ROOTS OF POLYNOMIALS: ON TWISTED QR METHODS FOR COMPANION MATRICES AND PENCILS*

JARED L. AURENTZ†, THOMAS MACH¶, LEONARDO ROBOL‡, RAF VANDEBRIL‡, AND DAVID S. WATKINS§

Abstract. Two generalizations of the companion QR algorithm by J.L. Aurentz, T. Mach, R. Vandebril, and D.S. Watkins, SIAM Journal on Matrix Analysis and Applications, 36(3): 942–973, 2015, to compute the roots of a polynomial are presented.

First, we will show how the fast and backward stable QR algorithm for companion matrices can be generalized to a QZ algorithm for companion pencils. Companion pencils admit a greater flexibility in scaling the polynomial and distributing the matrix coefficients over both matrices in the pencil. This allows for an enhanced stability for polynomials with largely varying coefficients.

Second, we will generalize the pencil approach further to a twisted QZ algorithm. Whereas in the classical QZ case Krylov spaces govern the convergence, the convergence of the twisted case is determined by a rational Krylov space.

A backward error analysis to map the error back to the original pencil and to the polynomial coefficients shows that in both cases the error scales quadratically with the input.

An extensive set of numerical experiments supports the theoretical backward error, confirms the numerical stability and shows that the computing time depends quadratically on the problem size.

Key words. polynomial, root, companion matrix, companion pencil, eigenvalue, QR algorithm, QZ algorithm, rotators, core transformation, backward stability

AMS subject classifications. 65F15, 65H17, 15A18, 65H04

1. Introduction.

There are many ways [30] to compute roots of polynomials. Here we focus on computing the n *roots* of a complex polynomial $p(z)$ expressed in terms of the monomial basis:

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0,$$

where we assume $a_n \neq 0$ and $a_0 \neq 0$. Note that if $a_0 = 0$ (resp. $a_n = 0$), we can immediately extract a root 0 (resp. ∞). We will compute the roots by finding the eigenvalues of an associated matrix pencil (V, W), where

$$V = \begin{bmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ & \ddots & \ddots & & \vdots \\ & & 1 & 0 & -a_{n-2} \\ & & & 1 & -a_{n-1} \end{bmatrix} \quad \text{and} \quad W = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & a_n \end{bmatrix}. \tag{1.1}$$

* The research was partially supported by the Research Council KU Leuven, projects C14/16/056 (Invers-free Rational Krylov Methods: Theory and Applications), CREA/13/012 (Can Unconventional Eigenvalue Algorithms Supersede the State-Of-The-Art), PFV/10/002 (Optimization in Engineering, OPTEC), by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization).

†Instituto de Ciencias Matemáticas, Universidad Autónoma de Madrid, Madrid, Spain (JaredAurentz@gmail.com).

‡Department of Computer Science, KU Leuven, Leuven, Belgium; ({Leonardo.Robol,Raf.Vandebril}@cs.kuleuven.be).

§Department of Mathematics, Washington State University, Pullman, WA 99164-3113, USA; (watkins@math.wsu.edu).

¶Department of Mathematics, School of Science and Technology, Nazarbayev University, 010000 Astana, Kazachstan; (thomas.mach@nu.edu.kz).

It is not hard to see that $p(z) = \det(V - zW)$.

This pencil is already in Hessenberg-triangular form, so one can apply the QZ algorithm [32, 40, 41] directly to compute the roots in $\mathcal{O}(n^3)$ time with $\mathcal{O}(n^2)$ storage. However, if we exploit the unitary-plus-rank-one structure of both matrices, we can reduce both the computational complexity and the storage cost by an order of magnitude.

We have already applied these techniques [4] to the *companion matrix*

$$A = \begin{bmatrix} & & -a_0/a_n \\ 1 & & -a_1/a_n \\ & \ddots & \vdots \\ & 1 & -a_{n-1}/a_n \end{bmatrix}, \tag{1.2}$$

whose eigenvalues also coincide with the zeros of the polynomial $p(z)$. Computing the roots via the pencil, instead of the companion matrix will be more expensive computationally, but it will yield more accurate results in some cases. Most notably, if a_n is tiny, some of the entries $-a_k/a_n$ in (1.2) will be very large, which is bad for stability. In addition, the pencil approach offers extra freedom, allowing for a more advanced balancing, leading to better backward error bounds [22, 36].

Some background information, including an overview of previous companion root-finding efforts, is presented in Section 2. The memory-efficient factorization of a companion pencil (V, W)

$$V = \begin{bmatrix} 0 & & & -v_1 \\ 1 & & & -v_2 \\ & 1 & & -v_3 \\ & & \ddots & \vdots \\ & & 1 & -v_n \end{bmatrix} \quad \text{and} \quad W = \begin{bmatrix} 1 & & & w_1 \\ & 1 & & w_2 \\ & & \ddots & \vdots \\ & & 1 & w_{n-1} \\ & & & w_n \end{bmatrix}, \tag{1.3}$$

whose roots agree with those of $p(z)$ is introduced in Section 3. We will show that 5 sequences of n small 2×2 unitary matrices suffice. Moreover, we have opted to describe and solve the problem (1.3) in its full generality and therefore did not restrict our description to the classical pencil (1.1). The companion QZ algorithm itself is discussed in Section 4. Section 5 explores extended QZ algorithms, generalizing thereby the work of Bevilacqua, Del Corso, and Gemignani [7]. Initialization of an arbitrary twisted form is discussed in Section 6. In Section 7 we examine the backward stability and prove that the backward error scales quadratically w.r.t. the input; moreover we will rectify an error in the proof of Aurentz et. al. [4]. We conclude with numerical experiments in Section 8 and conclusions in Section 9.

2. Preliminaries and earlier work. Computing roots of polynomials in the monomial basis via the companion matrix has been the subject of study of various research teams. Let us catalogue these efforts in two sections.

2.1. Companion matrices. Moler [31] suggested several decades ago that exploiting the properties of the companion matrix should lead to faster $\mathcal{O}(n^2)$ algorithms requiring only $\mathcal{O}(n)$ storage, but only in 2004 the first paper on this subject by Bini, Daddi, and Gemignani [10] was published. The observation that the successive QR iterates inherit particular rank structures has formed the basis of all future developments. Unfortunately, the explicit QR algorithm of Bini, et al. [10] suffers from numerical instabilities.

In 2007 Bini, Eidelman, Gemignani, and Gohberg [11] presented an explicit QR algorithm operating on an efficiently stored Hessenberg matrix. The matrix is written as the sum of a unitary plus low rank matrix, and both are stored in quasiseparable matrix format. In 2010 Bini, Eidelman, Gemignani, and Gohberg [9] presented an implicit version where the unitary matrix is now decomposed into smaller unitary factors. This was enhanced even further in 2012 by Boito, Eidelman, Gemignani, and Gohberg [16]. Numerical experiments suggest that the algorithm is backward stable, but no proof of backward stability is provided.

Meanwhile (2007) Chandrasekaran, Gu, Xia, and Zhu [18] published an implicit QR algorithm operating on a QR factorization of the companion matrix. The matrix Q was factored in smaller unitary matrices, and R was stored in sequentially semiseparable format, which equals the quasiseparable format in this case.

Van Barel, Vandebril, Van Dooren, and Frederix [35] presented in 2010 a novel representation based on three sequences of rotators and a vector. The implicit QR algorithm operates directly on the factorization, which is maintained under the QR algorithm.

Delvaux, Frederix, and Van Barel (2012) [20] present an algorithm for computing eigenvalues of block companion matrices via an implicit QR algorithm. The QR factorization of the block matrix is stored efficiently via the Givens-weight representation, but numerical round off slightly deteriorates the representation.

Aurentz, Mach, Vandebril, and Watkins [4] proposed in 2015 a new QR algorithm, which relies solely on unitary matrices and unitary operations. A proof of backward stability is presented and the numerical experiments illustrate the effectiveness and speed of this approach. We will build upon these results to develop the companion pencil solvers presented in this paper; moreover we will also rectify an error in the proof of backward stability from Aurentz et al. [4].

Recently the companion matrix was generalized to a factored form by Fiedler [24]. De Terán, Dopico, and Pérez [19] showed recently that, unfortunately, these novel linearizations do not lead to improved backward stability. The new proofs that we provide are in agreement with their optimal bounds.

A new approach to tackle the companion eigenvalue problem was proposed in 2015 by Bevilacqua, Del Corso, and Gemignani [7]. An initial similarity transformation is executed, transforming the companion matrix into a CMV-plus-rank-one matrix. An explicit QR algorithm exploiting this structure is presented, and the numerical experiments illustrate a high computational gain w.r.t. the implicit QR algorithm of Boito et al. [16]. We will also investigate this structure in this article, and prove that it fits the category of twisted QZ algorithms as introduced by Vandebril and Watkins [37,38].

To conclude we mention the monograph by Eidelman, Gohberg, and Haimovici (2013) [23] where the approach from Chandrasekaran et al. [18] is revisited. Another factorization based on three sequences of rotators [4, 35] is presented, but without numerical results.

For completeness we also mention non-unitary variants for computing the eigenvalues of the companion matrix. These methods are typically fast but potentially unstable, we refer to Zhlobich [42] and to Aurentz, Vandebril, and Watkins [2,3].

Amongst all non-companion rootfinders we would like to refer to the Ehrlich-Aberth based methods by Bini and Fiorentino [8,12], especially the multiple-precision code MPSolve [12].

2.2. Companion pencils. Consider the polynomial with complex coefficients $p(z) = a_n z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0$, with $a_n \neq 0$. The standard *companion pencil* is of the form (V, W) with V and W given by (1.1). One can show that $\det(zW - V) = p(z)$, so the eigenvalues of (V, W) coincide with the roots of $p(z)$.

Even though it is commonly known that the pencil approach has better backward stability properties, there is, to our knowledge, only one article by Boito, Eidelman, and Gemignani [14] presenting a structured QZ algorithm for computing roots of polynomials in the monomial basis. The authors consider a matrix pencil, say (V, W), where both V and W are of unitary-plus-rank-one form, V is Hessenberg and W is upper triangular. Both matrices are represented efficiently by a quasiseparable representation. To counter the effects of roundoff errors, some redundant quasiseparable generators to represent the unitary part are created; a compression algorithm to reduce the number of generators to a minimal set is presented. The computational cost of each structured QZ iteration is $\mathcal{O}(n)$. The authors run numerical experiments on the pencil (1.1) revealing that the backward error created via the pencil approach is significantly better. A double shift version of this algorithm is presented by Boito, Eidelman, and Gemignani in [15].

The companion pencil (1.1) is the most frequently appearing one in the literature, but, there is a wide variety of comparable matrix pencils with the same eigenvalues [6, 21, 29], many of which are highly structured. In this article we will focus on companion pencils of a slightly more general form.

THEOREM 2.1. *Consider a polynomial with complex coefficients* $p(z) = a_n z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0$. *Let* v, w *be vectors of length* n *with*

$$\begin{aligned} v_1 &= a_0, \\ v_{i+1} + w_i &= a_i, \ \textit{for } i = 1, \ldots, n-1, \textit{and} \\ w_n &= a_n. \end{aligned} \tag{2.1}$$

Then the determinant $\det(zW - V)$ *of the pencil* (V, W), *with*

$$V = \begin{bmatrix} 0 & & & -v_1 \\ 1 & & & -v_2 \\ & 1 & & -v_3 \\ & & \ddots & \vdots \\ & & 1 & -v_n \end{bmatrix} \quad \textit{and} \quad W = \begin{bmatrix} 1 & & & w_1 \\ & 1 & & w_2 \\ & & \ddots & \vdots \\ & & 1 & w_{n-1} \\ & & & w_n \end{bmatrix}, \tag{2.2}$$

coincides with $p(z)$.

