SIAM J. MATRIX ANAL. APPL.
Vol. 42, No. 3, pp. 1148–1171

RANK-STRUCTURED QR FOR CHEBYSHEV ROOTFINDING*

ANGELO CASULLI† AND LEONARDO ROBOL‡

Abstract. We consider the computation of roots of polynomials expressed in the Chebyshev basis. We extend the QR iteration presented in [Y. Eidelman, L. Gemignani, and I. Gohberg, *Numer. Algorithms*, 47 (2008), pp. 253–273] introducing an aggressive early deflation strategy, and showing that the rank-structure allows one to parallelize the algorithm, avoiding data dependencies which would be present in the unstructured QR. We exploit the particular structure of the colleague linearization to achieve quadratic complexity and linear storage requirements. The (unbalanced) QR iteration used for Chebyshev rootfinding does not guarantee backward stability on the polynomial coefficients, unless the vector of coefficients satisfy $\|p\| \approx 1$, a hypothesis which is almost never verified for polynomials approximating smooth functions. Even though the presented method is mathematically equivalent to the unbalanced QR algorithm, we show that exploiting the rank structure allows one to guarantee a small backward error on the polynomial, up to an explicitly computable amplification factor $\hat{\gamma}_1(p)$, which depends on the polynomial under consideration. We show that this parameter is almost always of moderate size, making the method accurate on several numerical tests, in contrast with what happens in the unstructured unbalanced QR.

Key words. Chebyshev polynomials, rootfinding, QR, rank-structure, backward error

AMS subject classifications. 65F15, 65H04

DOI. 10.1137/20M1375115

1. Introduction.

The QR method [19], also known as Francis' iteration, is the de facto standard in eigenvalue computations of small to medium size unstructured matrices. It is the method implemented by the `eig` MATLAB command, and is used in most generic eigenvalue routines available in mathematical software packages.

Recently, there has been an increasing amount of contributions dealing with fast variants of the QR iteration for matrices endowed with special structures, most often of the form $A = F + uv^*$, where F is either Hermitian or unitary, and uv^* is a rank 1 correction. Since the QR iteration can be described as a sequence of unitarily similar matrices $A^{(k+1)} = Q_k^* A^{(k)} Q_k$, where $Q_k^* Q_k = I$, the Hermitian-plus-rank-one (resp., unitary-plus-rank-one) structure is maintained throughout the iterations. Algorithms in this class typically achieve $\mathcal{O}(n^2)$ complexity in time and have $\mathcal{O}(n)$ storage requirements.

The roots of a polynomial $p(x)$ expressed in the monomial basis are equal to the eigenvalues of the so-called "companion matrix" whose entries are easily computable by using the coefficients of the polynomial [5, pp. 60–61]:

$$p(x) = \sum_{j=0}^{n} p_n x^n = 0 \iff \det\left(\begin{bmatrix} -\frac{p_{n-1}}{p_n} & \cdots & -\frac{p_1}{p_n} & -\frac{p_0}{p_n} \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix} - xI \right) = 0. \tag{1.1}$$

*Received by the editors October 22, 2020; accepted for publication (in revised form) by J. L. Barlow April 28, 2021; published electronically July 19, 2021.
https://doi.org/10.1137/20M1375115
Funding: The authors are members of the research group INdAM-GNCS. The work of the authors was partially supported by the GNCS project "Metodi low-rank per problemi di algebra lineare con struttura data-sparse."

†Scuola Normale Superiore, Pisa, P.za Cavalieri, 7, 56126 Pisa (PI), Italy (angelo.casulli@sns.it).
‡Department of Mathematics, University of Pisa, L.go B. Pontecorvo, 5, 56127 Pisa (PI), Italy (leonardo.robol@unipi.it).

The companion matrix can be decomposed as the sum of a unitary matrix and a rank 1 correction. Hence, structured QR algorithms allow one to design quadratic complexity methods for computing roots of polynomials. This has been one of the main motivations for developing methods in this class [4, 5, 6, 7, 9, 10, 11]. Only recently some of these methods were proved to be stable eigensolvers; however, it is a much harder challenge to obtain stability with respect to the polynomial coefficients; in this case, using a backward stable eigensolver is not sufficient [15, 16, 17], i.e., a small backward error on the companion matrix does not necessarily imply a small backward error on the polynomial itself. It has been recently shown that some structured methods enjoy this property thanks to the fact that the backward error shares the same unitary-plus-rank-one structure of the companion matrix [6, 26].

Chebyshev polynomials are a key tool for dealing numerically with smooth functions defined on a real interval, as demonstrated by the success of Chebfun [8] and Chebfun2 [27]; hence, the same problem of obtaining a fast and stable QR iteration has been considered for colleague matrices, the Chebyshev analogue of the companion matrix in (1.1), which will be introduced in section 2. Contributions in this direction can be found in [11, 12, 18, 20, 29]. Analogously to the monomial case, computing the roots of polynomials expressed in the Chebyshev basis by eigenvalue methods is not generally backward stable even when the underlying eigenvalues solver has this property; the extensions of the results in [17] is described in [24, 25]. More recently, it has been shown that an eigenvalue solver capable of preserving the Hermitian-plus-rank-one structure in the backward error, and ensuring a small relative backward error on both addends, is stable on the polynomial coefficients [26]. Unfortunately, to the best of our knowledge, this property has not been proved for any of the aforementioned rank-structured methods for the Hermitian-plus-rank-one case. This paper aims at improving this situation. The method analyzed does not possess this property for any polynomial, but we introduce an easily computable parameter that characterizes the stability; we show that this stronger stability property is almost always attained for polynomials arising from approximating smooth functions over $[-1, 1]$, and the parameter is effective at detecting the cases where it does not.

1.1. Main contributions. This work contains several contributions. We reconsider the algorithm described in [18] for computing the eigenvalues of Hermitian-plus-rank-one matrices in the setting of colleague matrices (which belong to this class).

We extend the method by describing how to perform aggressive early deflation working on the structured representation, and we show that—in contrast to what happens for the unstructured QR—using the Hermitian-plus-rank-one structure allows one to easily parallelize the algorithm, avoiding data dependencies during the unitary transformations.

We show that in this context the proposed method is a backward stable eigensolver, the same property of the unstructured unbalanced QR iteration, but only requires $\mathcal{O}(n)$ storage and $\mathcal{O}(n^2)$ floating point operations (flops). However, this is often not enough to deliver accurate results, because balancing plays an important role in this context, but would generally break the Hermitian-plus-rank-one structure, making it not applicable.

For this reason, we introduce an explicitly computable parameter $\hat{\gamma}_j(p)$ that characterizes the backward stability of the method on the polynomial coefficients. When this parameter is moderate, the method produces a small relative backward error on the polynomial coefficients, even if the coefficient vector $\|p\|$ has large norm, in contrast to the unstructured (unbalanced) QR iteration. The boundedness of such

parameter cannot be guaranteed a priori, but holds in most of the practical cases that we have tested and is easy and inexpensive to check at runtime. We demonstrate that this enables the use of the algorithm for rootfinding of analytic functions over a real interval, when coupled with an interpolation step similar to the one in Chebfun [8]. In particular, the obtained accuracy is on par with the one of MATLAB's `eig` (which uses balancing), but with a much lower computational cost.

The FORTRAN code implementing the algorithm (single and double shift, including a parallel version with aggressive early deflation of the single shift code) is publicly available at https://github.com/numpi/chebqr, and is distributed with MATLAB interfaces.

1.2. Notation. We denote by $\mathbb{C}^{m\times n}$ and $\mathbb{R}^{m\times n}$ the sets of $m \times n$ matrices with coefficients in the complex and real fields, respectively.

We employ a MATLAB-like notation for submatrices. For instance, given $A \in \mathbb{C}^{m\times n}$ the matrix $A_{i_1:i_2,j_1:j_2}$ is the submatrix obtained selecting only rows from i_1 to i_2 and columns from j_1 to j_2 (extrema included).

The symbol ϵ_m is used to indicate the unit roundoff. In the remainder of this paper floating point computations are performed in IEEE 754 `binary64` [2], called double precision before [1], for which the unit-roundoff is approximately $1.11 \cdot 10^{-16}$ [22]. The norms are denoted by $\|\cdot\|$. When not explicitly specified, we mean the spectral norm (for matrices), or the Euclidean one (for vectors).

Often we deal with error bounds obtained through backward error analysis, for which we ignore the constants and the (polynomial) dependency in the size of the problem. We denote backward errors on A as δA, and we write the bounds with constants omitted as follows:

$$\|\delta A\| \lesssim \|A\|\epsilon_m \iff \|\delta A\| \leq p(m,n) \cdot \|A\|\epsilon_m,$$

where $p(m,n)$ is a low-degree polynomial in m,n, the dimensions of A; in most cases discussed in this paper, the degree of the polynomial will be between 1 and 3. We say that an algorithm for computing the eigenvalues of A is *backward stable* if it computes the exact eigenvalues of $A+\delta A$ with $\|\delta A\| \lesssim \|A\|\epsilon_m$, and that an algorithm for computing the roots of a polynomial $p(x)$ is backward stable if it computes the roots of a polynomial $\delta p(x)$ where[1] $\|\delta p\| \lesssim \|p\|\epsilon_m$.

2. Structured QR iteration. Let $A = F + uv^*$ be an $n \times n$ upper Hessenberg complex matrix where F is Hermitian, and uv^* is a rank 1 perturbation. In this section we describe how to efficiently compute its eigenvalues in quadratic time. Although this may be applied to any matrix with such structure, the main application that we consider is the computation of the roots of a polynomial

$$p(x) = \sum_{j=0}^{n} p_j T_j(x)$$

expressed in the Chebyshev basis. This can be achieved by computing the eigenvalues of the colleague matrix (originally introduced by [21], and here scaled to make it

[1] In this case we use the notation $\|p\|$ to denote the norm of the coefficient vectors for $p(x)$.

symmetric-plus-rank-one):

$$C = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & & \\ 1 & & \ddots & & & \\ & \ddots & & 1 & & \\ & & 1 & & \sqrt{2} & \\ & & & \sqrt{2} & & \end{bmatrix} - \frac{1}{2p_n}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}\begin{bmatrix} p_{n-1} & p_{n-2} & \dots & p_1 & \sqrt{2}p_0 \end{bmatrix}. \tag{2.1}$$

In section 2.1 we briefly recall the structured QR iteration first proposed by Eidelman, Gemignani, and Gohberg in [18]; we will then improve this approach by showing how to perform aggressive early deflation in a structured way in section 2.2, and how the structure allows for an easy and efficient parallelization of the algorithm in section 2.3.

