrinnis, Polynomial factorization and partial fraction decomposition by simultaneous Newton's iteration, J. Complexity 14 (1998) 378–444.
[58] V.Y. Pan, New fast algorithms for polynomial interpolation and evaluation on the Chebyshev node set, Comput. Math. Appl. 35 (3) (1998) 125–129.
[59] B. Parlett, Laguerre's method applied to the matrix eigenvalue problem, Math. Comp. 18 (1964) 464–485.
[60] E. Hansen, M. Patrick, J. Rusnack, Some modification of Laguerre's method, BIT 17 (1977) 409–417.
[61] Q. Du, M. Jin, T.Y. Li, Z. Zeng, Quasi-Laguerre iteration in solving symmetric tridiagonal eigenvalue problems, SIAM J. Sci. Comput. 17 (6) (1996) 1347–1368.
[62] Q. Du, M. Jin, T.Y. Li, Z. Zeng, The quasi-Laguerre iteration, Math. Comp. 66 (217) (1997) 345–361.
[63] X. Zou, Analysis of the quasi-Laguerre method, Numer. Math. 82 (1999) 491–519.
[64] G. Malajovich, J.P. Zubelli, On the geometry of Graeffe iteration, J. Complexity 17 (3) (2001) 541–573.
[65] G.T. Wilson, Factorization of the covariance generating function of a pure moving-average process, SIAM J. Numer. Anal. 6 (1969) 1–7.
[66] G.E.P. Box, G.M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, San Francisco, CA, 1976.
[67] S. Barnett, Polynomial and Linear Control Systems, Marcel Dekker, New York, 1983.
[68] C.J. Demeure, C.T. Mullis, The Euclid algorithm and the fast computation of cross-covariance and autocovariance sequences, IEEE Trans. Acoust. Speech Signal Process. 37 (1989) 545–552.
[69] C.J. Demeure, C.T. Mullis, A Newton–Raphson method for moving-average spectral factorization using the Euclid algorithm, IEEE Trans. Acoust. Speech Signal Process. 38 (1990) 1697–1709.
[70] P.M. Van Dooren, Some numerical challenges in control theory, in: Linear Algebra for Control Theory, in: IMA Vol. Math. Appl., vol. 62, Springer, 1994.
[71] R.M. Corless, P.M. Gianni, B.M. Trager, S.M. Watt, The singular value decomposition for polynomail systems, in: Proc. Intern. Symposium on Symbolic and Algebriac Computation, ISSAC'95, ACM Press, New York, 1995, pp. 195–207.
[72] D. Bini, V.Y. Pan, Polynomial and Matrix Computations, Volume 1: Fundamental Algorithms, Birkhäuser, Boston, 1994.
[73] J. von zur Gathen, J. Gerhard, Modern Computer Algebra, second ed., Cambridge University Press, Cambridge, UK, 2003.
[74] D.A. Bini, P. Boito, A fast algorithm for approximate polynomial GCD based on structured matrix computations, in: Operator Theory: Advances and Applications, vol. 199, Birkhäuser Verlag, 2010, pp. 155–173.
[75] Y. Saad, Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms, John Wiley, New York, 1992.
[76] G.L.G. Sleijpen, H.A. Van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problem, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425 Reproduced in SIAM Rev. 42 (2) (2000) 276–293.
[77] Y. Saad, An introduction to iterative projection methods: basic ideas, Section 3.2 in [44].
[78] D.S. Watkins, The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods, SIAM, Philadelphia, 2007.
[79] R.A. Demillo, R.J. Lipton, A probabilistic remark on algebraic program testing, Inform. Process. Lett. 7 (4) (1978) 193–195.
[80] J.T. Schwartz, Fast probabilistic algorithms for verification of polynomial identities, J. ACM 27 (4) (1980) 701–717.
[81] R.E. Zippel, Probabilistic algorithms for sparse polynomials, in: Proceedings of EUROSAM'79, in: Lecture Notes in Computer Science, vol. 72, Springer, Berlin, 1979, pp. 216–226.
[82] I.Z. Emiris, V.Y. Pan, Symbolic and numerical methods for exploiting structure in constructing resultant matrices, J. Symbolic Comput. 33 (2002) 393–413.
[83] B. Mourrain, V.Y. Pan, Multivariate polynomials, duality and structured matrices, J. Complexity 16 (1) (2000) 110–180 (Proceedings version in: STOC'98).