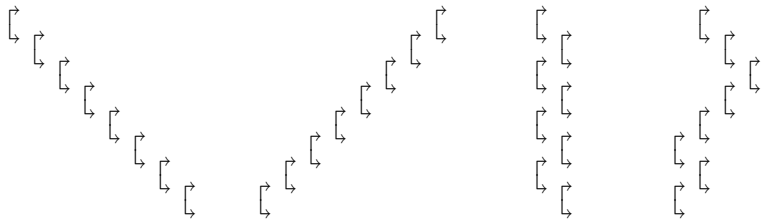
This theorem (see Aurentz et al. [5] or Mackey et al. [29]) allows for more flexibility than (1.1) in distributing the coefficients of the polynomial over the two pencil matrices. Also, V and W remain a Hessenberg-triangular pair of unitary-plus-rank-one form.

3. Core transformations and factorizations of companion matrices. Core transformations will be used throughout the paper and are the building blocks for a fast algorithm and an efficient representation of the companion pencil.

3.1. Core transformations. A nonsingular matrix G_i identical to the identity matrix except for a 2×2 submatrix in position $(i : i+1, i : i+1)$ is called a *core transformation*. The subscript i refers to the position of the diagonal block $(i : i+1, i : i+1)$ called the *active part* of the core transformation. Core transformations G_i and G_j commute if $|i - j| > 1$.

In previous work [2,3] we have used non-unitary core transformations to reduce the computational complexity. Here, however, we will only use unitary core transformations. Thus, in this paper, the term *core transformation* will mean *unitary* core transformation; the active part could be a rotator or a reflector, for example.

To avoid excessive index usage, and to ease the understanding of the interaction of core transformations, we depict them as , where the tiny arrows pinpoint the active part. For example, every unitary upper Hessenberg matrix Q can be factored as the product of $n-1$ core transformations in a *descending* order $Q = G_1 G_2 \cdots G_{n-1}$. Such a *descending sequence* of core transformations is represented pictorially by the first of the following four diagrams.



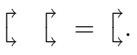
(3.1)

Upper Hessenberg | Lower Hessenberg | CMV | Irregular pattern

The other three diagrams depict unitary matrices of different types. The second is an *ascending sequence*, which represents a unitary lower Hessenberg matrix (which is also the inverse of a unitary upper Hessenberg matrix). Each of the four patterns represents the factorization of a unitary matrix into $n-1$ core transformations and can thus be considered as a generalization of unitary Hessenberg form [37]. We will use other factorizations later on in Section 5.

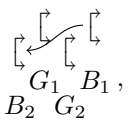
All of the algorithms in this paper are described in terms of core transformations and two operations: the *fusion* and the *turnover*.

Fusion. The product of two unitary core transformations G_i and H_i is again a unitary core transformation. Pictorially we can write this as

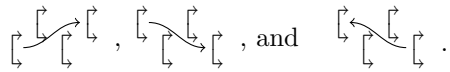


Turnover. The product of three core transformations $F_i G_{i+1} H_i$ is a 3×3 unitary matrix that can be factored also as $F_{i+1} G_i H_{i+1}$, depicted as

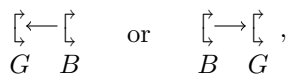
Pictorial action of a turnover and fusion. We will see, when describing the algorithms, that there are typically one or more core transformations not fitting the pattern (called the *misfit(s)*), that need to be moved around by executing turnovers and similarities. To describe clearly the movement of the misfit we use the following pictorial description:



where B_1 is the misfit before the turnover and B_2 after the turnover. The core transformations G_1 and G_2 are involved in the turnover and change once B_2 is created. The picture is mathematically equivalent to $G_1G_2B_1 = B_2\hat{G}_1\hat{G}_2$. Other possible turnovers are



Directly at the start and at the end of each QZ (and QR) iteration, a misfit is fused with another core transformation, so that it vanishes. We will describe this pictorially as



where B is to be fused with G.

3.2. A factorization of the companion pencil. The pencil matrices V and W from (1.3) are both unitary-plus-rank one, V is upper Hessenberg and W is upper triangular. We store V in QR decomposed form: $V = QR$, where Q is unitary and upper Hessenberg, and R is upper triangular and unitary-plus-rank-one. In fact

$$Q = \begin{bmatrix} 0 & & & & 1 \\ 1 & & & & 0 \\ & 1 & & & 0 \\ & & \ddots & & \vdots \\ & & & 1 & 0 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} 1 & & & & -v_2 \\ & 1 & & & -v_3 \\ & & \ddots & & \vdots \\ & & & 1 & -v_n \\ & & & & -v_1 \end{bmatrix}. \tag{3.2}$$

We need efficient representations of Q, R, and W. Q is easy; it is the product of $n-1$ core transformations: $Q = Q_1 \cdots Q_{n-1}$, with $Q_i(i:i+1, i:i+1) = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.

Factoring an upper triangular unitary-plus-rank-one matrix. The matrices R (3.2) and W (2.2) have exactly the same structure and can be factored in the same way. We focus on R. It turns out that for this factorization we need to add a bit of room by adjoining a row and column. Let

$$\tilde{R} = \left[\begin{array}{ccccc|c} 1 & & & & -v_2 & 0 \\ & 1 & & & -v_3 & 0 \\ & & \ddots & & \vdots & \vdots \\ & & & 1 & -v_n & 0 \\ & & & & -v_1 & 1 \\ \hline & & & & 0 & 0 \end{array}\right].$$

This is just R with a zero row and nearly zero column added. The 1 in the last column ensures that $\tilde{R}$ is unitary-plus-rank-one: $\tilde{R} = Y_n + xe_n^T$, where

$$Y_n = \left[\begin{array}{ccccc|c} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \vdots \\ & & & 1 & & \\ & & & & 0 & 1 \\ \hline & & & & 1 & 0 \end{array}\right] \quad \text{and} \quad x = -\left[\begin{array}{c} v_2 \\ v_3 \\ \vdots \\ v_n \\ v_1 \\ \hline 1 \end{array}\right].$$

Let $C_1, \ldots, C_n$ be core transformations such that $Cx = C_1 \cdots C_n x = \alpha e_1$, where $|\alpha| = \|x\|_2$. Let B be the unitary Hessenberg matrix $B = CY_n$. Clearly $B = B_1 \cdots B_n$, where $B_i = C_i$ for $i = 1, \ldots, n-1$, and $B_n = C_n Y_n$. This gives us a factorization of $\tilde{R}$ as

$$\tilde{R} = C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) = C^* (B + e_1 y^T), \tag{3.3}$$

with $y^T = \alpha e_n$. In the course of our algorithm, the core transformations B_i, C_i, and the vector y will be modified repeatedly, but the form (3.3) for $\tilde{R}$ will be preserved. $\tilde{R}$ remains upper triangular with its last row equal to zero. The theory that supports these claims can be found in [4, § 4]. Notice that the core transformations B_n and C_n both make use of row and column $n+1$. Had we not added a row and column, this factorization would not have been possible.

Multiplying (3.3) on the left by e_{n+1}^T, we find that $0 = e_{n+1}^T \tilde{R} = e_{n+1}^T C^* B + e_{n+1}^T C^* e_1 y^T$, so [4, Thm. 4.6]

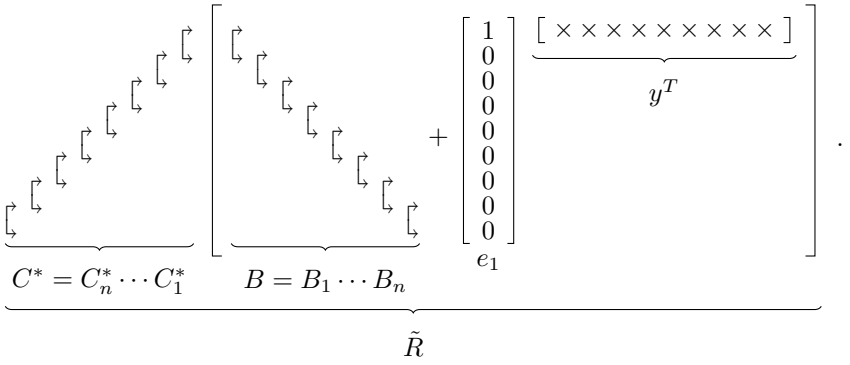
$$y^T = -(e_{n+1}^T C^* e_1)^{-1} e_{n+1}^T C^* B. \tag{3.4}$$

This equation demonstrates that the information about y, which determines the rank-one part, is encoded in the core transformations. This means that we will be able to develop an algorithm that does not keep track of y; the rank-one part can be simply ignored. If at any time we should need y or some part of y, we could recover it from C and B using (3.4). However, as we have shown in [4, § 4], it turns out that we never need to use (3.4) in practice.

Let $P = I_{(n+1)\times n}$, the $(n+1) \times n$ matrix obtained by deleting the last column from the $(n+1) \times (n+1)$ identity matrix. Then $R = P^T \tilde{R} P$, so our factored form of R is

$$R = P^T C_n^* \cdots C_1^* (B_1 \cdots B_n + e_1 y^T) P. \tag{3.5}$$

The matrices P and P^T are included just so that the dimensions of R come out right. They play no active role in the algorithm, and we will mostly ignore them. Pictorially, for $n = 8$, $\tilde{R}$ (and hence also R) can be represented as



Since we can ignore y, we see that R is represented by two sequences of core transformations. Hence $V = QR$ is represented by three sequences of core transformations.

The matrix W admits a factorization of the same form as (3.3)

$$W = P^T C_W^* (B_W + e_1 y_W^T) P,$$

with $C_W^* e_1 = [w_1, \ldots, w_n, 1]$.

Thus W is also represented by two sequences of core transformations. Altogether the pencil (V, W) is represented by five sequences of core transformations.

3.3. Core transformations and upper triangular matrices. In the next section we will describe how to compute the eigenvalues of the matrix pencil (V, W) via the QZ algorithm as described by Vandebril and Watkins [38]. An important operation is to refactor the product of an upper triangular matrix times a core transformation RG_i as the product of a core transformation times an upper triangular matrix[1] $\hat{G}_i\hat{R}$. The other way proceeds similarly. Considering dense matrices we get pictorially

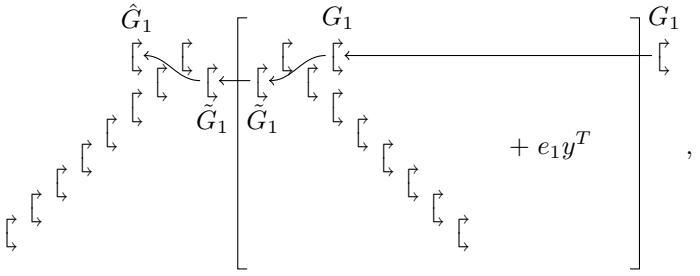
Applying a nontrivial core transformation from the right on the upper triangular matrix creates a non-zero subdiagonal, which can be removed by pulling out a nontrivial core transformation from the left.