2.1. Hermitian-plus-rank-one QR. This subsection recalls the structured implicit QR iteration of [18]; this is mathematically equivalent to the standard one, but an efficient representation for the upper Hessenberg matrix is used to reduce the computational and storage costs.

To establish the notation, we first summarize how the implicit QR iteration works in the generic case. Let A be an $n \times n$ irreducible upper Hessenberg matrix, that is, such that $A_{ij} = 0$ for any $i > j+1$, and $A_{ij} \neq 0$ for any $i = j+1$. Suppose a shift $\sigma \in \mathbb{C}$ has been chosen. Then, we compute a Givens rotation G_1, such that $G_1^*(A - \sigma I)e_1$ is a multiple of e_1 and update A by performing a similarity transformation $G_1^* A G_1$. Doing this we break the Hessenberg form; indeed a bulge has been created in the entry $(3, 1)$. The subsequent steps consist of computing Givens rotations $G_2, \dots, G_{n-1}$ such that the matrix $(G_2 \dots G_{n-1})^* A (G_2 \dots G_{n-1})$ is again in Hessenberg form. We refer to this update as a QR sweep, or iteration.

If the shifts σ are chosen appropriately, we expect that in a few sweeps some subdiagonal elements of A will become negligible. This allows one to split the eigenvalue problem into two subproblems (the matrix is now numerically block upper triangular). Most often, the negligible element will be in position $(n, n-1)$; hence, we immediately identify the element in position (n, n) as an eigenvalue and reduce the problem size to $n - 1$.

A more in depth discussion of the QR algorithm, and the role of the shifts, can be found in [30]. In that book the convergence of the algorithm is interpreted in terms of Krylov subspaces; in particular this allows for an easy introduction of the double shift (or, more generally, multishift) QR, which is obtained replacing the initial shifted column $(A - \sigma I)e_1$ with the evaluation of a low degree polynomial $\rho(A)e_1$.

2.1.1. Structure preservation. We note that each upper Hessenberg matrix $A = F + uv^*$ such that $F = F^*$ can be completely determined using an $\mathcal{O}(n)$ parameter. Indeed, we have the following relations for the entries of F in the lower triangular part, excluding the first subdiagonal:

$$0 = A_{ij} = F_{ij} + u_i \overline{v_j} \implies F_{ij} = -u_i \overline{v_j} \qquad \forall i > j+1, \tag{2.2}$$

and therefore, relying on $F_{ij} = \overline{F}_{ji}$, we may write

$$A_{ij} = F_{ij} + u_i \overline{v_j} = \overline{F_{ji}} + u_i \overline{v_j} = u_i \overline{v_j} - \overline{u_j} v_i \qquad \forall j > i+1. \tag{2.3}$$

Hence, all the entries above the first superdiagonal can be determined solely from u and v; the complete matrix A can be recovered storing its diagonal and subdiagonal entries, and the vectors u, v.

In addition, performing a unitary similarity preserves the Hermitian-plus-rank-one structure since

$$QAQ^* = QFQ^* + (Qu)(Qv)^*, \qquad Q^*Q = I.$$

Hence, the structure is inherited by all the matrices produced by the QR iteration.

Algorithmically, we store two vectors $d \in \mathbb{C}^n$ and $\beta \in \mathbb{C}^{n-1}$ whose entries are the diagonal and the subdiagonal entries of A, respectively. This completely determines the lower triangular part of A (thanks to the upper Hessenberg form); the upper part can be retrieved exploiting the symmetry as discussed above, which yields

$$A_{i,j} = \begin{cases} d_i, & i = j, \\ \beta_{i-1}, & i = j+1, \\ \overline{\beta_i} - \overline{u_{i+1}} v_i + u_i \overline{v_{i+1}}, & j = i+1, \\ u_i \overline{v_j} - \overline{u_j} v_i, & j > i+1. \end{cases} \tag{2.4}$$

The preservation of Hermitian-plus-rank-one structure implies that a matrix obtained after k sweeps of QR can be determined using four vectors $d^{(k)}, \beta^{(k)}, u^{(k)}, v^{(k)}$; we call these vectors the generators of $A^{(k)}$.

The structured QR iteration is then implemented as a sequence of rotation acting on consecutive rows. Each of these rotations can be applied to the structured representation by updating in place a constant number of entries in $d^{(i)}, \beta^{(i)}, u^{(i)}, v^{(i)}$.

Storing only these vectors makes the memory storage linear, in contrast to the quadratic cost for the standard QR algorithm.

The details of the iteration are reported in sections 2.1.2 (for the single shift case) and 2.1.3 (for the double shift algorithm).

2.1.2. Single shift iteration. In this section we briefly summarize the single shift iteration of the structured QR algorithm introduced in [18]. Let $A = F + uv^*$ be a Hermitian-plus-rank-one matrix in upper Hessenberg form; we discussed how to store it using $\mathcal{O}(n)$ memory, and we aim at computing its eigenvalues leveraging this representation, and achieving $\mathcal{O}(n^2)$ complexity.

We consider a single shift QR step in this structured format. Let $\rho(z) = z - \sigma$ be a shift polynomial. As in the implicit version of the QR algorithm, to obtain the starting Givens rotator G_1 we compute the first column of the matrix $A - \sigma I$. Since A is in upper Hessenberg form we have

$$\rho(A)e_1 = \begin{bmatrix} d_1 - \sigma \\ \beta_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.5}$$

The rotator such that $G_1^* \rho(A) e_1$ is a multiple of e_1 can be computed by standard BLAS routines, such as `zrotg`.

A is now replaced by $G_1^* A G_1$; since A is represented by means of the four vectors d, β, u, v, we directly update this representation. Note that the two matrices only differ in the top three rows. Thanks to the symmetry, the updated parameters can be recovered from the top 3×2 block of the updated matrix, which we compute as

follows:

$$(G_1^* A G_1)_{1:3,1:2} = \begin{bmatrix} \hat{G}_1^* & \\ & 1 \end{bmatrix} \begin{bmatrix} d_1 & (\overline{\beta_1} - \overline{u_2} v_1) + u_1 \bar{v}_2 \\ \beta_1 & d_2 \\ 0 & \beta_2 \end{bmatrix} \hat{G}_1, \tag{2.6}$$

where $\hat{G}_1 = (G_1)_{1:2,1:2}$.

The 3×2 updated block can be computed directly, using a constant number of floating point operations independent of n, and the the updated entries of d, β can be read off from the updated matrix. Similarly, the updated u, v can be computed by $G_1^* u$ and $G_1^* v$.

As in the standard implicit QR, this operation creates a bulge, breaking the Hessenberg structure in position $(3,1)$. In particular, the matrix cannot be recovered from the generators unless this bulge is stored as well. Since this is just one extra entry, it does not affect the computational costs or the storage.

The bulge can be chased with another rotation G_2 operating on the rows $(2,3)$, whose action can be computed updating the vectors d, β, u, v in a similar fashion. The procedure is repeated until the bulge disappears at the bottom of the matrix.

When applying G_i^* on the left and G_i on the right with $i > 1$ the analogous 3×2 block $A_{i:i+2,i:i+1}$ needs to be considered. We note that to compute G_i the data required are just the subdiagonal element β_{i-1} and the bulge given from the previous step. In Algorithm 2.1 we summarize how to perform the ith bulge chasing step.

Algorithm 2.1 Single shift bulge chasing (ith step).

procedure CHASING(d, β, u, v,bulge)
  $\gamma \leftarrow (\overline{\beta}_i - \overline{u}_{i+1} v_i) + u_i \overline{v}_{i+1}$
  $G^* \leftarrow \text{Givens}\left(\begin{bmatrix} \beta_{i-1} \\ \text{bulge} \end{bmatrix} \right)$
  $\beta_{i-1} \leftarrow \left\| \begin{bmatrix} \beta_{i-1} \\ \text{bulge} \end{bmatrix} \right\|_2$
  $\begin{bmatrix} d_i & \gamma \\ \beta_i & d_{i+1} \\ \text{bulge} & \beta_{i+1} \end{bmatrix} \leftarrow \begin{bmatrix} G^* & \\ & 1 \end{bmatrix} \begin{bmatrix} d_i & \gamma \\ \beta_i & d_{i+1} \\ 0 & \beta_{i+1} \end{bmatrix} G$
  $\begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} \leftarrow G^* \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}$
  $\begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix} \leftarrow G^* \begin{bmatrix} v_i \\ v_{i+1} \end{bmatrix}$
end procedure

The cost of each of the above steps is constant and every iteration requires $\mathcal{O}(n)$ steps; hence, the cost of each iteration is linear in n. Assuming the convergence of all the eigenvalues in a number of steps linear in the dimension of the problem[2] the total cost is quadratic in n.

The described procedure is backward stable in the original data: the computed eigenvalues are the exact ones of $A + \delta A$, where

$$\|\delta A\| \lesssim (\|F\|_2 + \|u\|_2 \|v\|_2)\, \epsilon_m,$$

[2]The QR iteration typically converges in about $2n$ or $3n$ iterations; this measure is only weakly influenced by the matrix under consideration, and is one of the features that makes the QR iteration the method of choice for the dense unsymmetric eigenvalue problems. We refer to section 5 for some numerical experiments supporting this claim.

where $\lesssim$ denotes an upper bound up to a small degree polynomial in the dimension, as is customary for backward error analyses. This result is proven in [18] but is, in general, weaker than the stability guaranteed by the usual unstructured QR iteration, which instead guarantees that $\|\delta A\| \lesssim \|A\|\epsilon_m$. Without further assumptions, $\|F\|_2 + \|u\|_2\|v\|_2$ might be arbitrarily larger than $\|A\|$.

In section 3 we reconsider the backward error analysis focusing on the problem of interest in this paper—the computation of roots for Chebyshev polynomials—and we show that for this case the same rigorous bound of the standard QR iteration can be obtained. In addition, we will also show that the structured method often exhibits superior stability properties.

2.1.3. Double shift iteration. Whenever the matrix A is real, it can be convenient to rely on a double shift iteration, instead of the single shift approach described in the previous section. This makes the method more efficient (since complex arithmetic can be avoided completely), and ensures that the computed eigenvalue has the required complex-conjugation symmetry.

