Calcolo manuscript No.
(will be inserted by the editor)

A Real QZ Algorithm for Structured Companion Pencils

P. Boito · Y. Eidelman · L. Gemignani

Received: date / Accepted: date

Abstract We design a fast implicit real QZ algorithm for eigenvalue computation of structured companion pencils arising from linearizations of polynomial rootfinding problems. The modified QZ algorithm computes the generalized eigenvalues of an $N \times N$ structured matrix pencil using $O(N)$ flops per iteration and $O(N)$ memory storage. Numerical experiments and comparisons confirm the effectiveness and the stability of the proposed method.

Keywords Rank–structured matrix · Quasiseparable matrix · Real QZ algorithm · Lagrange approximation · eigenvalue computation · complexity

Mathematics Subject Classification (2000) 65F15 · 65H17

1 Introduction

Linearization techniques based on polynomial interpolation are becoming nowadays a standard way to solve numerically nonlinear zerofinding problems for polynomials or more generally for analytic functions [2]. Since in many applications the interest is in the approximation of real zeros, methods using Chebyshev–like expansions are usually employed. Alternatively, Lagrange interpolation at the roots

This work was partially supported by GNCS–INDAM and University of Pisa.

P. Boito
XLIM–DMI UMR CNRS 7252 Faculté des Sciences et Techniques, 123 avenue A. Thomas, 87060 Limoges, France
and CNRS, Université de Lyon, Laboratoire LIP (CNRS, ENS Lyon, Inria, UCBL), 46 allée d'Italie, 69364 Lyon Cedex 07, France
E-mail: paola.boito@unilim.fr

Y. Eidelman
School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel-Aviv University, Ramat-Aviv, 69978, Israel
E-mail: eideyu@post.tau.ac.il

L. Gemignani
Dipartimento di Informatica, Università di Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa, Italy
E-mail: l.gemignani@di.unipi.it

of unity can be considered. For a real function a straightforward modification of the classical approach [6,18] yields a structured companion pencil $\mathcal{A}(\lambda) = F - \lambda G$ where F and G are real $N \times N$ low rank corrections of unitary matrices. The computation of the generalized eigenvalues of this pencil can be performed by means of the QZ algorithm [23] suitably adjusted to work in real arithmetic.

In this paper we propose a fast adaptation of the real QZ algorithm for computing the generalized eigenvalues of certain $N \times N$ structured pencils using only $O(N)$ flops per iteration and $O(N)$ memory storage. Since in most cases it is reasonable to assume that the total number of iterations is a small multiple of N (see e.g., [24]), we have a heuristic complexity estimate of $O(N^2)$ flops to compute all the eigenvalues.

The pencils $\mathcal{A}(\lambda) = F - \lambda G$, $F, G \in \mathbb{R}^{N\times N}$, we consider here satisfy two basic properties:

1. F is upper Hessenberg and G is upper triangular;
2. F and G are rank–one corrections of unitary matrices.

We refer to such a pencil $\mathcal{A}(\lambda)$ as a companion–like pencil, since the class includes companion pencils as a special case. Sometimes $\mathcal{A}(\lambda)$ is also denoted by $(F, G) \in \mathbb{R}^{N\times N} \times \mathbb{R}^{N\times N}$.

Let (F_k, G_k), $k \geq 0$, $F_0 = F, G_0 = G$, be the sequence of matrix pairs (pencils) generated by the real QZ algorithm starting from the companion–like pencil $\mathcal{A}(\lambda)$. Single or double shifting is applied in the generic iteration $F_k \to F_{k+1}$ $G_k \to G_{k+1}$ in order to carry out all the computations in real arithmetic. Whatever strategy is used, it is found that both $\mathcal{A}_k(\lambda)$ and $\mathcal{A}_{k+1}(\lambda)$ are still companion–like pencils. As a consequence of this invariance we obtain that all the matrices involved in the QZ iteration inherit a rank structure in their upper triangular parts. This makes it possible to represent F_k, G_k and F_{k+1}, G_{k+1} as data–sparse matrices specified by a number of parameters (called generators) which is linear w.r.t. the size of the matrices. This general principle has been applied, for instance, in [1] and [5].

In this paper we introduce a convenient set of generators and design a structured variant of the real QZ iteration which takes in input the generators of F_k and G_k together with the shift parameters and returns as output the generators of F_{k+1} and G_{k+1}. It is shown that the arithmetic cost for each iteration is $O(N)$ using linear memory storage. Numerical experiments confirm the effectiveness and the robustness of the resulting eigensolver.

The paper is organized as follows. In Section 2 we set up the scene by introducing the matrix problem and its basic properties. In Section 3 we define an appropriate set of generators for the matrices involved. In Section 4 we design the fast adaptation of the QZ algorithm using these generators and exploiting the resulting data–sparse representations. We focus here on double shifting, since the single shifted iteration has been already devised in [5]. A proof of the correctness of the algorithm is given in Appendix. Finally, in Section 5 we show the results of numerical experiments, whereas conclusion and future work are presented in Section 6.

2 The Problem Statement

Companion pencils and companion–like pencils expressed in the Lagrange basis at the roots of unity are specific instances of the following general class.