In our case the upper triangular matrix is represented in a data-sparse way (3.3). Instead of explicitly creating the upper triangular matrix to interchange the order of the upper triangular matrix and a core transformation we operate on the factored form directly. There are two versions, *passing a core transformation* from right to left and vice versa.

From right to left. Consider an arbitrary upper triangular unitary-plus-rank-one matrix, say, $R = P^TC^*(B+e_1y^T)P$. Let G_i $(1 \le i \le n-1)$ be the core transformation we want to move from right to left. Slightly abusing notation, we have $PG_i = G_iP$. This is not quite correct because the G_i to the right of P is $n \times n$, while the G_i to the left of P is $(n+1) \times (n+1)$. We have $RG_i = P^TC^*(B + e_1y^T)G_iP = P^TC^*(BG_i + e_1y^TG_i)P$. So y^T changes into $\hat{y}^T = y^TG_i$. The next step is to execute a turnover $B_iB_{i+1}G_i = \tilde{G}_{i+1}\hat{B}_i\hat{B}_{i+1}$. We now have $RG_i = P^TC^*(\tilde{G}_{i+1}\hat{B}+e_1\hat{y}_1^T)P = P^TC^*\tilde{G}_{i+1}(\hat{B} + e_1\hat{y}_1^T)P$. In the last step we have used the fact that $\tilde{G}_{i+1}e_1 = e_1$ because $i+1 > 1$. To complete the procedure we just need to do one more turnover, in which $\tilde{G}_{i+1}$ interacts with C_i^* and C_{i+1}^*. Specifically $C_{i+1}^*C_i^*\tilde{G}_{i+1} = \hat{G}_i\hat{C}_{i+1}^*\hat{C}_i^*$, resulting in $RG_i = P^T\hat{G}_i\hat{C}^*(\hat{B} + e_1\hat{y}^T)P$. Abusing notation again we have $P^T\hat{G}_i = \hat{G}_iP^T$. Here it is important that $i \le n-1$. Finally $RG_i = \hat{G}_iP^T\hat{C}^*(\hat{B} + e_1\hat{y}^T)P = \hat{G}_i\hat{R}$.

The total computational effort required for the task is just two turnovers. The operation $\hat{y}^T = y^TG_i$ is not actually performed, as we do not keep track of y.

Pictorially for $n = 8$ and $i = 1$, we have

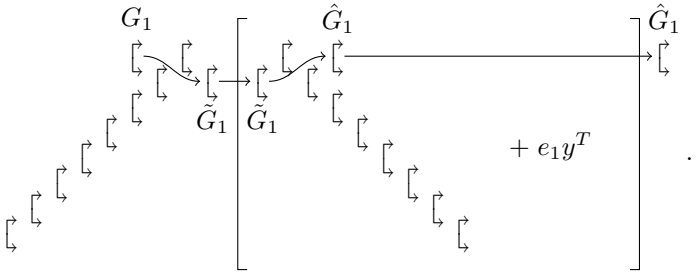


where we have ignored P and P^T and also the action on y^T.

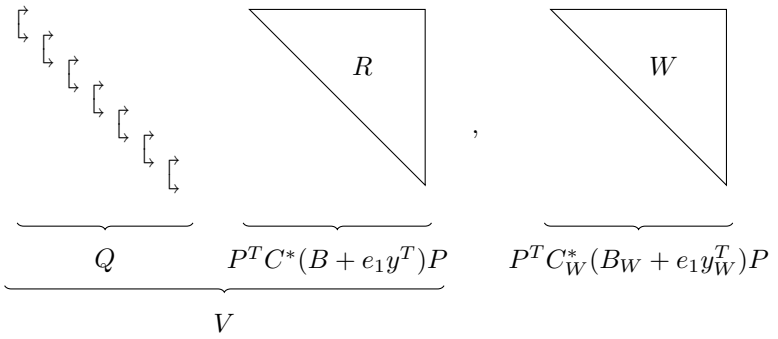
[1] We assume that eigenvalues at zero or infinity are deflated beforehand such that the involved upper triangular matrices are nonsingular.

Notice that when we apply a core transformation G_{n-1}, we temporarily create $\tilde{G}_n$, which makes use of row/column $n+1$. Here we see that the existence of an extra row/column is crucial to the functioning of the algorithm.

From left to right. The procedure to change $G_iR = \hat{R}\hat{G}_i$ is just the reverse of the previous procedure. We omit the details but depict it pictorially for $n = 8$ and $i = 1$



It is clear now that one can move a single core transformation through a factored upper triangular matrix in either direction by executing only two turnovers. From now on, to ease the notation, we will depict our Hessenberg-triangular pencil in a simpler format:



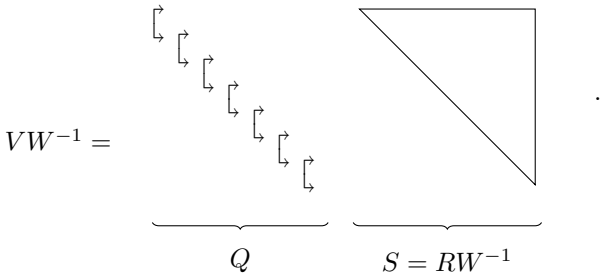
(3.6)

where we have replaced each upper triangular factor by a triangle. With this description, we can immediately apply the algorithms from Vandebril and Watkins [38]. For completeness, however, we will redescribe the flow of a single QZ step.

4. The companion QZ algorithm. We have implemented both single-shift and double-shift companion QZ algorithms. For simplicity we will describe only the single-shift case, as the double-shift iteration is a straightforward extension; we refer to Aurentz et al. and Vandebril and Watkins [4,38]. The companion QZ algorithm is easily described by viewing it as a version of the companion QR algorithm applied to the matrix VW^{-1}. Clearly

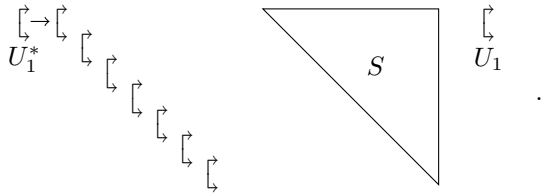
$$VW^{-1} = QRW^{-1} = QS,$$

where $S = RW^{-1}$ is upper triangular. Pictorially

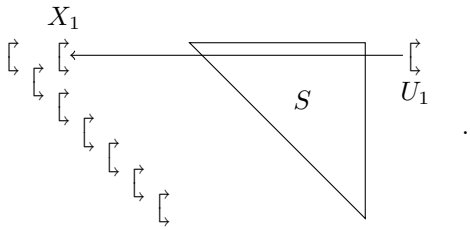


Of course we will find in the end that we do not have to invert W in practice.

To begin the iteration we select a suitable shift μ and compute $q = (V - \mu W)e_1$. Only the first two entries of q are nonzero, so we can construct a core transformation U_1 such that $U_1^* q = \alpha e_1$ for some α. Our first modification to VW^{-1} is to apply a similarity transformation by U_1:



We can immediately fuse U_1^* with Q_1 to make a new Q_1. (To keep the notation under control, we do not give the modified Q_1 a new name; we simply call it Q_1.) We can also pass U_1 through S to obtain

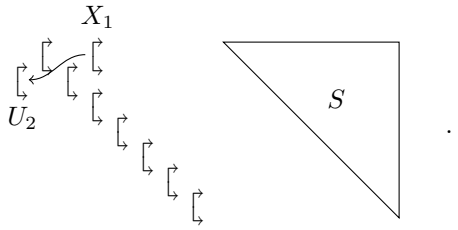


(4.1)

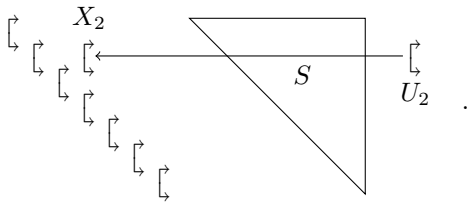
The details of passing a core transformation through S will be discussed below.

If we were to multiply the factors together, we would find that the matrix is no longer upper Hessenberg; there is a bulge in the Hessenberg form caused by a nonzero entry in position $(3, 1)$. The standard Francis algorithm chases the bulge until it disappears off the bottom of the matrix. In our current setting we do not see a bulge. Instead we see an extra core transformation X_1 in (4.1), which is in fact the cause of the bulge. X_1 is the *misfit*. Instead of chasing the bulge, we will chase the misfit through the matrix until it disappears at the bottom. We therefore call this a *core chasing* algorithm.

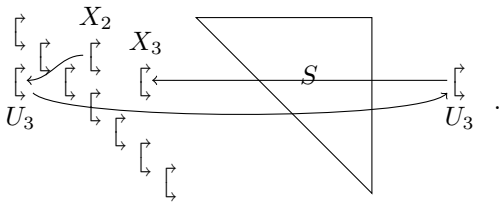
Proceeding from (4.1), the next step is to do a turnover $Q_1 Q_2 X_1 = U_2 \hat{Q}_1 \hat{Q}_2$. Core transformations $\hat{Q}_1$ and $\hat{Q}_2$ become the new Q_1 and Q_2. Pictorially



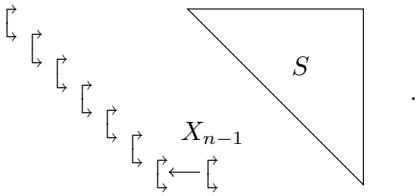
Next we do a similarity transformation, multiplying by U_2^* on the left and U_2 on the right. This has the effect of moving U_2 from the left side to the right side of the matrix. We can also pass U_2 through S to obtain



Now we are in the same position as we were at (4.1), except that the misfit has moved downward one position. The process continues as before:

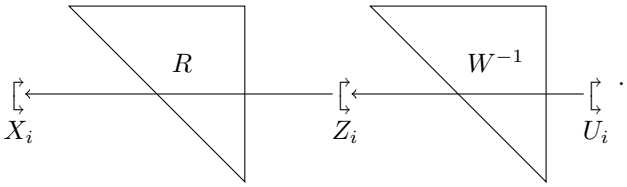


After $n-1$ such steps we arrive at



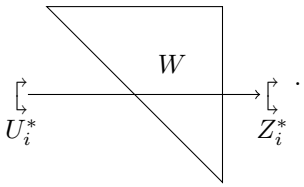
We can now fuse X_{n-1} with Q_{n-1}, completing the iteration.

To complete the description, we now consider the details of passing a core transformation through S. Since $S = RW^{-1}$, the operation has two stages:



(4.2)

Since we do not wish to invert W, we do not literally execute the operation depicted on the right of (4.2). Instead we do the equivalent operation



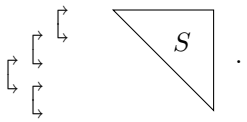
Since both R and W are unitary-plus-rank-one matrices stored in the factored form described in Section 3, each of the two stages costs two turnovers. Thus the computational cost of passing a core transformation through S is four turnovers.

5. The twisted companion QZ algorithm. So far we have focused on pencils (V, W) for which V and its factor Q are in upper Hessenberg form, and Q is stored as a descending sequence of core transformations: $Q = Q_1 Q_2 \cdots Q_{n-1}$. This is by no means necessary; one can equally well work with an ascending sequence $Q_{n-1} \ldots Q_1$, which is lower Hessenberg, or a sequence $Q_{i_1} \ldots Q_{i_{n-1}}$, where $i_1, \ldots, i_{n-1}$ is any permutation of $1, \ldots, n-1$. For example, the CMV pattern is $Q_1 Q_3 Q_5 \cdots Q_2 Q_4 Q_6 \cdots$, in which the odd numbered transformations are to the left of the even numbered ones. These factorizations are depicted in (3.1). We call them *twisted factorizations*.