From a high-level perspective, this introduces the following changes. Instead of choosing a degree 1 polynomial $\rho(z) = z - \sigma$ we pick a degree 2 one of the form $\rho(z) = z^2 - 2\Re(\sigma)z + |\sigma|^2$, where $\sigma \in \mathbb{C}$. Then, we determine two rotations $G_{1,1}$ and $G_{1,2}$ such that $G_{1,1}^* G_{1,2}^* \rho(A)e_1$ is a multiple of e_1. Performing a similarity using $G_{1,1}^* G_{1,2}^*$ in place of G_1 creates a perturbation in the upper Hessenberg structure, which is then chased to the bottom of the matrix. This procedure is standard, and we refer the reader to [30] for further details.

In a similar fashion to what is discussed in section 2.1.2, this procedure can be carried out relying on the structured representation of the matrix A. The column vector $\rho(A)e_1$ can be computed with a constant number of operations in terms of d, β, u, v:

$$\rho(A)e_1 = \begin{bmatrix} d_1^2 + \gamma_1\beta_1 - 2\Re(\alpha)d_1 + |\alpha|^2 \\ \beta_1(d_1 + d_2 - 2\Re(\alpha)) \\ \beta_1\beta_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.7}$$

We now compute the two Givens rotations, $G_{1,1}$ and $G_{1,2}$, such that $G_{1,1}^* G_{1,2}^* \rho(A)e_1$ is a multiple of e_1. The matrix $G_{1,1}^* G_{1,2}^* A G_{1,2} G_{1,1}$ differs from A only in the top three rows; analogously to the single shift case, we can recover the updated parameters by only computing the top 4×3 block, and updating u, v separately. This does also provide the three entries composing the "bulge," i.e., the ones that perturb the upper Hessenberg structure.

Let us denote for the sake of brevity by $\gamma_i := (\overline{\beta_i} - \overline{u_{i+1}}v_i) + u_i\bar{v}_{i+1}$ the entries on the upper diagonal of A, recovered according to the procedure described in (2.4). Then, the computation required to apply the two rotation is the following:

$$\begin{aligned} &\left(G_{1,1}^* G_{1,2}^* A G_{1,2} G_{1,1}\right)_{1:4,1:3} \\ &\quad = \begin{bmatrix} \hat{G}_{1,1}^* & \\ & 1 \end{bmatrix} \begin{bmatrix} \hat{G}_{1,2}^* & \\ & 1 \end{bmatrix} \begin{bmatrix} d_1 & \gamma_1 & u_1\overline{v}_3 - \overline{u_1}v_3 \\ \beta_1 & d_2 & \gamma_2 \\ 0 & \beta_2 & d_3 \\ 0 & 0 & \beta_3 \end{bmatrix} \hat{G}_{1,2}\hat{G}_{1,1}, \end{aligned} \tag{2.8}$$

where $\hat{G}_{1,1} = (G_{1,1})_{1:3,1:3}$ and $\hat{G}_{1,2} = (G_{1,2})_{1:3,1:3}$.

The cost in floating point operations of this update is independent of n, and therefore, it accounts for $\mathcal{O}(1)$ in the computational cost.

In the updated matrix of (2.8), the entries in positions $(3,1),(4,1)$, and $(4,2)$ constitute the bulge; they need to be chased down to the bottom of the matrix to complete a single sweep.

We describe in more detail the generic ith step of bulge chasing; assume that the three entries composing the bulge are known from the previous step, or from the initialization of the sweep described above. We denote them by $b_{i+2,i}, b_{i+3,i}, b_{i+3,i+1}$. Then, we compute Givens rotations $G_{i+1,1}$ and $G_{i+1,2}$ such that

$$\hat{G}^*_{i+1,1}\hat{G}^*_{i+1,2}\begin{bmatrix}\beta_i\\ b_{i+2,i}\\ b_{i+3,i}\end{bmatrix} = \begin{bmatrix}c\\0\\0\end{bmatrix} \tag{2.9}$$

for a suitable constant c, where $\hat{G}_{i+1,j} = (G_{i+1,j})_{i+1:i+3,i+1:i+3}$ for $j = 1,2$. The rotations can be used to build the matrix $A^{(i+1)}$ defined as follows:

$$A^{(i+1)} := G^*_{i+1,1}G^*_{i+1,2}A^{(i)}G_{i+1,2}G_{i+1,1}.$$

To continue with the process we need to update the generators used to store the matrix $A^{(i)}$, and also memorize the updated bulge, which will have moved down one step. Updating $u^{(i)}$ and $v^{(i)}$ requires only multiplying them on the left by $G^*_{i+1,1}G^*_{i+1,2}$, that is,

$$u^{(i+1)} = G^*_{i+1,1}G^*_{i+1,2}u^{(i)} \quad \text{and} \quad v^{(i+1)} = G^*_{i+1,1}G^*_{i+1,2}v^{(i)}.$$

Computing the action of the rotation in (2.9) yields the updated value of β_i. To update the other entries in the vectors d and β we compute the action of the rotations on the 4×3 submatrix $A^{(i)}_{i+1:i+4,i+1:i+3}$, which can be written explicitly as follows:

$$\begin{bmatrix}\hat{G}^*_{i+1} & \\ & 1\end{bmatrix}\begin{bmatrix} d^{(i)}_{i+1} & \gamma_{i+1} & u^{(i)}_{i+1}\overline{v}^{(i)}_{i+3} - \overline{u}^{(i)}_{i+3}v^{(i)}_{i+1}\\ \beta^{(i)}_{i+1} & d^{(i)}_{i+2} & \gamma_{i+2}\\ b_{i+3,i+1} & \beta^{(i)}_{i+2} & d^{(i)}_{i+3}\\ 0 & 0 & \beta_{i+3}\end{bmatrix}\hat{G}_{i+1}, \tag{2.10}$$

where $\hat{G}_{i+1} := \hat{G}_{i+1,2}\hat{G}_{i+1,1}$.

After the update has been performed, the new bulge will be available in the entries in positions $(3,1),(4,1)$, and $(4,2)$ of the matrix in (2.10). These entries will form the values $b_{i+3,i+1}, b_{i+4,i+1}$, and $b_{i+4,i+2}$ used in the next chasing step.

The cost of each step is independent of n, and every sweep requires n steps, so the cost of each iteration is linear. Using the same argument employed in the single shift case, and assuming convergence within $\mathcal{O}(n)$ sweeps, we conclude that the algorithm has a quadratic cost.

2.2. Aggressive early deflation. Modern implementations of the QR iteration, such as the one found in LAPACK [3], rely on a number of strategies to improve their efficiency. One of the most effective, developed by Braman, Byers, and Mathias in [14] is known as aggressive early deflation (AED). AED is a method for deflating eigenvalues that, combined with standard deflation, improves the convergence speed, and in particular speeds up the detection of almost deflated eigenvalues.

A practical description of this method can be found in [30]. In this section we consider the case of a matrix $A = F + uv^*$, with the now usual assumption $F = F^*$. It turns out that if we are running the QR iteration (either single or double shift) in the structured format described in sections 2.1.2 and 2.1.3, the AED step can be performed in structured form as well.

We now briefly recall how AED works in general, and then show how these operations can be carried out in a structured way. Let $k \ll n$, and consider the following partitioning of A:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{22} is $k \times k$ and A_{11} is $(n-k) \times (n-k)$. We note that A_{21} is made of a single entry in its top-right corner, which is equal to $A_{n-k+1,n-k}$.

We rely on a simple implementation of the QR iteration (i.e., without the AED scheme) to compute the Schur form of A_{22}. This does not pose any computational challenges, since we assume k to be small. Then, we use the computed Schur form $Q^*A_{22}Q = T$ to perform the following unitary similarity transformation on the original matrix A:

$$\begin{bmatrix} A_{11} & A_{12}Q \\ Q^*A_{21} & Q^*A_{22}Q \end{bmatrix}. \tag{2.11}$$

Note that Q^*A_{21} has nonzero entries only in its last column. Let us call this column x. Then, the AED scheme proceeds by partitioning the vector x as follows:

$$Q^*A_{21}e_{n-k} = x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

where x_1 has $0 \le j \le k$ entries, and j is chosen as small as possible under the constraint that all the entries of x_2 satisfy

$$|x_2|_i \le \min\{|T_{n-k+j+i}|, |A_{n-k+1,n-k}|\} \cdot \epsilon_m. \tag{2.12}$$

If (2.12) is satisfied for some entries, than these can be safely set to zero without damaging the backward stability of the approach, and allowing one to deflate $k - j$ eigenvalues at once. More precisely, it implies that $k - j$ eigenvalues of A_{22} are also (numerically) eigenvalues of A, even though they could not be detected immediately by the usual deflation criterion. The remaining eigenvalue of A_{22} can be put to good use by employing them as shifts for the next sweeps.

The described procedure can be applied directly in the structured format that we have relied upon for developing the QR iteration. As a first step, we need to characterize the structure of the partitioned matrix (2.11). Submatrices of A are also completely determined by the four vectors, as we summarize in the next lemma.

LEMMA 2.1. *Let $A = F + uv^*$ be an upper Hessenberg Hermitian-plus-rank-one matrix, with vector generators d, β, u, v. Then, if we partition*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad A_{11} \in \mathbb{C}^{(n-k)\times(n-k)}, \quad A_{22} \in \mathbb{C}^{k\times k},$$

then also A_{11} and A_{22} are upper Hessenberg Hermitian-plus-rank-one matrices, with generators $d_{1:n-k}$, $\beta_{1:n-k-1}$, $u_{1:n-k}$, $v_{1:n-k}$, and $d_{n-k+1:n}$, $\beta_{n-k+1:n-1}$, $u_{n-k+1:n}$, $v_{n-k+1:n}$, respectively.

Proof. The proof follows by direct verification. ∎

Lemma 2.1 implies that the generator representation used for A immediately gives generator representations for A_{11} and A_{22}. For the latter, we can rely on this representation and the algorithm presented in the manuscript to construct a Schur form, and at the same time update using the computed rotations the last column of A_{21}, which is equal to $\beta_{n-k}e_1$. Hence, we obtain a structured representation of the updated matrix:

$$\begin{bmatrix} A_{11} & A_{12}Q \\ Q^*A_{21} & Q^*A_{22}Q \end{bmatrix}.$$

Then, we proceed as in the dense case, and look at the entries of the vector x composing the last column of Q^*A_{21}. If it contains negligible elements, we set them to zero and deflate the associated components. Note that the check in (2.12) can be stated directly in terms of the generators as follows:

$$|x_2|_i < \min\{|\beta_{n-k}|, |d_{n-k+i+j}|\}\epsilon_m. \tag{2.13}$$

Once the deflated components have been removed, we are left with a matrix for which the top $(n-k-1)\times(n-k-1)$ part is already upper Hessenberg (and its structured representation has not changed), whereas the trailing $(j+1)\times(j+1)$ block has the following form:

$$\begin{bmatrix} * & * & * & \dots & * \\ x_1 & t_{1,1} & t_{1,2} & \dots & t_{1,j} \\ x_2 & & t_{2,2} & \dots & t_{2,j} \\ \vdots & & & \ddots & \vdots \\ x_j & & & & t_{j,j} \end{bmatrix}. \tag{2.14}$$

We remark that for the upper triangular matrix T composing the trailing $j\times j$ block we have a structured representation in terms of generators. In order to continue the QR iterations, we need to reduce this matrix again in upper Hessenberg form.