Definition 1 The matrix pair (A, B), $A, B \in \mathbb{R}^{N\times N}$, belongs to the class $\mathcal{P}_N \subset \mathbb{R}^{N\times N} \times \mathbb{R}^{N\times N}$ of companion–like pencils iff:

1. $A \in \mathbb{R}^{N\times N}$ is upper Hessenberg;
2. $B \in \mathbb{R}^{N\times N}$ is upper triangular;
3. There exist two vectors $\boldsymbol{z} \in \mathbb{R}^N$ and $\boldsymbol{w} \in \mathbb{R}^N$ and an orthogonal matrix $V \in \mathbb{R}^{N\times N}$ such that
$$A = V - \boldsymbol{z}\boldsymbol{w}^*; \tag{2.1}$$
4. There exist two vectors $\boldsymbol{p} \in \mathbb{R}^N$ and $\boldsymbol{q} \in \mathbb{R}^N$ and an orthogonal matrix $U \in \mathbb{R}^{N\times N}$ such that
$$B = U - \boldsymbol{p}\boldsymbol{q}^*. \tag{2.2}$$

In order to characterize the individual properties of the matrices A and B we give some additional definitions.

Definition 2 We denote by $\mathcal{T}_N$ the class of upper triangular matrices $B \in \mathbb{R}^{N\times N}$ that are rank-one perturbations of orthogonal matrices, i.e., such that (2.2) holds for a suitable orthogonal matrix U and vectors $\boldsymbol{p}, \boldsymbol{q}$.

Since B is upper triangular the strictly lower triangular part of the orthogonal matrix U in (2.2) coincides with the corresponding part of the rank one matrix $\boldsymbol{p}\boldsymbol{q}^*$, i.e.,
$$U(i, j) = p(i)q^*(j), \quad 1 \le j < i \le N, \tag{2.3}$$
where $\{p(i)\}_{i=1,\ldots,N}$ and $\{q(j)\}_{j=1,\ldots,N}$ are the entries of $\boldsymbol{p}$ and $\boldsymbol{q}$, respectively.

Definition 3 We denote by $\mathcal{U}_N$ the class of orthogonal matrices $U \in \mathbb{R}^{N\times N}$ that satisfy the condition (2.3), i.e., for which there exist vectors $\boldsymbol{p}, \boldsymbol{q}$ such that the matrix $B = U - \boldsymbol{p}\boldsymbol{q}^*$ is an upper triangular matrix.

Observe that we have
$$U \in \mathcal{U}_N \Rightarrow \operatorname{rank} U(k+1\colon N, 1\colon k) \le 1, \quad k = 1, \ldots, N-1.$$
From the nullity theorem [13], see also [11, p.142], it follows that the same property also holds in the strictly upper triangular part, namely,
$$U \in \mathcal{U}_N \Rightarrow \operatorname{rank} U(1\colon k, k+1\colon N) \le 1, \quad k = 1, \ldots, N-1. \tag{2.4}$$

Definition 4 We denote by $\mathcal{H}_N$ the class of upper Hessenberg matrices $A \in \mathbb{R}^{N\times N}$ that are rank one perturbations of orthogonal matrices, i.e., such that (2.1) holds for a suitable orthogonal matrix V and vectors $\boldsymbol{z}, \boldsymbol{w}$.

Definition 5 We denote by $\mathcal{V}_N$ the class of orthogonal matrices $V \in \mathbb{R}^{N\times N}$ for which there exist vectors $\boldsymbol{z}, \boldsymbol{w}$ such that the matrix $A = V - \boldsymbol{z}\boldsymbol{w}^*$ is an upper Hessenberg matrix.

We find that
$$V \in \mathcal{V}_N \Rightarrow \operatorname{rank} V(k+2\colon N, 1\colon k) \le 1, \quad k = 1, \ldots, N-2.$$

Again from the nullity theorem it follows that a similar property also holds in the upper triangular part, namely,

$$V \in \mathcal{V}_N \Rightarrow \operatorname{rank} V(1\colon k, k\colon N) \le 2, \quad k = 1, \dots, N. \tag{2.5}$$

In this paper we consider the problem of efficiently computing the (generalized) eigenvalues of a companion–like matrix pencil $(A, B) \in \mathcal{P}_N$ by exploiting the rank and banded structures of the matrix classes mentioned above. The QZ algorithm is the customary method for solving generalized eigenvalue problems numerically by means of unitary transformations (see e.g. [14] and [23]). For the pair (A, B) in Hessenberg/triangular form the implicit QZ step consists in the computation of unitary matrices Q and Z such that

$$A_1 = Q^* A Z \text{ is upper Hessenberg, } B_1 = Q^* B Z \text{ is upper triangular} \tag{2.6}$$

and some initial conditions hold. For the QZ iteration applied to a real matrix pair with double shifting the initial condition is

$$(Q^* p(AB^{-1}))(:, 2 : N) = 0, \tag{2.7}$$

where $p(z) = \alpha + \beta z + \gamma z^2$ is the shift polynomial. In this case one obtains the orthogonal Hessenberg matrices Q and Z in the form

$$Q = \tilde{Q}_1 \tilde{Q}_2 \cdots \tilde{Q}_{N-2} \tilde{Q}_{N-1}, \quad Z = \tilde{Z}_1 \tilde{Z}_2 \cdots \tilde{Z}_{N-2} \tilde{Z}_{N-1}, \tag{2.8}$$

where

$$\begin{aligned} \tilde{Q}_i &= I_{i-1} \oplus Q_i \oplus I_{N-i-2},\ i = 1, \dots, N-2, \quad \tilde{Q}_{N-1} = I_{N-2} \oplus Q_{N-1}, \\ \tilde{Z}_i &= I_{i-1} \oplus Z_i \oplus I_{N-i-2},\ i = 1, \dots, N-2, \quad \tilde{Z}_{N-1} = I_{N-2} \oplus Z_{N-1} \end{aligned} \tag{2.9}$$

and $Q_i, Z_i,\ i = 1, \dots, N-2$ are 3×3 orthogonal matrices, Q_{N-1}, Z_{N-1} are real Givens rotations.