We have introduced in 2011 [39] a *twisted* QR algorithm that acts on a matrix

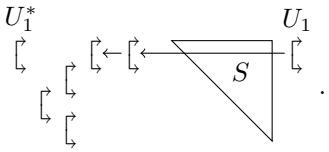
$$A = Q_{i_1} \cdots Q_{i_{n-1}} R,$$

in which $Q = Q_{i_1} \cdots Q_{i_{n-1}}$ is in an arbitrary twisted form (R remains upper triangular). From this we obtain the *twisted* companion QZ algorithm by replacing R by $S = RW^{-1}$, where R and W are stored in the factored form described in Section 3. See Vandebril and Watkins [39] for a detailed description of twisted QR iterations of arbitrary degree. Here we will just consider one simple example to show how the algorithm works in the single-shift case, i.e. iterations of degree one. Consider a 5×5 example in the factored form $VW^{-1} = Q_3 Q_2 Q_1 Q_4 S$:

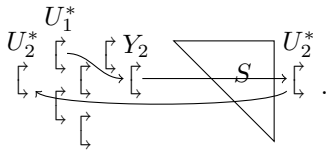


(5.1)

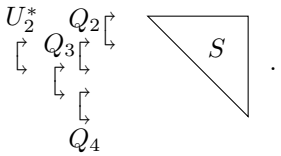
Notice that $Q_3 Q_2 Q_1 Q_4 = Q_3 Q_4 Q_2 Q_1 = Q_3 Q_2 Q_4 Q_1$. All that really matters is that Q_1 is to the right of Q_2, which is to the right of Q_3, which is to the left of Q_4. That information is conveyed concisely in (5.1). How the iteration is set in motion depends on the relative positions of Q_1 and Q_2. If Q_1 were to the left of Q_2, we would start the iteration exactly as we did in the upper Hessenberg case described above. Given a suitable shift μ, we would compute $q = (V - \mu W)e_1$. In our current example, Q_1 is to the right of Q_2, and we compute $q = (V - \mu W)V^{-1}e_1$ instead. We then proceed as before: We build a core transformation U_1 such that $U_1^* q = \alpha e_1$, and do a similarity transformation by U_1:



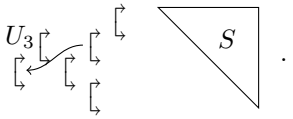
We can immediately pass U_1 through S and fuse it with Q_1, leaving U_1^* as the misfit. We chase the misfit to the right by performing a turnover to form Y_2, then passing Y_2 through S to form U_2^*. We then move U_2^* to the left side by a similarity transformation:



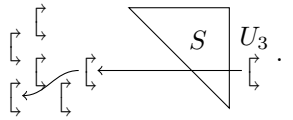
Notice that we are chasing the misfit in the opposite direction from the upper Hessenberg case. Our situation is now



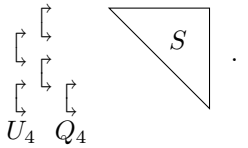
If Q_4 were to the left of Q_3, we would continue in this direction. But Q_4 is to the right of Q_3, so it blocks us from continuing. The twist in the factorization forces us to change directions! Q_2 becomes the new misfit, and we chase it to the left:



A turnover forms U_3, which we move to the right by a similarity transformation, and we continue as in the upper Hessenberg case:



Once we get to the bottom, we have



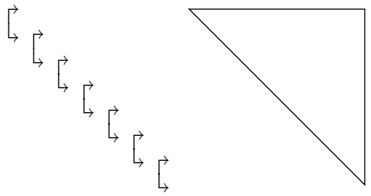
Here we have a choice. We can move U_4 to the right by a similarity transformation, pass it through S, then fuse it with Q_4 to form the final Q_4 to the right of Q_3. Alternatively, we can pass Q_4 through S, move it to the left by a similarity transformation, and fuse it with U_4 to form the final Q_4 to the left of Q_3. Thus there are two possible final patterns for Q:

To summarize, we chase a misfit core transformation through the factored matrix.

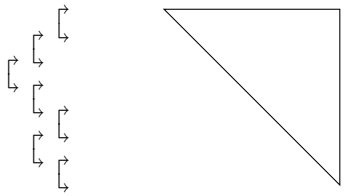
The chasing direction depends on the pattern of the twisted factorization; whenever we come to a twist in the factorization, we must change direction. The twist pattern does not remain static. Each twist moves up by one position on each iteration. At the end of the iteration we have the choice of how to continue the pattern. That is, we can place Q_{n-1} either to the left or to the right of Q_{n-2}. This is all explained in much greater detail by Vandebril and Watkins [38].

6. Initializing the matrix to an arbitrary twisted form. Our example shows how the twisted shape of the Q sequence evolves from one iteration to the next. Now we will show how to initialize the sequence to put it into an arbitrary desired twisted shape. This procedure is a generalization of the one used by Bevilacqua, Del Corso, and Gemignani [7] to transform the companion Hessenberg matrix immediately to the CMV form.

Let us illustrate the existence of this similarity transformation by a simple example. Suppose a companion matrix A is given in Hessenberg form ($n = 8$)

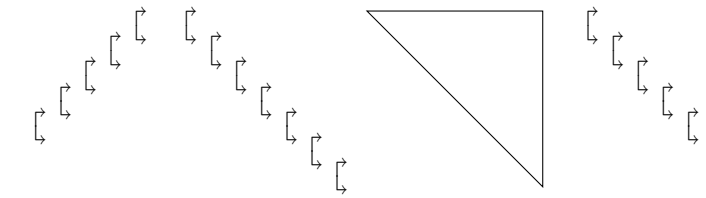
$A = QR =$  (6.1)

and we would like to execute an initial similarity transformation to the form

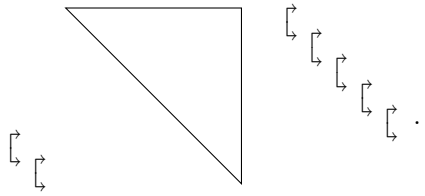
 . (6.2)

The algorithm proceeds from bottom up. Comparing (6.1) with (6.2), we see that the bottom two core transformations are positioned correctly, so we leave them alone. We execute a similarity transformation $U_1^* A U_1$, where U_1 contains the first five core transformations of the Q sequence. As a result the first five core transformations on the left are annihilated and appear on the right

$U_1^* A U_1 =$

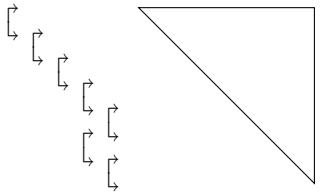


$=$



Passing now the core transformations on the right through the upper triangular matrix we get

$U_1^* A U_1 =$

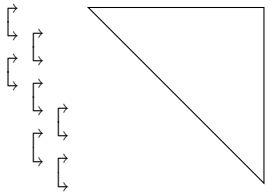


. (6.3)

As a result we see that the bottom three core transformations of (6.3) match the shape of (6.2).

This procedure can be repeated easily. Only the upper two core transformations are not yet in the desired positions. Executing a similarity transformations with U_2 equal to these first two core transformations leads to

$U_2^* U_1^* A U_1 U_2 =$



.

Finally after executing a similarity transformation with the top core transformation we are done.

The actual implementation of this method is much simpler and cheaper than one might expect. In fact nothing needs to be done, one just needs to reposition the factors in the Q sequence. A closer look at the upper triangular matrix R, initially of the form

$$R = \begin{bmatrix} 1 & & & -v_2 \\ & 1 & & -v_3 \\ & & \ddots & \vdots \\ & & 1 & -v_n \\ & & & -v_1 \end{bmatrix}$$

reveals that all core transformations Q_i, for $i = 1, \ldots, n-2$ that need to be passed through R do not change. We remark that even though the Q_i's do not change, the column vector containing the v_i's will be different. As a result, a simple permutation of the Q_i's and some updating of the v_i's provides us the desired format, and this is exactly what was done by Bevilacqua et al. [7].

7. Stability. Van Dooren and Dewilde [36] were the first to investigate the backward stability of polynomial root-finding via companion matrices and pencils. Edelman and Murakami [22] revisited this analysis, focusing on scalar polynomials. Jónsson and Vavasis [26] presented a clear summary of these results.

There are two important measures of backward accuracy when dealing with companions: the backward error on the companion matrix or pencil and the backward error on the coefficients of the original polynomial. Van Dooren and Dewilde [36] showed that pushing the error further back from the pencil (or matrix) to the polynomial coefficients introduced an additional factor in the upper bound of the order of the norm of the coefficients.

We illustrated [4] that the backward error on the polynomial coefficients scales quadratically with the norm of the coefficients. To prove this we relied on the analysis of Van Dooren and Dewilde and proved that the backward error on the companion matrix scales linearly with the polynomial coefficients. Unfortunately we realized later on that a factor was forgotten in the proof; as a result the backward error on the companion matrix scales quadratically instead. Moreover, this implies that we can not rely on the result from Van Dooren and Dewilde anymore as this would provide a bound with a cubic dependence and this is not tight. In this section we will rectify the proof. As our error is highly structured we are able to provide an analysis not introducing the additional factor. This is in agreement with the observations made in the numerical experiments by Aurentz et al. [4].

In this section we will first map the error back to the original companion matrix or pencil. In a second step we will push this error even further back to the coefficients of the original polynomial. We use the following conventions in this section: $\doteq$ denotes an equality where all, or some, higher order terms are dropped, $\lesssim$ stands for less than or equal up to a modest constant possibly depending on m and d, $\approx$ denotes equal up to a modest constant.

We assume in the remainder of the section that $\|a\| \geq 1$, with a the vector of coefficients of $p(\lambda)$. This implies that $\|a\|$ approximates the norm(s) of the rank-one part(s) in the companion matrix (pencil). If the norm of a would be small, the construction in Section 3.2 would imply that $\|y\| \approx 1$, and one would need to replace all norms $\|a\|$ in the theorems by 1.

7.1. Backward error on the companion matrix and pencil. The algorithms, in both the pencil and the matrix case, operate only on sequences of core transformations. These operations are backward stable and therefore fusions and turnovers only introduce errors of the order of the machine precision ϵ_m. The most sensitive step is reconstructing the rank-one part from the two sequences of core transformations. As a result we have that the backward error on a single upper triangular unitary-plus-rank-one-matrix scales quadratically w.r.t. the matrix's norm under unitary equivalence transformations.

THEOREM 7.1. *Let $\hat{R}$ be the result of a unitary equivalence transformation executed in finite precision on a factored (see Section 3.2) upper triangular unitary-plus-rank-one matrix R. Then we have that $\hat{R} = X^*(R+E)Z$, where $\|E\|/\|R\|^2 \lesssim \epsilon_m$.*

Proof. The matrix $R = C^*(B + e_1 y^T)$ is upper triangular[2] and the rank one part can be reconstructed (3.4) from the unitary factors as $y^T = -(e_{n+1}^T C^* e_1)^{-1}(e_{n+1}^T C^* B)$. Executing the unitary equivalence transformation would lead, in exact arithmetic, to

$$\tilde{R} = X^* R Z = X^* C^* Y (Y^* B Z + Y^* e_1 y^T Z) = \tilde{C}^*(\tilde{B} + e_1 \tilde{y}^T),$$

where $\tilde{C} = Y^* C X$, $\tilde{B} = Y^* B Z$, and $\tilde{y}^T = y^T Z$. Note that an additional Y is introduced, with $Y e_1 = e_1$, as a consequence of the implementation based on turnovers. When the operations are carried out in finite precision we have $\hat{C} = Y^*(C + \delta C)X$ and $\hat{B} = Y^*(B + \delta B)Z$ instead of $\tilde{C}$ and $\tilde{B}$. Turnovers are backward stable and therefore $\|\delta B\| \approx \|\delta C\| \approx \varepsilon_m$.