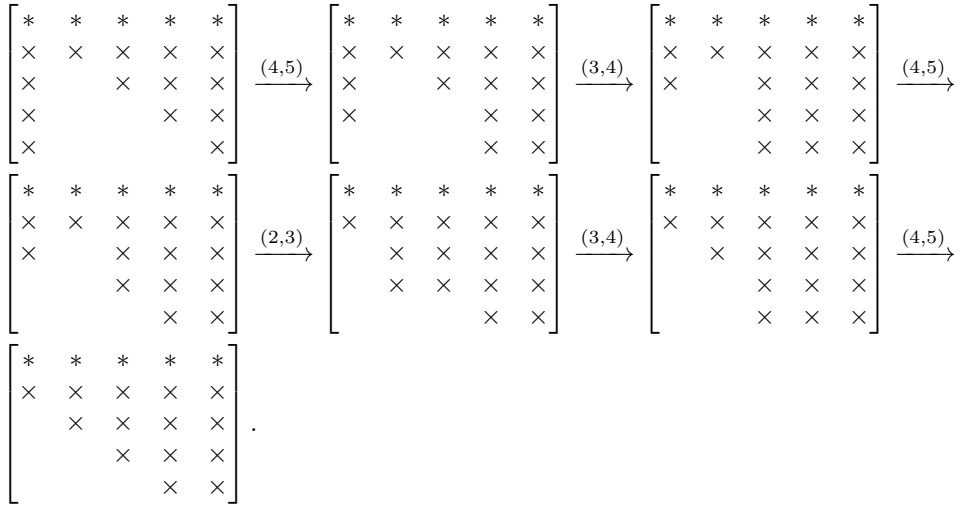
We now assume that the deflated eigenvalues (if any) have been deflated; hence, we are left with a trailing $(j+1)\times(j+1)$ block M which is not in upper Hessenberg form, and needs to be reduced before continuing the iterations. Since this process only works on the trailing block, we can safely ignore the rest of the matrix and its structured parametrization which will not be altered by this reduction; only the last j or $j-1$ entries of d, β, u, v will be updated.

The trailing matrix has the structure of (2.14). The $j\times j$ trailing upper triangular block is Hermitian-plus-rank-one, and we are given a structured representation in terms of the usual vectors d, β, u, v. The matrix M can be reduced to upper Hessenberg form directly producing the structured representation in $\mathcal{O}(j)$ flops working as follows:

(i) The entries x_i of the vector x composing the first column are annihilated one at a time using Givens rotations, starting from the bottom. The transformations are performed as similarities, acting on the left and right.
(ii) When an entry of x is annihilated (except for the last one) a bulge that breaks the Hessenberg form appears. We chase it to the bottom using Givens rotations. This does not create nonzero elements in x.
(iii) For each transformation, we update the vectors d, β, u, v as we have done for the QR chasing sweeps described in sections 2.1.2 and 2.1.3.

After j steps of the above procedure, we obtain a matrix in upper Hessenberg form, and step (iii) guarantees that we have a structured representation of it. We note that replacing the last entries of d, β, u, v, with the computed vectors yields a representation for the complete matrix.

To clarify the algorithm, we pictorially describe the transformation on a 5×5 example, which can be reduced in six steps. Each arrow represents a unitary similarity transformation by means of a Givens rotation acting on the specified rows.



Working with structured arithmetic requires one to update only the diagonal, a few super- and subdiagonal entries (as in section 2.1.2), the first column, and the vectors u, v. The upper triangular part is implicitly updated by keeping track of these changes.

In particular, it is never necessary to perform updates on A_{12}, since this is only implicitly determined by the four vector representation.

Remark 2.2. From the point of view of computational complexity the use of the structured QR algorithm in AED for the computation of the Schur form of the trailing principal submatrix is not of paramount importance; we may compute the Schur form of A_{22} by an unstructured QR iteration, and then recover the updated A_{12} a posteriori thanks to the structure of the matrix. Since the size of A_{22} is expected to be negligible with respect to n, this would not change the overall complexity. Nevertheless, we prefer to work directly on the structured representation using the same operations described in section 2.1.2 because doing otherwise would require recovering the structure from the matrix, complicating the backward error analysis. Indeed, we will show in section 3 how working directly on the structure can be extremely beneficial from the error analysis perspective.

2.3. Parallel structured QR. In section 2.11 we discussed the AED procedure; this enables the early deflation of eigenvalues, but also produces a larger number of shifts for later use in QR sweeps with respect to more traditional criteria such as the Wilkinson or Rayleigh shifts. These can be used in several ways.

When given m shifts $\mu_1, \dots, \mu_m$ that approximate m eigenvalues of A we can start the QR algorithm using $\rho(x) := (x-\mu_1)\cdots(x-\mu_m)$ as shift polynomial. This process is theoretically analogous to applying consecutively the single shift algorithm using the shifts μ_i, but using a shift polynomial of degree m we can deflate m eigenvalues in a single step. This can be implemented in the structured representation by mimicking

the extension done for going from single to double shift code, described in section 2.1.3. This may help with achieving better cache usage on modern processors [23], and is indeed used in LAPACK's QR code by tightly packing several bulges [13, 14].

In this work we describe an alternative strategy for effectively using these shifts, which is not trivially implemented in the unstructured QR iteration: parallelization. We discuss how exploiting the structure makes this task much easier.

In practice, we choose to work with small degree shift polynomials (usually of degree 1 or 2, and hence small bulges), but we allow one to start the following chasing step (with a new shift) before the end of the previous one. This choice allows one to exploit modern processors which are often composed of multiple cores.

The scheme can be described as follows. Assume to have p processors, with $p > 1$. The algorithm chooses a first shift,[3] and starts chasing it down to the bottom of the matrix. Before the end of the sweep, using another processor, we introduce a new bulge at the top and start chasing that one as well.

The idea is difficult to implement in the unstructured QR iteration, because of data dependencies. Indeed, when applying a Givens rotator on the left (resp., on the right), we modify entries in all columns (resp., all rows) of the matrix. So there may be entries modified by two processors at the same time, and this creates the need for careful data synchronization. Let us clarify this matter with an example. Assume that a chasing step is started before the previous bulge reaches the bottom right corner. That is, a Givens rotator is applied on the left and performs a linear combination of the rows i and $i+1$; at the same time, using another processor, a Givens rotator is applied on the right combining the jth and the $(j+1)$th column for $j > i+1$. The entries $a_{i,j}, a_{i+1,j}, a_{i,j+1}, a_{i+1,j+1}$ are modified at the same time by two different processors. Hence, this approach would require one to employ expensive synchronization techniques.

Exploiting the vector representation automatically solves this issue. Since the upper triangular part is never explicitly updated, and we need only work on the entries around the diagonal and the vectors u, v, there is no need for synchronization. Hence, it is possible to start a number of bulges at the top separated one from the other, and chase them in parallel to the end of the matrix without further complications. The differences between data dependency in a parallel chasing for the structured and unstructured QR are depicted in Figure 1.

3. Stability analysis. This section is devoted to the analysis of backward stability of the proposed approach for computing the roots of a polynomial expressed in the Chebyshev basis.

To make the analysis rigorous, it is crucial to specify what we consider to be the input of the algorithm; for instance, the algorithm might be stable when considered as an eigensolver, and unstable when considered as a polynomial rootfinder. We comment on this fact in section 3.1.

We analyze the stability under different viewpoints; we show that the method is stable when considered as an eigensolver, assuming that the input is a colleague linearization of a polynomial. This yields a slightly stronger backward stability result compared to the original analysis of [18], which we discuss in section 3.2. In particular, this implies that the method shares the same favorable properties of the unstructured QR. Then, we consider stability as a polynomial rootfinder. This is a property that the unstructured QR does not possess in general: the computed eigen-

[3] In principle, the shifts may be obtained using any suitable method; we will obtain them from the AED scheme, which provides a set of good shifts that fit well in this framework.

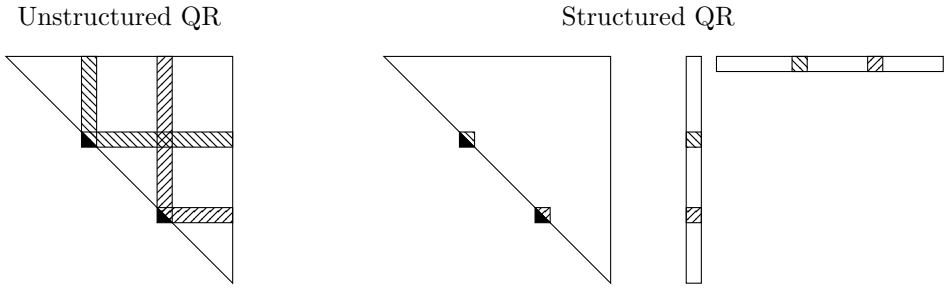


FIG. 1. *Pictorial description of the parallel implementation of chasing multiple bulges in the unstructured and structured QR. On the left, the entries in the matrix that need to be explicitly updated for chasing the bulges down one step in the unstructured QR are marked with diagonal lines; on the right, the same is done for the structured QR, where only diagonal and subdiagonal entries (along with the rank* 1 *correction) are updated explicitly. The vectors representing the rank* 1 *correction are depicted as an outer product of tall and thin matrices.*

values are eigenvalues of a close-by matrix, but not necessarily roots of a close-by polynomial. We show that, under an additional verifiable condition, we can prove that our method is stable as a polynomial rootfinder as well; this additional condition cannot be guaranteed in general, but can be easily checked at runtime, and it holds in typical cases arising from the polynomial approximations of smooth functions on $[-1, 1]$. This is discussed in section 3.3 and demonstrated on practical examples in section 5.