Since the Hessenberg/triangular form is preserved under the QZ iteration an easy computation then yields

$$(A, B) \in \mathcal{P}_N,\ (A, B) \overset{\text{QZ step}}{\to} (A_1, B_1) \Rightarrow (A_1, B_1) \in \mathcal{P}_N. \tag{2.10}$$

Indeed if Q and Z are unitary then from (2.1) and (2.2) it follows that the matrices $A_1 = Q^* A Z$ and $B_1 = Q^* B Z$ satisfy the relations

$$A_1 = V_1 - \boldsymbol{z}_1 \boldsymbol{w}_1^*, \quad B_1 = U_1 - \boldsymbol{p}_1 \boldsymbol{q}_1^*$$

with the unitary matrices $V_1 = Q^* V Z,\ U_1 = Q^* U Z$ and the vectors $\boldsymbol{z}_1 = Q^* z,\ \boldsymbol{w}_1 = Z^* w,\ \boldsymbol{p}_1 = Q^* p,\ \boldsymbol{q}_1 = Z^* q$. Moreover one can choose the unitary matrices Q and Z such that the matrix A_1 is upper Hessenberg and the matrix B_1 is upper triangular. Thus, one can in principle think of designing a structured QZ iteration that, given in input a condensed representation of the matrix pencil $(A, B) \in \mathcal{P}_N$, returns as output a condensed representation of $(A_1, B_1) \in \mathcal{P}_N$ generated by one step of the classical QZ algorithm applied to (A, B). In the next sections we first introduce an eligible representation of a rank-structured matrix pencil $(A, B) \in \mathcal{P}_N$ and then discuss the modification of this representation under the QZ process.

3 Quasiseparable Representations

In this section we exploit the properties of quasiseparable representations of rank–structured matrices [10], [11, Chapters 4,5]. First we recall some general results and definitions. Subsequently, we describe their adaptations for the representation of the matrices involved in the structured QZ iteration applied to an input matrix pencil $(A, B) \in \mathcal{P}_N$.

A matrix $M = \{M_{ij}\}_{i,j=1}^{N}$ is (r^L, r^U)-*quasiseparable*, with r^L, r^U positive integers, if, using MATLAB[1] notation,

$$\max_{1\le k\le N-1} \operatorname{rank}\left(M(k+1:N, 1:k)\right) \le r^L,$$
$$\max_{1\le k\le N-1} \operatorname{rank}\left(M(1:k, k+1:N)\right) \le r^U.$$

Roughly speaking, this means that every submatrix extracted from the lower triangular part of M has rank at most r^L, and every submatrix extracted from the upper triangular part of M has rank at most r^U. Under this hypothesis, M can be represented using $\mathcal{O}(((r^L)^2 + (r^U)^2)N)$ parameters. In this section we present such a representation.

The quasiseparable representation of a rank–structured matrix consists of a set of vectors and matrices used to generate its entries. For the sake of notational simplicity and clarity, generating matrices and vectors are denoted by a roman lower-case letter.

In this representation, the entries of M take the form

$$M_{ij} = \begin{cases} p(i)a_{ij}^{>}q(j), & 1 \le j < i \le N, \\ d(i), & 1 \le i = j \le N, \\ g(i)b_{ij}^{<}h(j), & 1 \le i < j \le N \end{cases} \tag{3.1}$$

where:

- $p(2), \ldots, p(N)$ are row vectors of length r^L, $q(1), \ldots, q(N-1)$ are column vectors of length r^L, and $a(2), \ldots, a(N-1)$ are matrices of size $r^L \times r^L$; these are called *lower quasiseparable generators* of order r^L;
- $d(1), \ldots, d(N)$ are numbers (the diagonal entries),
- $g(2), \ldots, g(N)$ are row vectors of length r^U, $h(1), \ldots, h(N-1)$ are column vectors of length r^U, and $b(2), \ldots, b(N-1)$ are matrices of size $r^U \times r^U$; these are called *upper quasiseparable generators* of order r^U;
- the matrices $a_{ij}^{>}$ and $b_{ij}^{<}$ are defined as

$$\begin{cases} a_{ij}^{>} = a(i-1)\cdots a(j+1) \text{ for } i > j+1; \\ a_{j+1,j}^{>} = 1 \end{cases}$$

and

$$\begin{cases} b_{ij}^{<} = b(i+1)\cdots b(j-1) \text{ for } j > i+1; \\ b_{i,i+1}^{<} = 1. \end{cases}$$

[1] MATLAB is a registered trademark of The Mathworks, Inc..

From (2.4) it follows that any matrix from the class $\mathcal{U}_N$ has upper quasiseparable generators with orders equal to one.

The quasiseparable representation can be generalized to the case where M is a block matrix, and to the case where the generators do not all have the same size, provided that their product is well defined. Each block M_{ij} of size $m_i \times n_j$ is represented as in (3.1), except that the sizes of the generators now depend on m_i and n_j, and possibly on the index of a and b. More precisely:

- $p(i), q(j), a(k)$ are matrices of sizes $m_i \times r^L_{i-1}$, $r^L_j \times n_j$, $r^L_k \times r^L_{k-1}$, respectively;
- $d(i)$ $(i = 1, \ldots, N)$ are $m_i \times n_i$ matrices,
- $g(i), h(j), b(k)$ are matrices of sizes $m_i \times r^U_i$, $r^U_{j-1} \times n_j$, $r^U_{k-1} \times r^U_k$, respectively.