The vector y is never modified directly, but obtained implicitly from the unitary matrices. The exact $\tilde{y}$ and computed $\hat{y}$ satisfy

$$\tilde{y}^T = -(e_{n+1}^T \tilde{C}^* e_1)^{-1} e_{n+1}^T \tilde{C}^* \tilde{B}, \qquad \hat{y}^T = -(e_{n+1}^T \hat{C}^* e_1)^{-1} e_{n+1}^T \hat{C}^* \hat{B}.$$

Let us set $\tilde{\rho} = e_{n+1}^T \tilde{C}^* e_1$ and $\hat{\rho} = e_{n+1}^T \hat{C}^* e_1$. By the assumption on the error on $\hat{C}$ we have $\hat{\rho} = e_{n+1}^T X^*(C + \delta C)^* Y e_1 = \tilde{\rho}(1 + \delta\rho)$ with $|\delta\rho| \lesssim \tilde{\rho}^{-1}\varepsilon_m$. Because $Y e_1 = e_1$ and $e_{n+1}^T X^* = e_{n+1}^T$ we have $\tilde{\rho} = \rho$ and by construction, $\rho = \|y\|_2^{-1}$. Moreover, for tiny $\delta\rho$ we get $(1 + \delta\rho)^{-1} \doteq 1 - \delta\rho$, where $\|\delta\rho\| \lesssim \|y\|_2 \epsilon_m \approx \|R\|\epsilon_m$. Combining all we get

$$\hat{y}^T \doteq -\rho^{-1}(1 - \delta\rho) e_{n+1}^T X^*(C + \delta C)^*(B + \delta B)Z \doteq (y + \delta y)^T Z,$$

where $\|\delta y\| \lesssim \|y\|_2^2 \epsilon_m \approx \|R\|^2 \epsilon_m$.

We can then write the total backward error $X(\hat{R} - \tilde{R})Z^*$ as (setting $\delta C^* e_1 = \delta x$)

$$\begin{aligned} X(\hat{R} - \tilde{R})Z^* &= (C + \delta C)^*((B + \delta B) + e_1(y + \delta y)^T) - C^*(B + e_1 y^T) \\ &\doteq \delta C^* B + C^* \delta B + \delta x\, y^T + x\, \delta y^T, \end{aligned} \tag{7.1}$$

where $\|\delta x\| \lesssim \epsilon_m$, which proves the bound. ∎

THEOREM 7.2 (Backward error on the companion matrix). *Let $\hat{A}$ be the result of a QR step as described by the algorithm from Aurentz et al. [4], starting from the companion matrix A. Then we have that $\hat{A} = U^*(A + E)U$, where $\|E\|/\|A\|^2 \lesssim \epsilon_m$.*

Proof. The matrix A is represented by its QR factorization, and R is stored efficiently by means of two sequences of core transformations $A = QR = QC^*(B + e_1 y^T)$. Executing a similarity leads to $\tilde{A} = U^* Q X X^* C^*(B + e_1 y^T)U$.

By Theorem 7.1 and the fact that turnovers are backward stable we get that the actual computed $\hat{A}$ equals

$$\hat{A} = U^*(Q + \delta Q)(C + \delta C)^*((B + \delta B) + e_1(y + \delta y)^T)U,$$

with $\|\delta y\| \lesssim \|R\|^2 \epsilon_m = \|A\|^2 \epsilon_m$ and the $\delta Q, \delta C$, and δB bounded in norm by a small multiple of the machine precision. The rest follows easily. ∎

Looking at the structure of the matrix A (1.2), we see that a tiny a_n can lead to a possibly large perturbation. This is why the backward error provided by the companion pencil is typically better.

THEOREM 7.3 (Backward error on the companion pencil). *Let $\hat{V} - \lambda\hat{W}$ be the the result of a QZ step as described by the algorithm from Section 5, starting from the companion pencil $V - \lambda W$. Then we have that $\hat{V} - \lambda\hat{W} = U^*\left((V + \delta V) - \lambda(W + \delta W)\right)Z$, where $\|\delta V\|/\|V\|^2 \lesssim \epsilon_m$ and $\|\delta W\|/\|W\|^2 \lesssim \epsilon_m$.*

[2] For simplicity we omit the projection operators P and P^T; take into consideration that y is slightly enlarged as well in the representation.

Proof. Analogous to the proof of Theorem 7.2. ∎

Even though we did not emphasize the structure of the backward error in the theorems, we can see from the formula describing the error (7.1) that the largest perturbation is introduced in the vector of coefficients y. Based on this observation, which holds for both the companion matrix and the pencil case, we will be able to map the error back to the coefficients without introducing an additional factor.

We note that the errors introduced by both the QR and the QZ algorithm link directly to the companion matrix and companion pencil. Problems arise in the companion matrix case when its norm gets large due to a tiny leading coefficient. In the pencil case one can distribute the coefficients over the vectors v and w to get almost equal norms for V and W, after which one can balance them to get norm approximately 1.

7.2. Backward error on the polynomial coefficients. We have established that the norm of the backward error on the companion pencil depends quadratically on the norm of the involved matrices. According to the work of Edelman and Murakami [22] this, in general, gives a backward error on the polynomial coefficients depending cubically on the polynomial coefficients.

LEMMA 7.4 (Edelman and Murakami [22]). *Let $V - \lambda W$ be the companion linearization of a monic polynomial $p(\lambda)$, and a the vector of its coefficients. Then, if $\|\delta V\|, \|\delta W\| \leq \epsilon$, we have that the eigenvalues of $V + \delta V - \lambda(W + \delta W)$ are the roots of a monic polynomial $\tilde{p}(x)$ with coefficient vector $\tilde{a}$ such that*

$$\|a - \tilde{a}\| \lesssim \max\{\|V\|, \|W\|\}\,\epsilon.$$

Of course, in case of the companion matrix, Lemma 7.4 simplifies since $W = I$ and $\delta W = 0$. Typically in the pencil case $\|V\|$ and $\|W\|$ have norm approximately equal to $\|a\|$ so one gets a cubic dependency on $\|a\|$. In the QR case (1.2), the situation is slightly different since the companion matrix A has its norm approximately equal to $\|a\|/|a_n|$ creating sensitivity w.r.t. the size of $|a_n|$ and a cubic dependency on $\|a\|/|a_n|$.

In this section we prove that, in our case, the additional factor does not show up because of the highly structured backward error. We provide some general results to analyze the backward error on the polynomial coefficients when computing the roots via a structured method operating on a linearization.

In the following we will need to measure norms of matrix polynomials, so we define $\|P(\lambda)\|$, for a degree d matrix polynomial $P(\lambda) = \sum_{j=0}^{d} P_j \lambda^j$, as follows:

$$\|P(\lambda)\| = \max_{j=0,\ldots,d} \|P_j\|. \tag{7.2}$$

In our case the pencil $V - \lambda W = Z(\lambda) + x(\lambda)y^T$, where

$$Z(\lambda) = \begin{bmatrix} -\lambda & & & 1 \\ 1 & -\lambda & & \\ & \ddots & \ddots & \\ & & 1 & -\lambda \end{bmatrix}, \; x(\lambda) = \begin{bmatrix} (v_0+1)/\alpha & \lambda w_1/\beta \\ v_1/\alpha & \lambda w_2/\beta \\ \vdots & \vdots \\ v_{n-2}/\alpha & \lambda w_{n-1}/\beta \\ v_{n-1}/\alpha & \lambda(w_n - 1)/\beta \end{bmatrix}, y = -\begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ \alpha & \beta \end{bmatrix},$$

with α and β chosen such that the columns of $x(\lambda)$ have unit norms.

LEMMA 7.5. *The matrix pencil $Z(\lambda)+x(\lambda)y^T$ is a linearization for the polynomial*

$$p(\lambda) = \det Z(\lambda) + y^T \operatorname{adj} Z(\lambda) x(\lambda).$$

Proof. We verify that $\det(Z(\lambda) + x(\lambda)y^T) = p(\lambda)$. By the Sherman-Morrison determinant formula we get:

$$\begin{aligned}\det(Z(\lambda) + x(\lambda)y^T) &= \det Z(\lambda)\det(I + Z(\lambda)^{-1}x(\lambda)y^T)\\ &= \det Z(\lambda)(1 + y^T Z(\lambda)^{-1}x(\lambda)).\end{aligned}$$

and using the relation $Z^{-1}\det(Z) = \operatorname{adj}(Z)$ yields that $Z(\lambda) + x(\lambda)y^T$ linearizes

$$p(\lambda) = \det Z(\lambda) + y^T \operatorname{adj} Z(\lambda)x(\lambda).$$

Since the right-hand side is a polynomial which coincides with $p(\lambda)$ at all the points where $Z(\lambda)$ is non-singular, we have that the equality holds for all λ. □

If we have a perturbed pencil, i.e. Z, x, and y perturbed with $\delta Z, \delta x$, and δy, respectively then we have that this pencil linearizes $\tilde{p}(\lambda) = \det(Z + \delta Z)(\lambda) + (y + \delta y)^T \operatorname{adj}(Z + \delta Z)(\lambda)\,(x + \delta x)(\lambda)$.

To measure the difference between $\tilde{p}(\lambda)$ and $p(\lambda)$ we rely on a result by Stewart [34] providing a bound for the conditioning of the adjugate operator

$$\kappa_{\mathrm{adj}}(M) = \limsup_{\|\delta M\|\to 0} \frac{\|\operatorname{adj}(M + \delta M) - \operatorname{adj}(M)\|}{\|\delta M\|},$$

leading to the first order bound

$$\|\operatorname{adj}(M + \delta M) - \operatorname{adj}(M)\| \le \kappa_{\mathrm{adj}}(M)\cdot\|\delta M\| + O(\|\delta M\|^2).$$

LEMMA 7.6 (Stewart). *Let M be a square matrix. The conditioning of the adjugate operator for the 2-norm at M satisfies*

$$\kappa_{adj}(M) \lesssim \frac{\sigma_1}{\sigma_{n-1}},$$

where $\sigma_1 > \ldots > \sigma_n$ are the singular values of M.

This result tells us that if we perturb a matrix M by δM, the change in its adjugate will be bounded by the ratio between the first and penultimate singular value. However, in our case we are dealing with a matrix polynomial and we need to be precise about how to measure perturbations. According to (7.2) we have defined the norm of a matrix polynomial as the maximum over the norm of its coefficients, so we have $\|P(\lambda) - Q(\lambda)\| = \max_j \|P_j - Q_j\|$, where P_j and Q_j are the coefficients of $P(\lambda)$ and $Q(\lambda)$. With this in mind we define the conditioning of the adjugate for a matrix pencil $M(\lambda)$ as

$$\kappa_{\mathrm{adj}}(M(\lambda)) = \limsup_{\|\delta M(\lambda)\|\to 0} \frac{\|\operatorname{adj}(M(\lambda) + \delta M(\lambda)) - \operatorname{adj}(M(\lambda))\|}{\|\delta M(\lambda)\|}. \tag{7.3}$$

For our purpose (cfr. $Z(\lambda)$) we can impose normality, monicity, and a constraint on the eigenvalues of the matrix pencil $M(\lambda)$ to get an explicit bound on the conditioning.