Throughout this section, we use the notation $x \lesssim \epsilon$ to mean that x is smaller than ϵ up to a (low-degree) polynomial in the size of the problem. The latter will be the degree of the polynomial, or the leading dimension of the colleague linearization. From now on, when not explicitly specified, $\|\cdot\|$ will be either the spectral norm (for matrices), or the Euclidean one (for vectors).

We use the notation $\|p\|$ to denote the norm of the vector containing the coefficients of the (monic) polynomial under consideration. We remark that whenever we write $\|\delta p\| \lesssim \|p\|\epsilon_m$ the constant and the polynomial dependency on n hidden in the $\lesssim$ notation need to be independent of the polynomial p under consideration. If there is some additional dependency on p, as will happen in section 3.3 with the term $\hat{\gamma}_j(p)$, then this is explicitly reported.

3.1. Polynomial roots through eigenvalues of colleague matrices. Approximating roots of polynomials by computing the eigenvalues of their linearization is the method of choice in most practical cases. Indeed, this is the way most mathematical software implements commands for this task, such as for MATLAB's `roots` command.

For this reason, the question of whether computing the roots of polynomials using the QR method is stable has been deeply analyzed in the literature [15, 16, 17, 24, 25, 26]. It is known that the QR method implemented in floating point with the usual model of floating point errors applied to a matrix $A \in \mathbb{C}^{n\times n}$ computes a Schur form T satisfying

$$Q^*(A + \delta A)Q = T, \qquad \|\delta A\| \lesssim \|A\| \cdot \epsilon_m.$$

If A is a companion linearization for a polynomial expressed in the monomial basis, Edelman and Murakami proved in the seminal paper [17] that $A+\delta A$ has as eigenvalues the roots of $p + \delta p$, with $\|\delta p\| \lesssim \|p\|^2\epsilon_m$. Here, both p and $p + \delta p$ are assumed to be monic, and we denote by $\|p\|$ the norm of the vectors of coefficients of $p(x)$. More

recently, it has been shown that the same result holds for colleague linearizations as well [24].

This result is satisfactory if $\|p\|$ is moderate. Otherwise, one has to resort to different strategies. One possibility is to perform a preliminary scaling of the matrix A, considering a diagonal scaling with a matrix D computed in order to minimize the norm of $D^{-1}AD$. This choice is often effective, but is not guaranteed to be backward stable; counterexamples where it gives inaccurate results can be found even when generating test polynomials with random coefficients [6]. Nevertheless, this is the chosen method for MATLAB's roots commands, as it behaves "well enough" in practice for polynomials with moderate norms, and sometimes balancing can deliver lower forward errors than a perfectly (backward) stable algorithm.

An alternative choice is to rely on the QZ algorithm, in place of QR. This amounts to computing the generalized Schur form of a pencil $A - \lambda B$, where A and B contain the coefficients of the polynomial $p(x)$; this algorithm has the same properties of the QR iteration, so the computed generalized Schur form $S - \lambda T$ is the exact one of $A + \delta A - \lambda(B + \delta B)$, where $\delta A, \delta B$ are relatively small compared to A and B, respectively. This guarantees that the error on the polynomial coefficients is bounded by $\|\delta p\| \lesssim \|p\|^2 \epsilon_m$; however, in this case the polynomial is not forced to be monic so it can always be scaled to have $\|p\| = 1$, which solves the issue. We refer the reader to [6] for further details.

3.2. Stability as an eigensolver. The algorithm proposed in this work is an improved version of the one in [18]. For the purposes of backward error analysis, we can analyze it as a sequence of unitary similarities described by Givens rotations; if each of these creates a small $\mathcal{O}(\epsilon_m)$ backward error, the whole algorithm is backward stable by composition. Indeed, we can bound the number of rotations to perform with a polynomial in the size of the problem by assuming that the QR iteration converges in a predictable number of steps.

The improvements introduced in this work (namely, AED and parallelization) rely on the same elementary operations, so the backward stability result found in [18] applies to the presented algorithm as well. More specifically, we recall [18, Theorem 4.1].

THEOREM 3.1 (from [18]). *Let $A = F + uv^*$, where $F = F^*$ and $u, v \in \mathbb{C}^{n\times k}$. Let T be the Schur form computed according to Algorithm* 2.1 *described in section* 2. *Then, there exists a unitary matrix Q such that*

$$T = Q^*(A + \delta A)Q, \qquad \|\delta A\|_F \lesssim (\|F\|_F + \|u\|_2\|v\|_2)\,\epsilon_m.$$

We remark that the algorithm of [18] is stated in more generality, and not only for colleague linearizations. In this more general context, it might happen[4] that $\|u\|_2\|v\|_2 \not\lesssim \|A\|_F$, so this theorem does not guarantee backward stability with respect to $\|A\|$.

Nevertheless, the additional hypotheses coming from considering only colleague matrices allow one to state a stronger result.

THEOREM 3.2. *Let $A = F + uv^*$ be the colleague linearization of a degree n polynomial $p(x)$ expressed in the Chebyshev basis, as in* (2.1)*, and T its approximate Schur form computed using the QR iteration described in section* 2. *Then, δA exists such that*

$$T = Q^*(A + \delta A)Q, \qquad \|\delta A\| \lesssim \|A\| \cdot \epsilon_m.$$

[4] As an elementary counterexample, consider $F = -e_1e_1^*$ and $u = v = e_1$, where $\|u\|_2 = \|v\|_2 = \|F\|_2 = 1$ and $\|A\|_F = 0$.

Proof. In view of [18, Theorem 4.1], here restated as Theorem 3.1, we have the bound

$$\|\delta A\| \lesssim (\|F\| + \|u\|\|v\|)\,\epsilon_m,$$

where we have replaced Frobenius norms with 2-norms, since they are equivalent up to a polynomial in n. Since F is the colleague linearization of $T_n(x)$, it has as eigenvalues the Chebyshev points inside $[-1,1]$. F is normal, so we have $\|F\| \leq 1$. In addition, $u = e_1$, and therefore, $\|u\| = 1$. Since for rank 1 matrices and the Euclidean and spectral norm it holds that $\|uv^*\| = \|u\|\|v\|$, we have

$$uv^* = A - F \implies \|v\| \leq 1 + \|A\|.$$

Combining these bounds we get

$$\|\delta A\| \lesssim (1 + \|v\|)\,\epsilon_m \lesssim (2 + \|A\|)\epsilon_m \lesssim \|A\|\epsilon_m. \qquad \square$$

3.3. Stability as a polynomial rootfinder. Theorem 3.2 guarantees that the computed eigenvalues are the exact ones of a slightly perturbed matrix. However, if we are given a polynomial $p(x)$, is natural to ask if these are also the roots of a close-by polynomial $p + \delta p$, satisfying $\|\delta p\| \lesssim \|p\|\epsilon_m$.

The following result generalizes the bound on the norm of the perturbation that holds for the unstructured QR iteration [24].

THEOREM 3.3. *Let $p(x)$ be a monic polynomial in the Chebyshev basis. Then, the roots obtained by computing the eigenvalues of the colleague linearization* (2.1) *relying on the algorithm of section* 2 *are the roots of $p(x) + \delta p(x)$, where*

$$\|\delta p\| \lesssim \|p\|^2 \epsilon_m,$$

where ϵ_m is the unit roundoff, and $\|\cdot\|$ denotes the norm of the vector of coefficients.

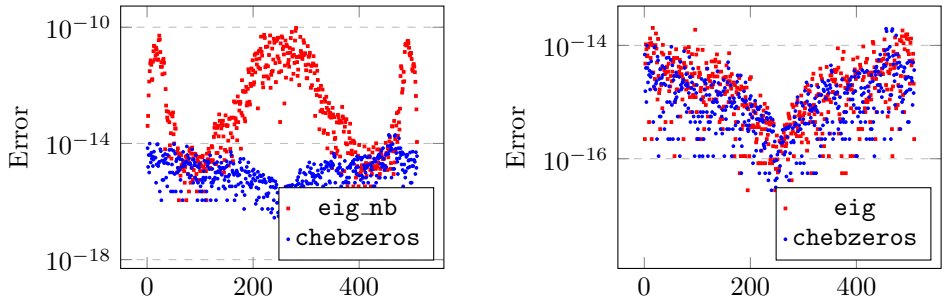
Proof. Thanks to Theorem 3.2 we have that the computed eigenvalues are the ones of $A + \delta A$, with $\|\delta A\| \lesssim \|A\|\epsilon_m$. The result follows by [24, Corollary 2.8]. $\square$

If the polynomial $p(x)$ under consideration has coefficients of moderate norm, the previous result is satisfactory. Otherwise, the quadratic term $\|p\|^2$ suggests that instabilities may arise. In fact, the hypothesis $\|p\| \approx 1$ is far from being satisfied in most cases of practical interest. Indeed, if we interpolate an analytic function $f(x)$ at the Chebyshev points, as we describe in section 4, we expect the magnitude of its coefficients to decay exponentially with the degree; when—after truncation—we normalize the polynomial to make it monic the norm of its coefficient vector is bound to become very large. Hence, one may expect the method to be completely unreliable for the problem of analytic rootfinding.

The bound given by Theorem 3.3 only depends on the fact that the presented QR method is backward stable as an eigensolver. Mathematically, the method is equivalent to calling `eig(A, 'nobalance')` in MATLAB, where `A` is the colleague linearization for $p(x)$.

We now compare these approaches on a simple example: we approximate $f(x) = e^x \sin(800x)$ on $[-1,1]$ by Chebyshev interpolation using Chebfun [8], which yields a polynomial of degree 891. Then, we compute the roots using the structured QR iteration described in section 2 (`chebzeros`), and the (balanced and unbalanced) QR implemented by the `eig` function in MATLAB.

The left plot of Figure 2 shows the absolute errors on the computed roots that are in $[-1,1]$ (which are known exactly for $f(x)$). Clearly, the structured QR (identified



(a) Comparison of the errors produced on the roots by the MATLAB command `eig` without balancing and by `chebzeros`.

(b) Comparison of the errors produced on the roots by the MATLAB command `eig` and by `chebzeros`.

FIG. 2. *Comparison of the errors produced during the computations of the zeros of* $e^x \sin(800x)$ *in* $[-1, 1]$. *The zeros are computed finding the roots of the Chebyshev interpolant using a QR algorithm with and without balancing and the algorithm* `chebzeros`. *The exact zeros of the function are computed analytically.*

by `chebzeros`) outperforms the unbalanced QR (identified by `eig_nb`) with respect to accuracy, despite the mathematical equivalence of these two approaches. Using balancing with `eig` greatly improves the accuracy, as shown by the right plot in Figure 2, and yields about the same accuracy of the proposed algorithm (`chebzeros`).