The numbers r^L_k, r^U_k $(k = 1, \ldots, N-1)$ are called the *orders* of these generators.

It is worth noting that lower and upper quasiseparable generators of a matrix are not uniquely defined. A set of generators with minimal orders can be determined according to the ranks of maximal submatrices located in the lower and upper triangular parts of the matrix.

One advantage of the block representation for the purposes of the present paper consists in the fact that $N \times N$ upper Hessenberg matrices can be treated as $(N+1) \times (N+1)$ block upper triangular ones by choosing block sizes as

$$m_1 = \cdots = m_N = 1,\ m_{N+1} = 0, \quad n_1 = 0,\ n_2 = \cdots = n_{N+1} = 1. \tag{3.2}$$

Such a treatment allows also to consider quasiseparable representations which include the main diagonals of matrices. Assume that C is an $N \times N$ scalar matrix with the entries in the upper triangular part represented in the form

$$C(i,j) = g(i) b^{<}_{i-1,j} h(j), \quad 1 \le i \le j \le N \tag{3.3}$$

with matrices $g(i), h(i)$ $(i = 1, \ldots, N)$, $b(k)$ $(k = 1, \ldots, N-1)$ of sizes $1 \times r_i, r_i \times 1, r_k \times r_{k+1}$. The elements $g(i), h(i)$ $(i = 1, \ldots, N)$, $b(k)$ $(k = 1, \ldots, N-1)$ are called *upper triangular generators* of the matrix C with orders r_k $(k = 1, \ldots, N)$. From (2.5) it follows that any matrix from the class $\mathcal{V}_N$ has upper triangular generators with orders not greater than two. If we treat a matrix C as a block one with entries if sizes (3.2) we conclude that the elements $g(i)$ $(i = 1, \ldots, N)$, $h(j-1)$ $(j = 2, \ldots, N+1)$, $b(k-1)$ $(k = 2, \ldots, N)$ are upper quasiseparable generators of C.

Matrix operations involving zero-dimensional arrays (empty matrices) are defined according to the rules used in MATLAB and described in [7]. In particular, the product of a $m \times 0$ matrix by a $0 \times m$ matrix is a $m \times m$ matrix with all entries equal to 0. Empty matrices may be used in assignment statements as a convenient way to add and/or delete rows or columns of matrices.

3.1 **Representations of matrix pairs from the class $\mathcal{P}_N$**

Let (A, B) be a matrix pair from the class $\mathcal{P}_N$. The corresponding matrix A from the class $\mathcal{H}_N$ is completely defined by the following parameters:

1. the subdiagonal entries σ^A_k $(k = 1, \ldots, N-1)$ of the matrix A;
2. the upper triangular generators $g_V(i), h_V(i)$ $(i = 1, \ldots, N)$, $b_V(k)$ $(k = 1, \ldots, N-1)$ of the corresponding unitary matrix V from the class $\mathcal{V}_N$;

3. the vectors of perturbation $\boldsymbol{z} = \mathrm{col}(z(i))_{i=1}^{N}$, $\boldsymbol{w} = \mathrm{col}(w(i))_{i=1}^{N}$.

From (2.5) it follows that the matrix $V \in \mathcal{V}_N$ has upper triangular generators with orders not greater than two.

The corresponding matrix B from the class $\mathcal{T}_N$ is completely defined by the following parameters:

1. the diagonal entries $d_B(k)$ $(k = 1, \ldots, N)$ of the matrix B;
2. the upper quasiseparable generators $g_U(i)$ $(i = 1, \ldots, N-1)$, $h_U(j)$ $(j = 2, \ldots, N)$, $b_U(k)$ $(k = 2, \ldots, N-1)$ of the corresponding unitary matrix U from the class $\mathcal{U}_N$;
3. the vectors of perturbation $\boldsymbol{p} = \mathrm{col}(p(i))_{i=1}^{N}$, $\boldsymbol{q} = \mathrm{col}(q(i))_{i=1}^{N}$.

From (2.4) it follows that the matrix $U \in \mathcal{U}_N$ has upper quasiseparable generators with orders equal one.

All the given parameters define completely the matrix pair (A, B) from the class $\mathcal{P}_N$. Updating of these parameters while keeping the minimal orders of generators is a task of the fast QZ iteration described in the next section.

4 A fast implicit double shifted QZ iteration via generators

In this section we present our fast adaptation of the double–shifted QZ algorithm for a matrix pair $(A, B) \in \mathcal{P}_N$. The algorithm takes in input a quasiseparable representation of the matrices A and B together with the coefficients of the real quadratic shift polynomial and it returns as output a possibly not minimal quasiseparable representation of the matrices $(A_1, B_1) \in \mathcal{P}_N$ such that (2.10) holds. The algorithm computes the unitary matrices Q_i and Z_i defined in (2.9). It basically splits into the following four stages:

1. a **preparative phase** where Q_1 is found so as to satisfy the shifting condition;
2. the **chasing the bulge** step where the unitary matrices $Q_2, \ldots, Q_{N-2}$ and $Z_1, \ldots, Z_{N-3}$ are determined in such a way to perform the Hessenberg/triangular reduction procedure;
3. a **closing** phase where the last three transformations Q_{N-1}, Z_{N-2} and Z_{N-1} are carried out;
4. the final stage of **recovering the generators** of the updated pair.