LEMMA 7.7. *Let $M(\lambda)$ be an $n \times n$ normal monic matrix pencil with simple finite eigenvalues $\lambda_1, \ldots, \lambda_n$. Then the adjugate matrix polynomial* $\operatorname{adj}(M(\lambda))$ *has degree (at most) $n-1$ and the conditioning of the adjugate operator is bounded by:*

$$\kappa_{\mathrm{adj}}(M(\lambda)) \lesssim \frac{1 + \|M(\lambda)\|}{\min_{s\neq k}|\lambda_k - \lambda_s|}.$$

Proof. The degree of $\operatorname{adj}(M(\lambda))$ follows, since each entry of $\operatorname{adj}(M(\lambda))$ is the determinant of an $(n-1)\times(n-1)$ pencil. As $M(\lambda)$ is normal, its singular values are just the moduli of its eigenvalues. Let ξ_n be a primitive n-th root of unity, then, for all $j = 1, \dots, n$, we have $\sigma_1(M(\xi_n^j)) \leq 1 + \| M(\lambda) \|$.

Being $M(\xi_n^j)$ normal, $\sigma_n(M(\xi_n^j))$ and $\sigma_{n-1}(M(\xi_n^j))$ are the moduli of the smallest and second smallest eigenvalue of $M(\xi_n^j)$ respectively. If k is the index that realizes $\sigma_n(M(\xi_n^j)) = |\xi_n^j - \lambda_k|$, then for all $s \neq k$ we have $|\lambda_k - \lambda_s| \leq 2|\xi_n^j - \lambda_s|$. Taking the minimum over all $s \neq k$ gives us $\frac{1}{2}\min_{s\neq k}|\lambda_k - \lambda_s| \leq \sigma_{n-1}(M(\xi_n^j))$.

Using the bounds for $\sigma_1(M(\xi_n^j))$ and $\sigma_{n-1}(M(\xi_n^j))$ yields

$$\kappa_{\text{adj}}(M(\xi_n^j)) \lesssim \frac{1 + \| M(\lambda) \|}{\min_{s\neq k} |\lambda_k - \lambda_s|}.$$

Consider now a small linear perturbation $\delta M(\lambda)$ to the matrix pencil $M(\lambda)$. We want to measure the difference in norm of each coefficient of $\operatorname{adj}(M(\lambda) + \delta M(\lambda)) - \operatorname{adj}(M(\lambda))$. We can recover the coefficients from the evaluations in the roots of unity by performing an inverse discrete Fourier transform. More precisely, let F be the $n \times n$ Vandermonde matrix built using the n-th roots of unity as nodes. Then we have that the block vector of coefficients of $\delta A_0 + \delta A_1 \lambda + \cdots + \delta A_{n-1}\lambda^{n-1} = \operatorname{adj}(M(\lambda) + \delta M(\lambda)) - \operatorname{adj}(M(\lambda))$ is given by:

$$\begin{bmatrix} \delta A_0 \\ \vdots \\ \delta A_{n-1} \end{bmatrix} = \frac{1}{n}(F^H \otimes I) \left(\begin{bmatrix} \operatorname{adj}(M(1) + \delta M(1)) \\ \vdots \\ \operatorname{adj}(M(\xi_n^{n-1}) + \delta M(\xi_n^{n-1})) \end{bmatrix} - \begin{bmatrix} \operatorname{adj}(M(1)) \\ \vdots \\ \operatorname{adj}(M(\xi_n^{n-1})) \end{bmatrix} \right).$$

The above implies the following first order bound ($\|\delta M(\xi_n^j)\| \leq 2\|\delta M(\lambda)\|$):

$$\| \delta A_j \| \leq 2 \| \delta M(\lambda) \| \max_j \kappa_{\text{adj}}(M(\xi_n^j)) \lesssim \frac{(1 + \| M(\lambda) \|)\| \delta M(\lambda) \|}{\min_{s\neq k} |\lambda_k - \lambda_s|} + O(\| \delta M(\lambda) \|^2).$$

Taking the limit for $\| \delta M(\lambda) \| \to 0$ in (7.3) concludes the proof. ∎

The matrix pencil $Z(\lambda)$ is normal, monic, has norm 1, and its eigenvalues are exactly the n-th roots of the unity. Since the distance between two n-th roots of unity is at least $\frac{4}{n}$ we have that $\kappa_{\text{adj}}(Z(\lambda))$ is bounded by a polynomial in n and independent of the roots. Following our conventions we can thus write $\kappa_{\text{adj}}(Z(\lambda)) \lesssim 1$. We can finally state our desired result of the perturbation analysis.

THEOREM 7.8. *Let $A(\lambda) = Z(\lambda) + x(\lambda)y^T$ a companion pencil as defined in (7.2). Let ϵ_Z, ϵ_x and ϵ_y be positive numbers, and $\delta Z(\lambda), \delta x(\lambda), \delta y$ a matrix pencil, and two vectors such that $\| \delta Z(\lambda) \| \leq \epsilon_Z$, $\| \delta x(\lambda) \| \leq \epsilon_x$, and $\| \delta y \| \leq \epsilon_y$. Let $p(\lambda) + \delta p(\lambda)$ be the polynomial linearized by $Z(\lambda) + \delta Z(\lambda) - (x + \delta x)(\lambda)(y + \delta y)^T$. Then we have the following first order bound*

$$\| \delta p \| \lesssim \epsilon_Z + \| x(\lambda) \| \epsilon_y + \| y \| \epsilon_x + \| x(\lambda) \| \| y \| \epsilon_Z.$$

Proof. We can write $p(\lambda) + \delta p(\lambda)$ as

$$p(\lambda) + \delta p(\lambda) = \det(Z(\lambda) + \delta Z(\lambda)) - (y + \delta y)^T \operatorname{adj}(Z(\lambda) + \delta Z(\lambda))(x + \delta x).$$

We will bound all the terms separately. First, notice that $\det(Z(\lambda) + \delta Z(\lambda))$ is the polynomial linearized by a slight perturbation of the companion pencil of $q(\lambda) =$

$\lambda^n - 1$. Thanks to Edelman and Murakami (Lemma 7.4), given that $Z(\lambda)$ has norm 1, we can write it as $q(\lambda) + \delta q(\lambda)$, with the vector of coefficients of $\delta q(\lambda)$ being bounded in norm by a modest multiple of ϵ_Z. Concerning the second addend, we can write it, ignoring second order terms, as

$$y^T \operatorname{adj}(Z(\lambda))x + \delta y^T \operatorname{adj}(Z(\lambda))x + y^T \operatorname{adj}(Z(\lambda))\delta x + y^T \left[\operatorname{adj}(Z(\lambda) + \delta Z(\lambda)) - \operatorname{adj}(Z(\lambda))\right] x.$$

The first term cancels out with the corresponding term in $p(\lambda)$; the second and the third ones can be bounded in norm by $\epsilon_y \| x \|$ and $\| y \| \epsilon_x$, respectively. The last one can be bounded by $\| y \| \| x \| \kappa_{\text{adj}}(Z(\lambda))$, thanks to Lemma 7.7. Combining all these pieces together yields

$$\| \delta p \| \lesssim \epsilon_Z + \| x \| \epsilon_y + \| y \| \epsilon_x + \| x \| \| y \| \epsilon_Z.$$

which concludes the proof. □

If we apply this theorem to the companion QR and the companion QZ algorithm we get the following.

THEOREM 7.9 (Backward error on coefficients from companion matrix). *The roots of a polynomial $p(\lambda)$ computed with the algorithm of Aurentz et al. [4] operating on a companion matrix are the exact roots of a polynomial $\tilde{p}(\lambda)$, where the coefficients a and $\tilde{a}$ of $p(\lambda)$ and $\tilde{p}(\lambda)$ respectively satisfy*

$$\| a - \tilde{a} \| \lesssim \left(\frac{\| a \|}{|a_n|} \right)^2 \epsilon_m.$$

Proof. According to the analysis of the previous section, we have that $\epsilon_Z \approx \varepsilon_m$, $\epsilon_x \approx \varepsilon_m$ and $\epsilon_y \approx (\| a \| / |a_n|)^2 \varepsilon_m$. Substituting these values in Theorem 7.8 yields the desired result. □

THEOREM 7.10 (Backward error on coefficients from companion pencil). *The roots of a polynomial $p(\lambda)$ computed with the algorithm described in this paper, operating on the companion pencil, are the exact roots of a polynomial $\tilde{p}(\lambda)$, where the coefficients a and $\tilde{a}$ of $p(\lambda)$ and $\tilde{p}(\lambda)$ respectively satisfy*

$$\| a - \tilde{a} \| \lesssim \| a \|^2 \epsilon_m.$$

Proof. The proof proceeds identically as the one of Theorem 7.9, with the difference that $\epsilon_y \approx \| a \|^2 \varepsilon_m$ □

The theorems in this section illustrate that the structured error on the companion pencil is not amplified with another norm of the polynomial coefficients when mapping the error back to the coefficients. They tell us that we can expect a quadratic growth in the backward error in $\| a \|$ both on the polynomial coefficients and on the companion pencil. The numerical experiments validate these theorems.

8. Numerical experiments. In this section we test the speed and the backward accuracy of the companion pencil algorithm. We compare it against the companion QR algorithm from Aurentz et al. [4]. The computations have been executed on an Intel Xeon CPU E5-2697v3 running at 2.60 GHz with 128 GB of memory.

This implementation of the QZ algorithm (single as well as double shift code) is an extension of the one of the QR algorithm; see [4] for more details. Both the QR

and the QZ algorithm are available as part of the **eiscor** package, which can be found at https://github.com/eiscor/eiscor. To compile the Fortran codes GFortran 5.4.0 was used. For the numerical experiments only the single shifted QZ algorithm was used.

Section 8.1 discusses the storage in our implementation. Section 8.2 compares the complexity of the QZ and the QR algorithm; theoretically, as the cost is dominated by turnovers, we expect that the QZ will take 5/3 of the time of the QR. In Section 8.3 48 special polynomials [4] are tested illustrating the positive influence of the scaling on the backward error. Section 8.4 illustrates that the iteration count of some particular polynomials depends significantly on the chosen shape in the QZ algorithm. The backward error of the algorithm is examined in Section 8.5.

8.1. Some details on the implementation. As core transformations we opted to use rotations of the form $\left[\begin{smallmatrix} c & -s \\ s & \overline{c} \end{smallmatrix}\right]$ with $|c|^2+s^2 = 1$. The s is kept real and c complex. Thus three real entries are needed to store a rotation. The advantage of keeping s real is that this is preserved during the turnover and the implementation of the turnover becomes simpler and faster.