The plots report forward errors, but for the theoretical analysis we prefer to work with backward errors instead. If c is the vector of coefficients of $p(x)$, given n approximation to the roots y_j, for $j = 1, \dots, n$, we may define the relative backward error as follows:

(3.1)
$$B(p; x) := \min_{\alpha \in \mathbb{R}} \frac{\|c - \alpha \hat{c}\|_2}{\|c\|_2}, \qquad p(x) = \sum_{j=0}^{n} c_j T_j(x), \qquad \prod_{j=1}^{n} (x - y_j) = \sum_{j=0}^{n} \hat{c}_j T_j(x).$$

The number $B(p; x)$ measures the distance of the polynomial $p(x)$ to the closest polynomial of the same degree which vanishes at the computed roots. The backward error for this example is around $1 \cdot 10^{-11}$ for both `chebzeros` and `eig`, but is about $5 \cdot 10^{-7}$ for `eig` without balancing. As already mentioned, this unexpected stability is in contrast with the fact that `chebzeros` is mathematically equivalent to the QR algorithm without balancing.

The rest of this section is devoted to analyzing the backward stability of the approach in a polynomial sense, i.e., to deriving bounds of the form

(3.2)
$$\|\delta p\| \lesssim C \|p\| \epsilon_m.$$

We recall that, for the standard unbalanced QR iteration the results by Edelman and Murakami [17] yield $C \approx \|p\|$. We show that in our approach the constant C can be bounded with a quantity that depends on some features of the magnitude distributions in the vectors u, v during the QR iteration, and this quantity is as follows:

(i) Explicitly computable: it can be given as output by the algorithm at (almost) no additional cost.
(ii) Typically very small for the cases of interest, as we show in the numerical experiments.

Unfortunately, we cannot guarantee C to be always small. However, in our numerical experiments we managed to find only one example where C exhibits enough growth for the results to slightly deteriorate. Despite some effort, we were not able to produce other examples that were not small variations of the latter.

DEFINITION 3.4. *Let u, v be two vectors in $\mathbb{C}^n$. We define $\gamma_j(u, v)$ for any positive integer j as the quantity:*

$$\gamma_j(u,v) := \sup_{i=1,\dots,n-j} \left\| \begin{bmatrix} u_i \\ \vdots \\ u_{\min\{i+j+1,n\}} \end{bmatrix} \right\|_2 \cdot \left\| \begin{bmatrix} v_{\max\{1,i-1\}} \\ \vdots \\ v_{i+j} \end{bmatrix} \right\|_2.$$

Intuitively, the above quantity bounds the norm of the submatrix of the rank 1 matrix uv^* obtained selecting a $(j+1)\times(j+1)$ minor close to the diagonal. This is the submatrix whose entries are updated during a chasing step. We denote by $u^{(k)}, v^{(k)}$ the vectors obtained after applying k rotations in the algorithm described in section 2, starting from $u^{(0)} := u$ and $v^{(0)} := v$. We can now define the supremum over all QR iterations of the quantities $\gamma_j(\cdot,\cdot)$, and we make use of the following notation:

$$\hat{\gamma}_j(p) := \sup_k \gamma_j(u^{(k)}, v^{(k)}), \tag{3.3}$$

where k varies from 0 to the number of rotations performed in the QR algorithm. We note that one has the trivial upper bound:

$$\hat{\gamma}_j(p) \le \|u^{(k)}\|_2 \|v^{(k)}\|_2 = \|u\|_2\|v\|_2 \lesssim \|p\|,$$

thanks to the fact that the Givens rotations preserve the norms throughout the iterations, and $\|u\|_2 = 1$, $\|v\|_2 \approx \|p\|_2$. We now show that the constant C in (3.2) can be bounded using $\hat{\gamma}_j(p)$. In almost all cases, we will have that $\hat{\gamma}_j(p) \ll \|p\|_2$.

To obtain this result, we build on a recent result in [26], which enables one to bound the backward error on p by looking at the backward errors on the Hermitian and rank 1 part separately. We state this result (slightly simplified and with the current notation) for completeness, and to better motivate the next developments.

THEOREM 3.5 (from [26]). *Let $A = F + uv^*$ be the linearization for a polynomial $p(x)$ expressed in the Chebyshev basis given by* (2.1)*. Consider perturbations $\|\delta F\|_2 \le \epsilon_F$, $\|\delta u\| \le \epsilon_u$, and $\|\delta v\| \le \epsilon_v$. Then, the matrix $A + \delta A := F + \delta F + (u + \delta u)(v + \delta v)^*$ linearizes the polynomial $p + \delta p$, where $\|\delta p\| \lesssim \|v\|_2\epsilon_u + \epsilon_v + \|v\|_2\epsilon_F$.*

In our case, bounds on the perturbation on F, u, v are guaranteed by the following result.

LEMMA 3.6. *Let $C = F + uv^*$ be the colleague linearization for a polynomial expressed in the Chebyshev basis. Let $\hat{\gamma}_j(p)$ be the constant defined in* (3.3)*. If the Schur form is computed with the algorithm described in section* 2*, then it is the exact Schur form of*

$$C + \delta C = F + \delta F + (u + \delta u)(v + \delta v),$$

where $\|\delta u\|_2 \lesssim \|u\|_2\epsilon_m$, $\|\delta v\|_2 \lesssim \|v\|_2\epsilon_m$, $\delta F = \delta F^*$, and $\|\delta F\| \lesssim \hat{\gamma}_j(p) \cdot \|F\|_2\epsilon_m$, where $j = 1$ in the single shift case, and $j = 2$ in the double shift one.

Proof. Throughout this proof, we denote by d, β, u, v the vectors of the representation at a generic step of the QR iteration, dropping the indices k.

The Hermitian-plus-rank-one structure preservation in the backward error is an immediate consequence of using the generators d, β, u, v for storing the matrix A. Hence, the Schur form computed according to the algorithm described in section 2 can be written as the exact Schur form of a slightly perturbed matrix:

$$T = Q^*(F + \delta F + (u + \delta u)(v + \delta v)^*)Q$$

for some unitary matrix Q and perturbations $\delta F = \delta F^*$, δu, and δv. The bounds on the backward errors on u, v follow by standard backward error analysis since all the rotations G are unitary.

The Hermitian part F is not computed directly, rather it is is implicitly determined by the vectors d, β, u, v. Thanks to the symmetry, we may just prove that the backward error on the lower triangular part is bounded. The vectors d, β are contaminated with an error that can be accounted for in δF and is created when applying a rotation on the small blocks around the diagonal as in (2.6) (for the single shift case) and in (2.8), (2.10) for the double shift one.

In the single shift case, the updated d, β are obtained[5] by applying unitary operations on the 3×3 matrix B that contains one superdiagonal and up to the third subdiagonal of the matrix A. This is computed relying on (2.4), and the backward error can be bounded by $\|B\|_2 \epsilon_m$ up to a small constant. When operating on a generic block we have in the single shift case

$$B = F_{i:i+2,i-1:i+1} + W,$$

where W is constructed according to (2.2) and (2.3) using u, v; in particular, W is obtained selecting a few entries from uv^*, and satisfies $\|W\|_2 \leq \gamma_1(u, v) \leq \hat{\gamma}_1(p)$. Using $\frac{1}{\sqrt{2}} \leq \|F\|_2 \leq 1$ we obtain $\|B\| \lesssim \hat{\gamma}_1(p) \cdot \|F\|_2$ which in turn implies that applying Givens rotations on B and discarding backward errors on diagonals and subdiagonal entries of F yields the upper bounds

$$|\delta F_{ij}| \lesssim \hat{\gamma}_1(p) \cdot \|F\|_2 \epsilon_m, \qquad i \in \{j, j+1\}.$$

It now remains to show that the elements below the first subdiagonal, which are only implicitly determined through u, v, respect a similar bound. We consider a single rotation G acting on two consecutive indices $(s, s+1)$. By standard backward error analysis, we have that the floating point result for applying G to a vector w will have the form $\mathrm{fl}\,[Gw] = (G + \delta G)w$, where $\|\delta G\| \lesssim w$ [22].

Thanks to the upper Hessenberg structure of A and $A + \delta A$, for every $i > j + 1$ we may write

$$0 = \mathrm{fl}\,[G(A + \delta A)G^*]_{ij} = \mathrm{fl}\,[GFG^*]_{ij} + ((G + \delta G_u)u)_i \overline{((G + \delta G_v)v)}_j.$$

Hence, at the first order we compute the exact unitary transformation of a perturbed matrix $F + \delta F + (G + \delta G_u)uv^*(G + \delta G_v)^*$, such that

$$\begin{aligned} 0 &= F_{ij} + \delta F_{ij} + u_i v_j + (\delta G_u u)_i v_j + u_i (\delta G_v v)_j^* \\ &\implies |\delta F_{ij}| \leq |(\delta G_u u)_i||v_j| + |u_i||(\delta G_v v)_j|, \end{aligned}$$

[5] We have extended the matrix B with one more column on the left with respect to the one in (2.6) to also take into account the update to β_{i-1}, even though in section 2.1.2 it is described as a separate step.

since $F_{ij} + u_i v_j = 0$. We now observe that

$$\|\delta G_u u\|_2 \lesssim \left\| \begin{bmatrix} u_s \\ u_{s+1} \end{bmatrix} \right\|_2 \epsilon_m, \qquad \|\delta G_v v\|_2 \lesssim \left\| \begin{bmatrix} v_s \\ v_{s+1} \end{bmatrix} \right\|_2 \epsilon_m.$$

Hence, we can bound the magnitude of δF_{ij} by

$$|\delta F_{ij}| \lesssim \left(\left\| \begin{bmatrix} u_s \\ u_{s+1} \end{bmatrix} \right\|_2 |v_j| + |u_i| \left\| \begin{bmatrix} v_s \\ v_{s+1} \end{bmatrix} \right\|_2 \right) \epsilon_m.$$

Since we are in the lower triangular part below the first subdiagonal, we have that

$$F_{s:s+1,j} = -u_{s:s+1}\overline{v}_j \implies \left\| \begin{bmatrix} u_s \\ u_{s+1} \end{bmatrix} \right\|_2 |v_j| = \|F_{s:s+1,j}\|_2 \le \|F\|_2,$$

and similarly for the other term. Hence, for these lower subdiagonal elements we can guarantee $|\delta F_{ij}| \le \|F\|_2 \epsilon_m$, and therefore, we have the global bound $\|\delta F\|_2 \lesssim \hat{\gamma}_1(p) \cdot \|F\|_2 \epsilon_m$.