For the sake of brevity the stage 2 and 3 are grouped together by using empty and zero quantities when needed. The correctness of the algorithm is proved in the Appendix. Some technical details concerning shifting strategies and shifting techniques are discussed in the section on numerical experiments. Compression of generators yielding minimal representations can be achieved by using the methods devised in [5]. The incorporation of these compression schemes does not alter the complexity of the main algorithm shown below.

ALGORITHM: Implicit QZ iteration for companion–like pencils with double shift

1. **INPUT**:
   (a) the subdiagonal entries σ_k^A $(k = 1, \ldots, N-1)$ of the matrix A;
   (b) the upper triangular generators $g_V(i), h_V(i)$ $(i = 1, \ldots, N)$, $b_V(k)$ $(k = 1, \ldots, N-1)$ with orders r_k^V $(k = 1, \ldots, N)$ of the matrix V;

(c) the diagonal entries $d_B(k)$ $(k = 1, \dots, N)$ of the matrix B;
(d) the upper quasiseparable generators $g_U(i)$ $(i = 1, \dots, N-1)$, $h_U(j)$ $(j = 2, \dots, N)$, $b_U(k)$ $(k = 2, \dots, N-1)$ with orders r_k^U $(k = 1, \dots, N-1)$ of the matrix U;
(e) the perturbation vectors $\boldsymbol{z} = \mathrm{col}(z(i))_{i=1}^N$, $\boldsymbol{w} = \mathrm{col}(w(i))_{i=1}^N$, $\boldsymbol{p} = \mathrm{col}(p(i))_{i=1}^N$, $\boldsymbol{q} = \mathrm{col}(q(i))_{i=1}^N$;
(f) the coefficients of the shift polynomial $p(z) = \alpha + \beta z + \gamma z^2 \in \mathbb{R}[z]$;

2. **OUTPUT**:
(a) the subdiagonal entries $\sigma_k^{A_1}$ $(k = 1, \dots, N-1)$ of the matrix A_1;
(b) upper triangular generators $g_V^{(1)}(i), h_V^{(1)}(i)$ $(i = 1, \dots, N)$, $b_V^{(1)}(k)$ $(k = 1, \dots, N-1)$ of the matrix V_1;
(c) the diagonal entries $d_B^{(1)}(k)$ $(k = 1, \dots, N)$ of the matrix B_1;
(d) upper quasiseparable generators $g_U^{(1)}(i)$ $(i = 1, \dots, N-1)$, $h_U^{(1)}(j)$ $(j = 2, \dots, N)$, $b_U^{(1)}(k)$ $(k = 2, \dots, N-1)$ of the matrix U_1;
(e) perturbation vectors $\boldsymbol{z}_1 = \mathrm{col}(z^{(1)}(i))_{i=1}^N$, $\boldsymbol{w}_1 = \mathrm{col}(w^{(1)}(i))_{i=1}^N$, $\boldsymbol{p}_1 = \mathrm{col}(p^{(1)}(i))_{i=1}^N$, $\boldsymbol{q}_1 = \mathrm{col}(q^{(1)}(i))_{i=1}^N$;

3. **COMPUTATION**:

– **Preparative Phase**

(a) Compute $\boldsymbol{s} = (p(AB^{-1})\boldsymbol{e}_1)(1\colon 3)$ and determine the 3×3 orthogonal matrix Q_1 from the condition

$$Q_1^* \boldsymbol{s} = \begin{pmatrix} \times & 0 & 0 \end{pmatrix}^*. \tag{4.1}$$

(b) Compute

$$\begin{pmatrix} \tilde{g}_V(3) \\ \beta_3^V \end{pmatrix} = Q_1^* \begin{pmatrix} g_V(1)h_V(1) & g_V(1)b_V(1)h_V(2) & g_V(1)b_V(1)b_V(2) \\ \sigma_1^V & g_V(2)h_V(2) & g_V(2)b_V(2) \\ z(3)w(1) & \sigma_2^V & g_V(3) \end{pmatrix} \tag{4.2}$$

and determine the matrices f_3^V, ϕ_3^V of sizes $2 \times 2, 2 \times r_3^V$ from the partition

$$\beta_3^V = \begin{bmatrix} f_3^V & \phi_3^V \end{bmatrix}. \tag{4.3}$$

(c) Compute

$$\begin{pmatrix} z^{(1)}(1) \\ \chi_3 \end{pmatrix} = Q_1^* \begin{pmatrix} z(1) \\ z(2) \\ z(3) \end{pmatrix}, \quad \gamma_2 = \begin{pmatrix} w(1) \\ w(2) \end{pmatrix} \tag{4.4}$$

with the number $z^{(1)}$ and two-dimensional columns χ_3, γ_2. Compute

$$f_3^A = f_3^V - \chi_3 \gamma_2^*, \quad \varphi_3^A = \phi_3^V. \tag{4.5}$$

(d) Set

$$c_2 = \begin{pmatrix} p(1) \\ p(2) \end{pmatrix}, \quad \theta_1 = q(1),\ \theta_2 = \begin{pmatrix} q(1) \\ q(2) \end{pmatrix}. \tag{4.6}$$