Unfortunately a fusion of two rotators with real s does not necessarily result in a rotation with real s again. The phase factors will therefore be stored in an additional diagonal matrix,

$$\begin{bmatrix} c_1 & -s_1 \\ s_1 & \overline{c}_1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 \\ s_2 & \overline{c}_2 \end{bmatrix} = \begin{bmatrix} c_1c_2 - s_1s_2 & -c_1s_2 - s_1\overline{c}_2 \\ s_1c_2 + \overline{c}_1s_2 & -s_1s_2 + \overline{c}_1\overline{c}_2 \end{bmatrix} = \begin{bmatrix} c_3 & -s_3 \\ s_3 & \overline{c}_3 \end{bmatrix} \begin{bmatrix} d_1 & \\ & d_2 \end{bmatrix}.$$

The full factorization of our pencil is $(QDC^*(B + e_1y_1^T), C_W^*(B_W + e_1^T y_W^T))$, with D the diagonal containing the phase factors. There are no fusions executed within the matrix W so no diagonal D_W is required.

In the real double shift code all computations are restricted to the real field. As a result, the c is also real, reducing the storage of a single rotation to only two real entries. The diagonal D is omitted in this case. Overall the double shift code is expected to be about $\frac{3}{2}$ times faster than the complex single shift code for real polynomials.

8.2. Complexity. The algorithm requires $O(n^2)$ flops with n the degree of the polynomial. This is reflected in Figure 8.1. For the experiment we used random normally distributed coefficients scaled to have norm one. The twisted companion QZ algorithm is about a factor of 1.5–1.8 slower than the QR variant AMVW. This is expected since the QZ variant needs $\frac{5}{3}$ times as many turnovers as the QR variant.

Figure 8.1 also includes the backward error (no scaling was applied to the pencil). We compute the coefficients of the polynomial defined by the computed roots. We compare these coefficients to the coefficients of the original polynomial to obtain the backward error. These computations are ldone in multiple precision using MPFUN [33].

8.3. Special real polynomials. We examined the accuracy for the 48 special real polynomials used by Aurentz et al. [4]. These are special test polynomials from different sources [1, 7, 12, 13, 17, 18, 25] and from Noferini in personal communication. For details on the polynomials we refer the reader to [4].

We compare the backward error on the coefficients again by computing a polynomial from the roots in multiple precision. The results are shown in Table 8.1. The companion QR without balancing and the companion QZ algorithm without balancing perform comparable. The balancing on the QZ algorithm is based on a scaling

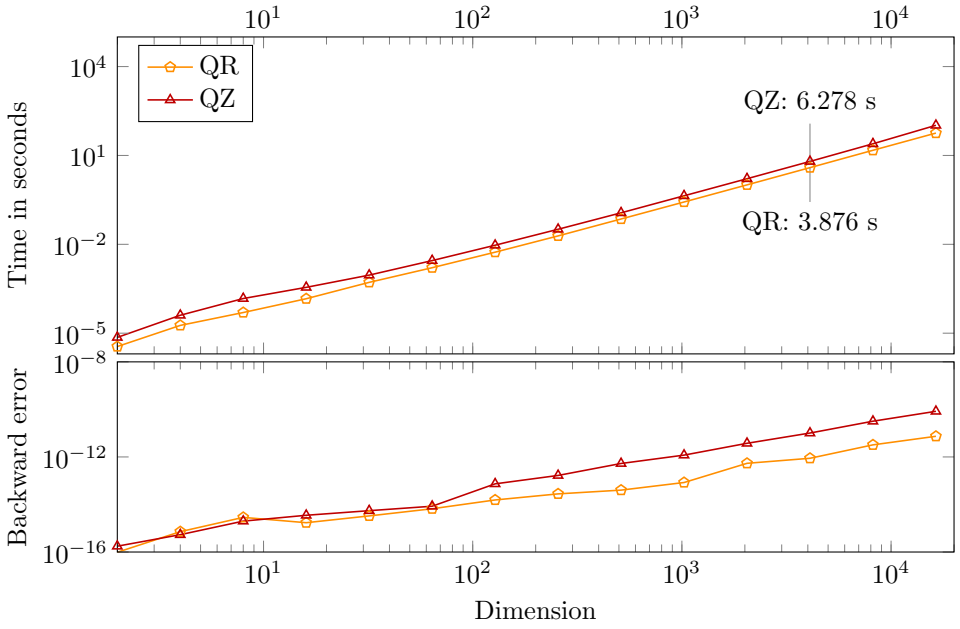


FIG. 8.1. *Runtime and absolute backward error on the coefficients for polynomials with random coefficients.*

of the variable and distribution of the coefficients, such that both vectors v and w in (2.2) are roughly of the same norm, followed by balancing such that these vectors v and w get norms close to 1. If we allow balancing for the companion QR and QZ algorithm we see a slight improvement in some cases and no improvement in others. Overall the QZ performs best.

8.4. Different shapes. Permitting different shapes for Q can reduce the number of iterations as shown by Vandebril [37]. In Table 8.2 we investigate the effect of the shape on the number of iterations for computing the roots of the 48 special polynomials from [4]. Table 8.2 only depicts the relevant rows, i.e., when there is a significant difference in the number of iterations between the various shapes.

We observe that the inverse Hessenberg shape performs well for equally spaced roots, e.g., for polynomials 1,2, and 3 with roots at $1,\ldots,10$, $1,\ldots,15$, and $1,\ldots,20$ respectively.

For some of the other special polynomials we also see a large difference in the number of iterations. The effect is especially visible for polynomial 8 and 22 with roots at $2^{-10},\ldots,2^{9}$ and $10^{-20},\ldots,10^{-1}$. For polynomial 12, which we obtained from MPSolve [12] but originates from C. Traverso, the Hessenberg shape does best.

8.5. Backward error. We show that the backward error on the matrix pencil and on the coefficients of the polynomials is bounded by $\mathcal{O}(\|a\|^2)$, where a is the vector of coefficients of the polynomial.

Figures 8.2 and 8.3 depict the absolute backward error on the coefficients and confirm the quadratic bound on both the matrix pencil and the polynomial coefficients.

After scaling the polynomial the backward error on the original (unscaled) coefficients grows linearly with the scaling factor and thus with $\|a\|_2$, see Figure 8.4.

No	QZ	QR	balanced QZ	balanced QR
1	7.77 e−16	3.25 e−16	5.57 e−16	5.48 e−15
2	8.22 e−16	4.10 e−16	1.01 e−15	8.05 e−14
3	1.08 e−15	7.65 e−16	9.64 e−16	4.12 e−12
4	2.14 e−15	1.67 e−15	1.15 e−15	1.85 e−15
5	1.19 e−15	4.25 e−16	1.65 e−15	1.84 e−16
6	1.49 e−15	1.13 e−15	2.18 e−15	4.23 e−16
7	3.08 e−15	9.43 e−16	3.09 e−15	2.04 e−16
8	1.24 e−15	1.01 e−15	2.39 e−15	7.00 e−16
9	3.37 e−15	1.25 e−15	2.25 e−15	2.82 e−11
10	3.18 e−15	1.83 e−15	3.88 e−15	3.00 e−15
11	3.11 e−15	1.95 e−15	3.11 e−15	1.95 e−15
12	1.41 e−16	8.69 e−17	2.08 e−16	2.98 e−14
13	1.03 e−15	1.01 e−15	1.50 e−15	2.38 e−15
14	2.27 e−15	3.35 e−15	2.62 e−15	2.17 e−15
15	2.71 e−15	5.49 e−16	2.71 e−15	9.04 e−11
16	3.70 e−15	2.52 e−15	7.43 e−15	2.95 e−06
17	1.57 e−16	7.07 e−17	1.57 e−16	1.57 e−16
18	1.57 e−16	1.96 e−17	1.57 e−16	1.57 e−16
19	1.41 e−16	1.41 e−16	1.41 e−16	2.07 e−13
20	9.95 e−17	9.95 e−17	9.95 e−17	1.80 e−11
21	3.95 e−16	5.54 e−16	3.95 e−16	6.90 e−17
22	1.65 e−15	6.63 e−16	1.56 e−15	2.76 e−17
23	2.57 e−16	1.29 e−16	2.57 e−16	1.29 e−16
24	2.63 e−16	1.32 e−16	2.63 e−16	1.32 e−16
25	3.93 e−26	1.57 e−26	3.93 e−26	1.57 e−26
26	2.34 e−14	6.54 e−15	2.54 e−14	1.29 e−14
27	3.82 e−15	2.17 e−15	4.07 e−15	3.77 e−15
28	5.89 e−16	4.07 e−16	1.21 e−15	1.14 e−14
29	7.74 e−15	3.56 e−15	6.29 e−15	2.11 e−15
30	1.76 e−14	5.79 e−15	2.21 e−14	6.62 e−15
31	2.50 e−14	1.23 e−14	3.13 e−14	1.60 e−14
32	3.38 e−12	2.56 e−13	3.38 e−12	2.31 e−13
33	1.12 e−11	6.33 e−13	9.53 e−12	1.24 e−12
34	1.54 e−15	1.63 e−15	4.81 e−15	1.63 e−15
35	7.31 e−15	3.90 e−15	8.35 e−15	3.90 e−15
36	1.39 e−14	4.79 e−15	1.06 e−14	4.79 e−15
37	2.91 e−13	4.14 e−14	2.38 e−13	4.14 e−14
38	6.83 e−13	5.14 e−14	6.53 e−13	5.14 e−14
39	3.39 e−15	3.22 e−15	3.39 e−15	3.22 e−15
40	4.48 e−14	5.57 e−15	4.48 e−14	5.57 e−15
41	4.58 e−14	2.22 e−14	4.58 e−14	2.22 e−14
42	2.69 e−12	2.92 e−13	2.69 e−12	2.92 e−13
43	1.06 e−11	7.99 e−13	1.06 e−11	1.40 e−12
44	3.39 e−15	3.22 e−15	3.39 e−15	3.22 e−15
45	4.48 e−14	5.57 e−15	4.48 e−14	5.57 e−15
46	4.58 e−14	2.22 e−14	4.58 e−14	2.22 e−14
47	2.69 e−12	2.92 e−13	2.69 e−12	2.92 e−13
48	1.06 e−11	1.46 e−12	1.06 e−11	2.78 e−13

TABLE 8.1
Relative backward error on the polynomial coefficients.

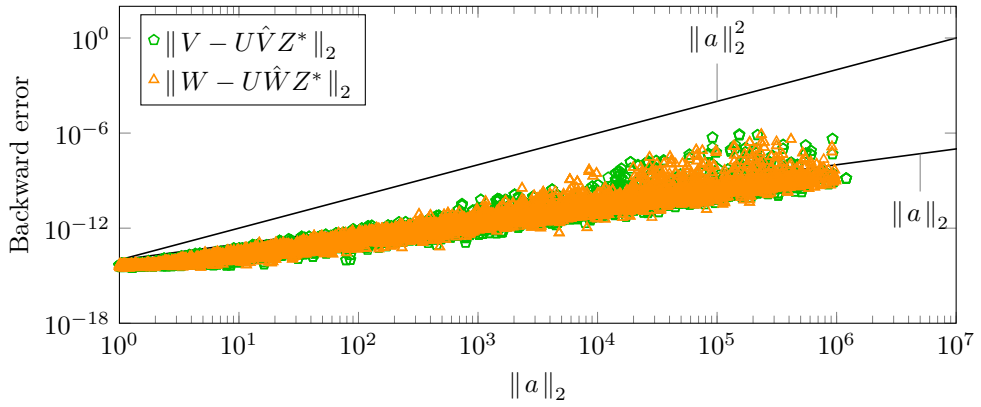


FIG. 8.2. *Absolute backward error on the matrix pencil depending on* $\|a\|_2$.