Repeating the same steps for the double shift case yields essentially the same result, with the only exception that two upper diagonals instead of one need to be considered in (2.8) and (2.10); hence, we have $\|\delta F\| \lesssim \hat{\gamma}_2(p) \cdot \|F\|_2 \epsilon_m$. ∎

We can now prove the main result, which uses the structure of the matrix backward error to characterize the backward error on the polynomial coefficients.

THEOREM 3.7. *Let $C = F + uv^*$ be the colleague matrix of a monic polynomial $p(x)$, and let $\hat{\gamma}_j(p)$ be defined as in* (3.3)*. Then, the eigenvalues computed by the structured QR iteration of section* 2 *are the roots of a polynomial $p + \delta p$ satisfying*

$$\|\delta p\| \lesssim \hat{\gamma}_j(p) \cdot \|p\| \cdot \epsilon_m,$$

where $j = 1$ in the single shift case, and $j = 2$ for the double shift one.

Proof. In view of Lemma 3.6, we have the backward error bound $\|\delta F\| \lesssim \hat{\gamma}_j(p)\epsilon_m$, where j is either 1 or 2 depending on the chosen shift degree. Concerning u, v, by standard backward error analysis of the Givens rotations we obtain $\|\delta u\| \lesssim \epsilon_m$ and $\|\delta v\| \lesssim \|v\|\epsilon_m$, since $\|u\| = 1$. We use these different bounds on the backward errors in Theorem 3.5 to obtain $\|\delta p\| \lesssim \hat{\gamma}_j(p)\|p\|\epsilon_m$. ∎

4. Rootfinding for analytic functions on $[-1, 1]$. The tools developed in the previous section can be used for approximating the roots of an analytic function $f(x)$ over $[-1, 1]$. This can be achieved by finding the roots of the Chebyshev interpolant of the function, using the presented algorithm for the computation of the eigenvalues of the colleague matrix.

From a high-level perspective, the approach works as follows:

(i) The function $f(x)$ is evaluated at the Chebyshev points x_i, $i = 1, \dots, n$, and its polynomial interpolant at those points is computed using the FFT (see [28]). The degree is adaptively chosen to ensure that $\|f(x) - p(x)\|_\infty \lesssim \epsilon_m$.

(ii) The roots of the interpolating polynomial are computed using the QR iteration proposed in section 2.1.3.

(iii) Among the computed roots, the ones in $[-1, 1]$ are returned by the algorithm.

We refer to the behavior of the `roots` command in Chebfun [8, 28] for what concerns points (i) and (iii), and we employ the structured QR iteration for addressing point (ii). We now discuss why we generally expect $\hat{\gamma}_j(p)$ to be small on such examples. Let us recall the definition of a Bernstein ellipse.

DEFINITION 4.1. *Let* $\rho \geq 1$. *The* Bernstein ellipse E_ρ *is the set*

$$E_\rho := \left\{ \frac{z + z^{-1}}{2}, \quad |z| = \rho \right\} \subseteq \mathbb{C}.$$

Bernstein ellipses are the Chebyshev analogue of circles for Taylor series. If a function is analytic inside E_ρ, the coefficients of its Chebyshev series decay as $\mathcal{O}(\rho^{-k})$, and the constant depends on the maximum of the absolute value of the function on E_ρ. We refer the reader to [28, Chapter 8] for further details. Remarkably, the polynomial interpolant at the Chebyshev points, which are numerically much easier to determine than the coefficients of the Chebyshev series, have the same decay property, just weakened by a factor of 2 (see the proof of [28, Theorem 8.2] and Chapter 4 on aliasing in the same book).

As already pointed out, normalizing such polynomial approximant to be monic requires one to divide by a very small leading coefficient, making the norm of polynomial coefficients very large.

5. Numerical experiments.

The experiments have been run on a server with two Intel Xeon E5-2650v4 CPUs with 12 cores and 24 threads each, running at 2.20 GHz, using MATLAB R2017a with the Intel Math Kernel Library Version 11.3.1. The parallel implementation is based on OpenMP. Throughout this section, we refer to the algorithm described in section 2 as `chebzeros`.

5.1. Accuracy.

We have tested the accuracy of the proposed algorithm by computing the roots of several polynomials; we have tested both random polynomials of different degrees, and the more representative example of polynomials obtained by approximating smooth functions over $[-1, 1]$. We have also tested random oscillatory functions generated with the Chebfun command `randnfun`. For `chebzeros`, and the QR iteration with and without balancing we have computed the backward errors on the original polynomial.

The results are summarized in Table 1. For each test, the method also returns the value of $\hat{\rho}_1(p)$ computed during the iterations. We see that this value is relatively small for all functions considered, except for $f(x) = \sin(1/(x^2 + 10^{-2}))$. In fact, the accuracy on this particular example is not on par with the QR iteration. However, we remark that the accuracy for the computed roots in $[-1, 1]$ (which are easy to compute explicitly in this case) is at the machine precision level, and the only inaccurate roots are the ones close to the boundary of the Bernstein ellipse where the function is defined. We have not been able to find other examples (except ones based on modifying this particular function) where this happens. We are still unaware of whether the algorithm may be modified (for instance, with a smarter shifting strategy) to avoid this growth. Upon closer inspection, it seems to be caused by a single entry in the vectors u, v which is growing large.

We note that the QR without balancing performs poorly in all the cases where analytic functions are involved since these correspond to companion matrices with a large norm. The behavior is instead comparable to the balanced case (and to `chebzeros`) for polynomials with random Chebyshev coefficients.

5.2. Performances and asymptotic complexity.

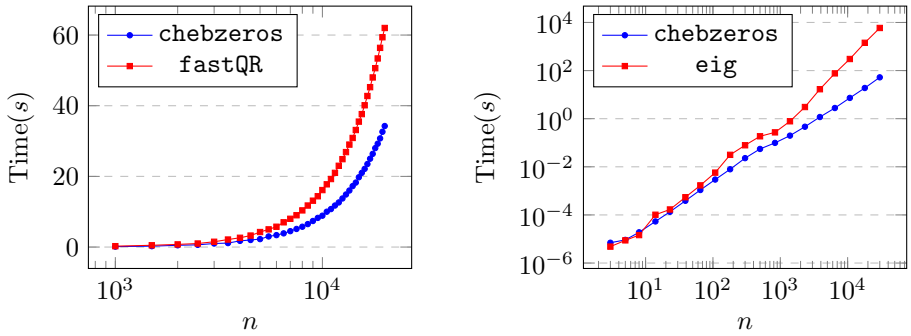
We tested the speed of the proposed algorithm experimentally. The single shift iteration has been implemented in FORTRAN 90 and a MEX file has been used to interface it with MATLAB.

First, we have tested the speed up obtained by modifying the algorithm of [18] introducing the aggressive early deflation strategy. As is visible in Figure 3a, the

TABLE 1

Relative backward errors on the polynomial coefficients obtained using different methods, as defined in (3.1). **chebzeros** *refers to the fast unbalanced QR iteration presented in this paper,* **eig** *is the balanced QR implemented in LAPACK, as included in MATLAB, and* **eig_nb** *is the unbalanced QR in LAPACK. The symbol $p_n(x)$ is used to denote a monic polynomial of degree n where the coefficients of degree smaller than n are chosen according to a Gaussian distribution of $N(0,1)$. The column named degree reports the degree of the polynomial approximant.* **randnfun** *refers to random oscillatory functions generated with this command included in Chebfun. $J_0(x)$ denotes the Bessel function of the first kind. The column $\hat{\gamma}_1(p)$ denotes the amplification factor for the backward error on the polynomial predicted by Theorem* 3.7, *$\|p\|_2$ the norm of the monic polynomial, and # its the total number of iterations required for convergence.*

$f(x)$	Degree	chebzeros	eig	eig_nb	$\hat{\gamma}_1(p)$	$\|p\|_2$	# its
$p_{100}(x)$	100	1.7×10^{-12}	7.6×10^{-13}	8.6×10^{-13}	1.02	5.18	262
$p_{200}(x)$	200	1.6×10^{-12}	2.5×10^{-12}	2.1×10^{-12}	7.1×10^{-1}	6.88	501
$p_{500}(x)$	500	6.1×10^{-12}	1.5×10^{-11}	2.8×10^{-11}	5.0×10^{-1}	1.1×10^{1}	1180
$p_{1000}(x)$	1000	2.2×10^{-11}	1.0×10^{-10}	7.9×10^{-11}	1.35	1.6×10^{1}	1896
$\log(1+x+10^{-3})$	688	7.7×10^{-12}	1.3×10^{-11}	1.8×10^{-2}	7.0×10^{3}	3.5×10^{15}	1734
$\sqrt{x+1.01} - \sin(10^2 x)$	180	7.4×10^{-13}	3.2×10^{-13}	7.2×10^{-6}	1.4×10^{1}	1.1×10^{15}	522
$e^x \sin(800x)$	891	1.2×10^{-11}	9.2×10^{-12}	5.7×10^{-7}	3.01	7.3×10^{13}	1963
$\sin(\frac{1}{x^2+10^{-2}})$	1430	1.6×10^{-6}	1.2×10^{-11}	2.6×10^{-1}	4.2×10^{8}	3.2×10^{15}	2818
randnfun(1e-1)	711	3.6×10^{-12}	4.0×10^{-12}	4.2×10^{-7}	2.58	1.1×10^{14}	1780
randnfun(1e-2)	710	3.8×10^{-12}	2.8×10^{-12}	1.6×10^{-7}	3.47	8.2×10^{13}	1754
randnfun(5e-3)	1355	6.9×10^{-12}	6.9×10^{-12}	1.6×10^{-6}	1.66	4.7×10^{13}	2846
$J_0(20x)$	50	3.3×10^{-14}	1.9×10^{-14}	2.4×10^{-14}	9.4×10^{1}	1.2×10^{15}	154
$J_0(100x)$	148	1.3×10^{-13}	1.5×10^{-13}	2.6×10^{-11}	1.4×10^{1}	4.7×10^{14}	388
$\frac{e^{x^2-\frac{1}{2}}-1}{10^{-2}+x^2}$	380	4.5×10^{-13}	1.7×10^{-13}	6.5×10^{-1}	1.2×10^{3}	1.2×10^{16}	1192
$\frac{e^{x^2-\frac{1}{2}}-1}{10^{-4}+x^2}$	3632	3.6×10^{-13}	3.0×10^{-13}	9.6×10^{-1}	9.3×10^{2}	1.3×10^{16}	6480



(a) Comparison of the CPU time of the algorithm implemented with and without the aggressive early deflation. Both the algorithms have been executed sequentially, using only one core.