Compute

$$d_U(1) = d_B(1) + p(1)q(1), \quad d_U(2) = d_B(2) + p(2)q(2) \tag{4.7}$$

and set

$$f_2^U = \begin{pmatrix} d_U(1) & g_U(1)h_U(2) \\ p(2)q(1) & d_U(2) \end{pmatrix},\ \phi_2^U = \begin{pmatrix} g_U(1)b_U(2) \\ g_U(2) \end{pmatrix}, \tag{4.8}$$

$$\varepsilon = g_U(1)h_U(2) - p(1)q(2), \tag{4.9}$$

$$f_2^B = \begin{pmatrix} d_B(1) & \varepsilon \\ 0 & d_B(2) \end{pmatrix},\ \varphi_2^B = \phi_2^U. \tag{4.10}$$

– **Chasing the Bulge** For $k = 1, \ldots, N-1$ perform the following:

(a) *(Apply Q_k and determine Z_k).* Compute the two-dimensional column ε_{k+1}^B via

$$\varepsilon_{k+1}^B = \varphi_{k+1}^B h_U(k+2) - c_{k+1}q(k+2), \tag{4.11}$$

and the 3×3 matrix Φ_k by the formula

$$\Phi_k = Q_k^* \begin{pmatrix} f_{k+1}^B & \varepsilon_{k+1}^B \\ 0 & d_B(k+2) \end{pmatrix}. \tag{4.12}$$

Determine the 3×3 orthogonal matrix Z_k such that

$$\Phi_k(2:3,:)Z_k = \begin{pmatrix} 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix}. \tag{4.13}$$

(b) *(Determine Q_{k+1}).* Compute the column

$$\epsilon_{k+2}^A = \varphi_{k+2}^A h_V(k+2) - \chi_{k+2}w(k+2) \tag{4.14}$$

and the 3×3 matrix Ω_k by the formula

$$\Omega_k = \begin{pmatrix} f_{k+2}^A & \epsilon_{k+2}^A \\ 0 & \sigma_{k+2}^A \end{pmatrix} Z_k. \tag{4.15}$$

Determine the 3×3 orthogonal matrix Q_{k+1} and the number $(\sigma_k^A)^{(1)}$ such that

$$Q_{k+1}^* \Omega_k(:,1) = \begin{pmatrix} (\sigma_k^A)^{(1)} \\ 0 \\ 0 \end{pmatrix}. \tag{4.16}$$

(c) *(Update generators for U and B).* Compute

$$d_U(k+2) = d_B(k+2) + p(k+2)q(k+2), \tag{4.17}$$

$$\begin{pmatrix} \tilde{d}_U(k+2) & \tilde{g}_U(k+2) \\ \times & \beta_{k+2}^U \end{pmatrix} = Q_k^*\ \tilde{U}_k\ \begin{pmatrix} Z_k & 0 \\ 0 & I_{r_{k+2}^U} \end{pmatrix} \tag{4.18}$$

where

$$\tilde{U}_k = \begin{pmatrix} f_{k+1}^U & \phi_{k+1}^U h_U(k+2) & \phi_{k+1}^U b_U(k+2) \\ p(k+2)\theta_{k+1}^* & d_U(k+2) & g_U(k+2) \end{pmatrix}$$

and determine the matrices f_{k+2}^U, ϕ_{k+2}^U of sizes $2 \times 2, 2 \times r_{k+2}^U$ from the partition

$$\beta_{k+2}^U = \left[\, f_{k+2}^U\ \phi_{k+2}^U \,\right]. \tag{4.19}$$

Compute

$$\begin{gathered}\left(\tilde{h}_U(k+2)\ \tilde{b}_U(k+2)\right)= \\ \begin{pmatrix} I_2 & 0 & 0 \\ 0 & h_U(k+2) & b_U(k+2) \end{pmatrix} \begin{pmatrix} Z_k & 0 \\ 0 & I_{r^U_{k+2}} \end{pmatrix}.\end{gathered} \tag{4.20}$$

Compute

$$\begin{gathered}\begin{pmatrix} p^{(1)}(k) \\ c_{k+2} \end{pmatrix} = Q_k^* \begin{pmatrix} c_{k+1} \\ p(k+2) \end{pmatrix} \\ \begin{pmatrix} q^{(1)}(k) \\ \theta_{k+2} \end{pmatrix} = Z_k^* \begin{pmatrix} \theta_{k+1} \\ q(k+2) \end{pmatrix}\end{gathered} \tag{4.21}$$

with the numbers $p^{(1)}(k), q^{(1)}(k)$ and two-dimensional columns c_{k+2}, θ_{k+2}. Compute

$$f^B_{k+2} = f^U_{k+2} - c_{k+2}\theta^*_{k+2}, \quad \varphi^B_{k+2} = \phi^U_{k+2}. \tag{4.22}$$

(d) *(Update generators for V and A).* Compute

$$\sigma^V_{k+2} = \sigma^A_{k+2} + z(k+3)w(k+2), \tag{4.23}$$

$$\begin{gathered}\begin{pmatrix} \tilde{d}_V(k+3)\ \tilde{g}_V(k+3) \\ \times \qquad\quad \beta^V_{k+3} \end{pmatrix} = \\ Q^*_{k+1}\ \tilde{V}_{k+2}\ \begin{pmatrix} Z_k & 0 \\ 0 & I_{r^V_{k+3}} \end{pmatrix},\end{gathered} \tag{4.24}$$

where

$$\tilde{V}_{k+2} = \begin{pmatrix} f^V_{k+2} & \phi^V_{k+2}h_V(k+2) & \phi^V_{k+2}b_V(k+2) \\ z(k+3)\gamma^*_{k+1} & \sigma^V_{k+2} & g_V(k+3) \end{pmatrix}.$$