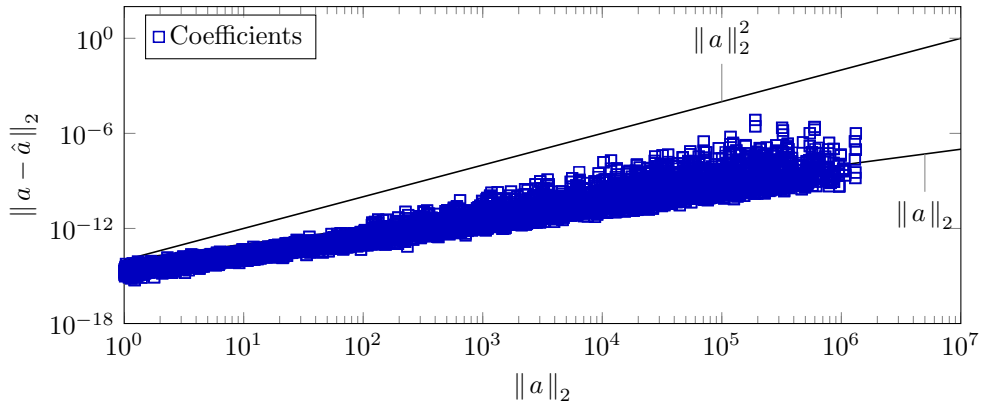


FIG. 8.3. *Absolute backward error of the coefficients of the polynomials over* $\|a\|_2$.

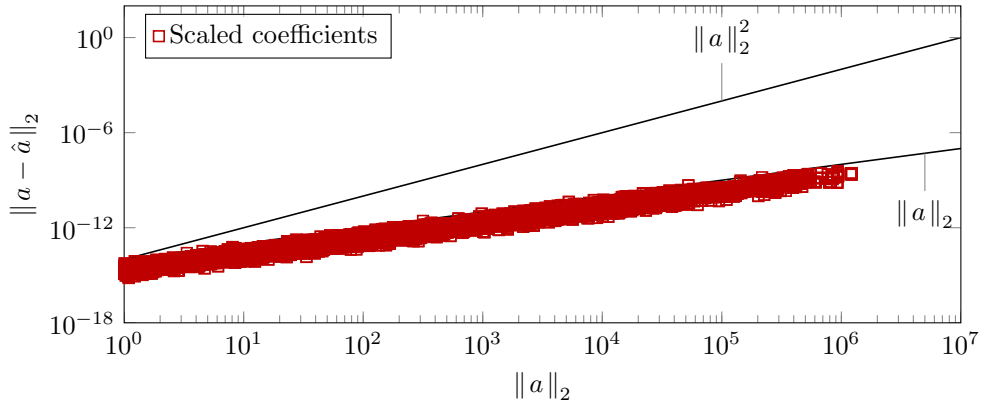


FIG. 8.4. *Absolute backward error of the original coefficients of the polynomials over* $\|a\|_2$ *using scaling.*

No	Hessenberg	inverse Hessenberg	CMV	Random
1	41	29	35	35
2	59	45	50	52
3	61	58	64	65
4	64	52	62	65
5	43	33	43	42
6	60	48	67	60
7	89	60	82	62
8	99	39	49	47
9	88	61	67	68
10	68	56	68	70
11	58	58	59	58
12	43	63	80	75
13	114	98	105	105
14	226	196	209	210
15	36	27	33	37
16	113	94	106	105
21	56	38	43	38
22	140	64	64	73
26	182	173	176	173
27	70	63	67	67
28	58	73	69	63

TABLE 8.2
Number of iterations for different patterns.

9. Conclusions and future work. We have extended the companion QR algorithm for computing roots of polynomials to a companion QZ algorithm. The companion QZ algorithm is roughly 5/3rd slower than the QR version, but provides better backward error norm when scaling is used. Furthermore we have generalized both the QR and QZ algorithms to the twisted setting, allowing for more advanced convergence behavior.

As future work we aspire towards (twisted) multishift QR and QZ algorithms (see, e.g., [38, 39]), which are proven to be more efficient for dense matrices as they allow for a more efficient usage of cache by blocking some of the operations [27]. This could then be combined with aggressive early deflation [28] to accelerate the computation of the roots even further.

REFERENCES

[1] T. Aktosun, D. Gintides, and V. G. Papanicolaou, *The uniqueness in the inverse problem for transmission eigenvalues for the spherically symmetric variable-speed wave equation*, Inverse Problems, 27 (2011), p. 115004.

[2] J. Aurentz, R. Vandebril, and D. S. Watkins, *Fast computation of the zeros of a polynomial via factorization of the companion matrix*, SIAM Journal on Scientific Computing, 35 (2013), pp. A255–A269.

[3] ———, *Fast computation of eigenvalues of companion, comrade, and related matrices*, BIT, 54 (2014), pp. 7–30.

[4] J. L. Aurentz, T. Mach, R. Vandebril, and D. S. Watkins, *Fast and backward stable computation of roots of polynomials*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 942–973.

[5] ———, *Fast and stable unitary QR algorithm*, Electronic Transactions on Numerical Analysis, 44 (2015), pp. 327–341. Submitted for publication.
[6] J. L. Aurentz, T. Mach, R. Vandebril, and D. S. Watkins, *A note on companion pencils*, Contemporary Mathematics, 658 (2016), pp. 91–101.
[7] R. Bevilacqua, G. M. Del Corso, and L. Gemignani, *A CMV–based eigensolver for companion matrices*, SIAM Journal on Matrix Analysis and Applications, 36 (2015), pp. 1046–1068.
[8] D. A. Bini, *Numerical computation of polynomial zeros by means of Aberth's algorithm*, Numerical Algorithms, 13 (1996), pp. 179–200.
[9] D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg, *A fast implicit QR eigenvalue algorithm for companion matrices*, Linear Algebra and its Applications, 432 (2010), pp. 2006–2031.
[10] D. A. Bini, F. Daddi, and L. Gemignani, *On the shifted QR iteration applied to companion matrices*, Electronic Transactions on Numerical Analysis, 18 (2004), pp. 137–152.
[11] D. A. Bini, Y. Eidelman, L. Gemignani, and I. Gohberg, *Fast QR eigenvalue algorithms for Hessenberg matrices which are rank-one perturbations of unitary matrices*, SIAM Journal on Matrix Analysis and Applications, 29 (2007), pp. 566–585.
[12] D. A. Bini and G. Fiorentino, *Design, analysis, and implementation of a multiprecision polynomial rootfinder*, Numerical Algorithms, 23 (2000), pp. 127–173.
[13] D. A. Bini, G. Fiorentino, L. Gemignani, and B. Meini, *Effective fast algorithms for polynomial spectral factorization*, Numerical Algorithms, 34 (2003), pp. 217–227.
[14] P. Boito, Y. Eidelman, and L. Gemignani, *Implicit QR for companion-like pencils*, Mathematics of Computation, 85 (2016), pp. 1753–1774.
[15] P. Boito, Y. Eidelman, and L. Gemignani, *A real qz algorithm for structured companion pencils*. https://arxiv.org/abs/1608.05395, 2016.
[16] P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg, *Implicit QR with compression*, Indagationes Mathematicae, 23 (2012), pp. 733–761.
[17] A. Böttcher and M. Halwass, *Wiener–Hopf and spectral factorization of real polynomials by Newton's method*, Linear Algebra and its Applications, 438 (2013), pp. 4760–4805.
[18] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu, *A fast QR algorithm for companion matrices*, Operator Theory: Advances and Applications, 179 (2007), pp. 111–143.
[19] F. de Terán, F. M. Dopico, J. Pérez, and a. a. check, *Backward stability of polynomial root-finding using fiedler companion matrices*, IMA Journal of Numerical Analysis, (2014).
[20] S. Delvaux, K. Frederix, and M. Van Barel, *An algorithm for computing the eigenvalues of block companion matrices*, Numerical Algorithms, 62 (2012), pp. 261–287.
[21] B. Eastman, I.-J. Kim, B. Shader, and K. Vander Meulen, *Companion matrix patterns*, Linear Algebra and its Applications, 463 (2014), pp. 255–272.
[22] A. Edelman and H. Murakami, *Polynomial roots from companion matrix eigenvalues*, Mathematics of Computation, 64 (1995), pp. 763–776.
[23] Y. Eidelman, I. C. Gohberg, and I. Haimovici, *Separable Type Representations of Matrices and Fast Algorithms – Volume 2: Eigenvalue Method*, no. 235 in Operator Theory: Advances and Applications, Springer Basel, 2013.
[24] M. Fiedler, *A note on companion matrices*, Linear Algebra and its Applications, 372 (2003), pp. 325–331.
[25] M. A. Jenkins and J. F. Traub, *Principles for testing polynomial zerofinding programs*, ACM Transactions on Mathematical Software, 1 (1975), pp. 26–34.
[26] G. F. Jónsson and S. Vavasis, *Solving polynomials with small leading coefficients*, SIAM Journal on Matrix Analysis and Applications, 26 (2004), pp. 400–414.
[27] D. Kressner, *Numerical methods for general and structured eigenvalue problems*, vol. 46 of LNCSE, Springer, 2005.
[28] T. Mach and R. Vandebril, *On deflations in extended QR algorithms*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 559–579.
[29] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann, *Vector spaces of linearizations for matrix polynomials*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 971–1004.
[30] J. M. McNamee, *A bibliography on roots of polynomials*, Journal of Computational and Applied Mathematics, 47 (1993), pp. 391–394.
[31] C. B. Moler, *Cleve's corner: Roots - of polynomials, that is*, The Mathworks Newsletter, 5 (1991), pp. 8–9.
[32] C. B. Moler and G. W. Stewart, *An algorithm for generalized matrix eigenvalue problems*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 241–256.
[33] *MPFUN - multiprecision software*. http://www.netlib.org/mpfun/, 2005.

[34] G. W. Stewart, *On the adjugate matrix*, Linear Algebra and its Applications, 283 (1998), pp. 151–164.
[35] M. Van Barel, R. Vandebril, P. Van Dooren, and K. Frederix, *Implicit double shift QR-algorithm for companion matrices*, Numerische Mathematik, 116 (2010), pp. 177–212.
[36] P. Van Dooren and P. Dewilde, *The eigenstructure of an arbitrary polynomial matrix: Computational aspects*, Linear Algebra and its Applications, 50 (1983), pp. 545–579.
[37] R. Vandebril, *Chasing bulges or rotations? A metamorphosis of the QR-algorithm*, SIAM Journal on Matrix Analysis and Applications, 32 (2011), pp. 217–247.
[38] R. Vandebril and D. S. Watkins, *An extension of the QZ algorithm beyond the Hessenberg-upper triangular pencil*, Electronic Transactions on Numerical Analysis, 40 (2012), pp. 17–35.
[39] ———, *A generalization of the multishift QR algorithm*, SIAM Journal on Matrix Analysis and Applications, 33 (2012), pp. 759–779.
[40] D. S. Watkins, *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*, SIAM, Philadelphia, USA, 2007.
[41] ———, *Fundamentals of Matrix Computations*, Pure and Applied Mathematics, John Wiley & Sons, Inc., New York, third ed., 2010.
[42] P. Zhlobich, *Differential qd algorithm with shifts for rank-structured matrices*, SIAM Journal on Matrix Analysis and Applications, 33 (2012).