(b) Comparison of the CPU time of the algorithm **chebzeros** with the CPU time taken by the MATLAB command **eig**. Both the algorithms have been executed using four cores.

FIG. 3.

speed up is considerable for large matrices. In more detail, Figure 3a compares the CPU time of the two algorithms for randomly generated polynomials for different values of the degree n.

In Figure 3b we compare **chebzeros** with the MATLAB command **eig** applied to the companion matrix in the Chebyshev basis as in (2.1). We observe that **chebzeros** is faster than **eig** already for n about 10. For polynomials of a very high degree the difference of the CPU time is remarkable. This fact is coherent with the theory since

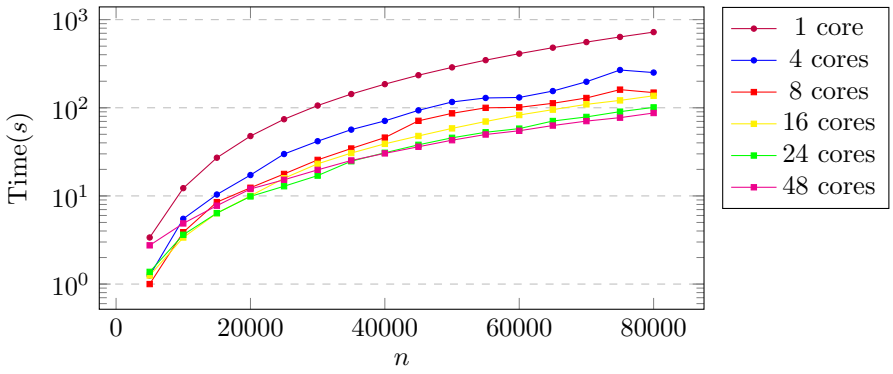


FIG. 4. *Comparison of the clock time of* `chebzeros` *exploiting parallelism using* 1, 4, 8, 16, 24, *and* 48 *cores, for the computation of the roots of random polynomials expressed in the Chebyshev basis, with different degrees* n.

the asymptotic cost of `chebzeros` is $\mathcal{O}(n^2)$ while the asymptotic cost of `eig` is $\mathcal{O}(n^3)$.

Then, we enabled the concurrency on the implementation as described in section 2.3, and again we have tested the performance for different values of n. In order to attain optimal performances, the aggressive early deflation needs to provide a number of shifts that is larger than the number of available cores; hence, the dimension of the block subject to the AED has been increased proportionally to the number of available cores. After some tuning, it has been found the optimal choice to be about nine times the number of available cores.

This choice produces a number of shifts that is about six times the number of parallel tasks. As the dimension of the problem decreases, thanks to deflations, the use of a large number of cores becomes less advantageous. Hence, the implementation halves the number of concurrent tasks (and hence the needed shifts) when the size of the matrix becomes smaller than 64 times the number of used cores.

In Figure 4 are reported the clock times needed for the computation using 1, 4, 8, 16, 24, and 48 cores. It is immediate to note the good performances of the parallel implementation already with four cores, which brings a speed up of about 300%. The gain from increasing the concurrency is reduced as the number of cores increases up to 48. We note that the server only had 24 combined physical cores, and going above 12 required communication between the different CPUs, which inevitably reduces the efficiency of the parallelization. Considering that most consumer hardware has between 2 and 8 or 16 cores, this shows that the method is well tuned for the currently available architectures.

6. Conclusions. A quadratic time structured QR algorithm for Hermitian-plus-rank-one matrices has been presented by improving the ideas originally discussed in [18]. The method relies on a structured representation for the matrix that requires one to store only four vectors. We have shown that it is possible and advantageous to perform aggressive early deflation in this format, and that the structured representation allows one to easily parallelize the scheme avoiding many of the difficulties present in dense chasing algorithms.

The backward stability of the method has been carefully analyzed, with a particular focus on the use of the algorithm for polynomial rootfinding in the Chebyshev basis. It has been shown that, under suitable assumptions that are easy to verify

at runtime, the method has a small backward error on the polynomial coefficients, a property that is not present in the balanced and unbalanced QR iteration. Numerical experiments confirm the stability in the vast majority of the considered cases, and in particular when dealing with polynomial approximating smooth functions. We also show that this criterion is effective in detecting the cases where there could have been some accuracy loss. In all the cases considered the roots in $[-1, 1]$, which are the ones of interest, were computed up to machine precision.

Acknowledgment. We wish to thank the referees, who helped to greatly improve the clarity of this manuscript.

REFERENCES

[1] *IEEE standard for binary floating-point arithmetic*, in ANSI/IEEE Std 754-1985, 1985, pp. 1–20, https://doi.org/10.1109/IEEESTD.1985.82928.

[2] *IEEE standard for binary floating-point arithmetic*, in IEEE Std 754-2008, 2008, pp. 1–70, https://doi.org/10.1109/IEEESTD.2008.4610935.

[3] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, SIAM, 1999, https://doi.org/10.1137/1.9780898719604.

[4] J. Aurentz, T. Mach, L. Robol, R. Vandebril, and D. Watkins, *Fast and backward stable computation of eigenvalues and eigenvectors of matrix polynomials*, Math. Comp., 88 (2019), pp. 313–347.

[5] J. L. Aurentz, T. Mach, L. Robol, R. Vandebril, and D. S. Watkins, *Core-Chasing Algorithms for the Eigenvalue Problem*, SIAM, Philadelphia, 2018, https://doi.org/10.1137/1.9781611975345.

[6] J. L. Aurentz, T. Mach, L. Robol, R. Vandebril, and D. S. Watkins, *Fast and backward stable computation of roots of polynomials, Part* II*: Backward error analysis; companion matrix and companion pencil*, SIAM J. Matrix Anal. Appl., 39 (2018), pp. 1245–1269, https://doi.org/10.1137/17M1152802.

[7] J. L. Aurentz, T. Mach, R. Vandebril, and D. S. Watkins, *Fast and backward stable computation of roots of polynomials*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 942–973, https://doi.org/10.1137/140983434.

[8] Z. Battles and L. N. Trefethen, *An extension of MATLAB to continuous functions and operators*, SIAM J. Sci. Comput., 25 (2004), pp. 1743–1770, https://doi.org/10.1137/S1064827503430126.

[9] D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg, *A fast implicit QR eigenvalue algorithm for companion matrices*, Linear Algebra Appl., 432 (2010), pp. 2006–2031.

[10] D. A. Bini, F. Daddi, and L. Gemignani, *On the shifted QR iteration applied to companion matrices*, Electron. Trans. Numer. Anal., 18 (2004), pp. 137–152.

[11] D. A. Bini, L. Gemignani, and V. Y. Pan, *Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations*, Numer. Math., 100 (2005), pp. 373–408.

[12] P. Boito, Y. Eidelman, and L. Gemignani, *Implicit QR for rank-structured matrix pencils*, BIT, 54 (2014), pp. 85–111.

[13] K. Braman, R. Byers, and R. Mathias, *The multishift QR algorithm, Part I: Maintaining well-focused shifts and level* 3 *performance*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 929–947, https://doi.org/10.1137/S0895479801384573.

[14] K. Braman, R. Byers, and R. Mathias, *The multishift QR algorithm, Part II: Aggressive early deflation*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 948–973, https://doi.org/10.1137/S0895479801384585.

[15] F. De Terán, F. M. Dopico, and J. Pérez, *Backward stability of polynomial root-finding using Fiedler companion matrices*, IMA J. Numer. Anal., 36 (2016), pp. 133–173.

[16] F. M. Dopico, P. W. Lawrence, J. Pérez, and P. Van Dooren, *Block Kronecker linearizations of matrix polynomials and their backward errors*, Numer. Math., 140 (2018), pp. 373–426.

[17] A. Edelman and H. Murakami, *Polynomial roots from companion matrix eigenvalues*, Math. Comp., 64 (1995), pp. 763–776.

[18] Y. Eidelman, L. Gemignani, and I. Gohberg, *Efficient eigenvalue computation for quasisep-*

arable Hermitian matrices under low rank perturbations, Numer. Algorithms, 47 (2008), pp. 253–273.

[19] J. G. Francis, *The QR transformation a unitary analogue to the LR transformation—Part 1*, Comput. J., 4 (1961), pp. 265–271.

[20] L. Gemignani and L. Robol, *Fast Hessenberg reduction of some rank structured matrices*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 574–598, https://doi.org/10.1137/16M1107851.

[21] I. Good, *The colleague matrix, a Chebyshev analogue of the companion matrix*, Quart. J. Math. Oxford Ser. (2), 12 (1961), pp. 61–68.

[22] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002, https://doi.org/10.1137/1.9780898718027.

[23] L. Karlsson, D. Kressner, and B. Lang, *Optimally packed chains of bulges in multishift QR algorithms*, ACM Trans. Math. Software, 40 (2014), 12.

[24] Y. Nakatsukasa and V. Noferini, *On the stability of computing polynomial roots via confederate linearizations*, Math. Comp., 85 (2016), pp. 2391–2425.

[25] V. Noferini and J. Pérez, *Chebyshev rootfinding via computing eigenvalues of colleague matrices: When is it stable?*, Math. Comp., 86 (2017), pp. 1741–1767.

[26] V. Noferini, L. Robol, and R. Vandebril, *Structured Backward Errors in Linearizations*, preprint, https://arxiv.org/abs/1912.04157, 2019.

[27] A. Townsend and L. N. Trefethen, *An extension of Chebfun to two dimensions*, SIAM J. Sci. Comput., 35 (2013), pp. C495–C518, https://doi.org/10.1137/130908002.

[28] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2019, https://doi.org/10.1137/1.9781611975949.

[29] R. Vandebril and G. M. Del Corso, *An implicit multishift QR-algorithm for Hermitian plus low rank matrices*, SIAM J. Sci. Comput., 32 (2010), pp. 2190–2212, https://doi.org/10.1137/090754522.

[30] D. S. Watkins, *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*, SIAM, Philadelphia, 2007, https://doi.org/10.1137/1.9780898717808.