Determine the matrices f^V_{k+3}, ϕ^V_{k+3} of sizes $2 \times 2, 2 \times r^V_{k+3}$ from the partition

$$\beta^V_{k+3} = \left[\, f^V_{k+3}\ \phi^V_{k+3} \,\right]. \tag{4.25}$$

Compute

$$\begin{gathered}\left(\tilde{h}_V(k+3)\ \tilde{b}_V(k+3)\right)= \\ \begin{pmatrix} I_2 & 0 & 0 \\ 0 & h_V(k+2) & b_V(k+2) \end{pmatrix} \begin{pmatrix} Z_k & 0 \\ 0 & I_{r^V_{k+3}} \end{pmatrix},\end{gathered} \tag{4.26}$$

and

$$\begin{gathered}\begin{pmatrix} z^{(1)}(k+1) \\ \chi_{k+3} \end{pmatrix} = Q^*_{k+1} \begin{pmatrix} \chi_{k+2} \\ z(k+3) \end{pmatrix} \\ \begin{pmatrix} w^{(1)}(k) \\ \gamma_{k+2} \end{pmatrix} = Z^*_k \begin{pmatrix} \gamma_{k+1} \\ w(k+2) \end{pmatrix}\end{gathered} \tag{4.27}$$

with the numbers $z^{(1)}(k+1), w^{(1)}(k)$ and two-dimensional columns χ_{k+3}, γ_{k+2}. Compute

$$f^A_{k+3} = f^V_{k+3} - \chi_{k+3}\gamma^*_{k+2}, \quad \varphi^A_{k+3} = \phi^V_{k+3}. \tag{4.28}$$

– **Recovering of generators**

(a) Set

$$g_V^{(1)}(i-2) = \tilde{g}_V(i),\ i = 3, \ldots, N+2,$$
$$h_V^{(1)}(j-3) = \tilde{h}_V(j),\ j = 4, \ldots, N+3,$$
$$b_V^{(1)}(k-3) = \tilde{b}_V(k),\ j = 4, \ldots, N+2.$$

(b) Set

$$g_U^{(1)}(i-2) = \tilde{g}_U(i),\ i = 3, \ldots, N+1,$$
$$h_U^{(1)}(j-2) = \tilde{h}_U(j),\ j = 4, \ldots, N+2,$$
$$b_U^{(1)}(k-2) = \tilde{b}_U(k),\ k = 3, \ldots, N+1,$$
$$d_U^{(1)}(k-2) = \tilde{d}_U(k),\ k = 3, \ldots, N+2.$$

END

Remark 1 The complete algorithm incorporates the compression technique introduced in [5] to further process the generators returned by the algorithm by computing final upper quasiseparable generators $g_U^{(1)}(i)$ $(i = 1, \ldots, N-1))$, $h_U^{(1)}(j)$ $(j = 2, \ldots, N)$, $b_U^{(1)}(k)$ $(k = 2, \ldots, N-1)$ with orders not greater than one of the matrix U_1 and, moreover, upper triangular generators $g_V^{(1)}(i), h_V^{(1)}(i)$ $(i = 1, \ldots, N)$, $b_V^{(1)}(k)$ $(k = 1, \ldots, N-1)$ with orders not greater than two of the matrix V_1.

Remark 2 It can be interesting to compare the complexity and timings of the above algorithm versus the single-shift version presented in [5]. Roughly speaking, each iteration of double-shift Fast QZ requires about twice as many floating-point operations as single-shift Fast QZ; however, the double-shift version works in real arithmetic, whereas the single-shift algorithm requires complex operations. So we can expect a double-shift iteration to be $\mu/2$ times faster than a single-shift one, where μ is the speedup factor of real vs. complex arithmetic. A naïve operation count suggests that a complex addition requires two real flops and a complex multiplication requires six real flops; see also [Knuth, section 4.6.4] and MATLAB's old documentation page for the `flops` function. This yields on average $\mu \approx 4$, although in practice μ is more difficult to quantify; here, for practical purposes, we have used the experimental estimate given below.

For the computation of all eigenvalues, the double-shift algorithm is about $\rho\mu/2$ times faster than the single-shift version, where ρ is the ratio between the number of iterations needed to approximate a single eigenvalue with double shift and the number of iterations per eigenvalue with single shift. In practice, ρ is often close to 2, because each double-shift iteration approximates two eigenvalues instead of a single one, so the total number of iterations will be cut by one half.

Experiments done on the same machine and configuration used for the Fortran tests in Section 5 gave the following results:

– After testing on a large number of scalar $ax + y$ operations, the parameter μ was estimated at about 2. We used scalar operations for consistency with the structure of the algorithm. It should be pointed out, however, that experimental estimates of μ may depend on the machine and on the way the operations are computed, because the weight of increased storage and bandwidth may become

prominent. (The same experiment run on matrix-vector operations gives $\mu \approx 4$ as predicted by the operation count).
- For random polynomials we found $\rho \approx 2$, whereas in the case of cyclotomic polynomials the double-shift algorithm converged faster and ρ was closer to 3.
- Comparison on total running time showed double-shift QZ to be about twice as fast as the single-shift version in the case of random polynomials, and about three times as fast for cyclotomic polynomials, which is consistent with the discussion above.

In the next section we report the results of numerical experiment to illustrate the performance of the algorithm.

5 Numerical Results

The fast QZ algorithm for eigenvalue computation of structured pencils described in the previous section has been implemented in MATLAB and in Fortran 90.[2] The program deals with real companion–like pencils by applying the QZ method with single or double shift and it returns as output the list of real or complex conjugate paired approximations of the eigenvalues.

The design of a practical algorithm needs to account for various possible shifting strategies and deflation techniques. Deflation is an important concept in the practical implementation of the QR/QZ iteration. Deflation amounts to setting a small subdiagonal element of the Hessenberg matrix A to zero. This is called deflation because it splits the Hessenberg/triangular matrix pair into two smaller subproblems which may be independently refined further. We say that $a_{k+1,k}$ is negligible if

$$|a_{k+1,k}| \leq \mathtt{u}(|a_{k+1,k+1}| + |a_{k,k}|),$$

and then we set $a_{k+1,k} = 0$ and split the computation into two smaller eigenproblems. Here $\mathtt{u}$ denotes the machine precision. Another kind of deflation can happen in the matrix B and it is related to the occurrence of infinite eigenvalues. If $b_{k,k}$ is numerically zero then there exists at least an infinite eigenvalue and this can be deflated by moving up the zero entry to the top left corner of B. The criterion used in our implementation to check the nullity of $b_{k,k}$ is

$$|b_{k,k}| \leq \mathtt{u} \parallel B \parallel .$$

Eligible shift polynomials are generally determined from the (generalized) eigenvalues of the trailing principal submatrices of A and B We first compute the generalized eigenvalues (α_1, β_1) and (α_2, β_2) of the matrix pair $(A(n-1\colon n, n-1\colon n), B(n-1\colon n, n-1\colon n))$. If they correspond with a pair of complex conjugate numbers then we set

$$p(z) = (\beta_2 z - \alpha_2)(\beta_2 z - \alpha_2).$$

Otherwise we perform a linear shift, that is, $p(z) = \beta z - \alpha$, where the eigenvalue $\sigma = \alpha/\beta$ is the closest to the value $a_{N,N}/b_{N,N}$.

Our resulting algorithm has been tested on several numerical examples. We begin with some classical polynomials that are meant to test the algorithm for

[2] Both implementations are available for download at
`http://www.unilim.fr/pages_perso/paola.boito/software.html`.

Table 1 Timings and errors for the Fortran implementation of Fast QZ applied to random polynomials.

N	abs. forward error	average n. it.	Fast QZ time	LAPACK time
50	1.34e−14	1.82	1.19e−2	8.80e−3
100	1.09e−14	1.67	2.73e−2	9.70e−3
200	1.87e−14	1.59	8.84e−2	6.26e−2
300	3.03e−14	1.50	1.76e−1	1.97e−1
400	1.88e−13	1.46	3.12e−1	4.71e−1
500	8.08e−14	1.42	4.72e−1	1.18
600	4.73e−13	1.45	7.03e−1	2.32
700	2.19e−13	1.41	9.54e−1	4.04
800	1.46e−13	1.39	1.22	5.15
900	1.04e−13	1.37	1.50	9.00
1000	1.57e−13	1.39	1.90	13.06

speed and for backward stability. With the exception of Example 6, all polynomials are normalized so as to have 2-norm equal to 1: in practice, the algorithm is always applied to $p/\|p\|_2$. Absolute forward and backward errors for a polynomial $p(x) = \sum_{j=0}^{N} p_j x^j = p_N \prod_{k=1}^{N}(x - \alpha_k)$ are defined as

$$\text{forward error} = \max_{k=1,\ldots,N} |\alpha_k - \tilde{\alpha}_k|,$$
$$\text{backward error} = \max_{j=0,\ldots,N} |p_j - \tilde{p}_j|,$$

where $\{\tilde{\alpha}_k\}_{k=1,\ldots,N}$ are the computed roots, and $\{\tilde{p}_j\}_{j=0,\ldots,N}$ are the polynomial coefficients reconstructed from the computed roots, working in high precision. The polynomial $\tilde{p}(x) = \sum_{j=0}^{N} \tilde{p}_j x^j$ is also normalized so that $\|\tilde{p}\|_2 = 1$ prior to backward error computation.

Examples 1 and 2 use the Fortran implementation of Fast QZ, compiled with GNU Fortran compiler and running under Linux Ubuntu 14.04 on a laptop equipped with an Inter i5-2430M processor and 3.8 GB memory. All the other tests are based on the MATLAB version of the code and were run on a Mac Book Pro equipped with MATLAB R2016a.

Example 1 Fortran implementation applied to random polynomials. Polynomial coefficients are random real numbers uniformly chosen in $[-1, 1]$. Here N denotes the degree. Table 1 shows forward absolute errors w.r.t. the roots computed by LAPACK, as well as the average number of iterations per eigenvalue and the running times, in seconds, for LAPACK and Fast QZ. All the results are averages over 10 runs for each degree.

In this example, Fast QZ is faster than LAPACK for polynomials of degree larger than 250. (Of course, the results of timing comparisons may vary slightly depending on the machine and architecture). The quadratic growth of the running time for our algorithm is shown in Figure 1.

Example 2 Fortran implementation applied to cyclotomic polynomials. The polynomials used in this example take the form $p(x) = x^N - 1$. In this case we know the exact roots, which can be computed using the Fortran function `cos` and `sin`. We can therefore compute errors for Fast QZ and for Lapack, both with respect to

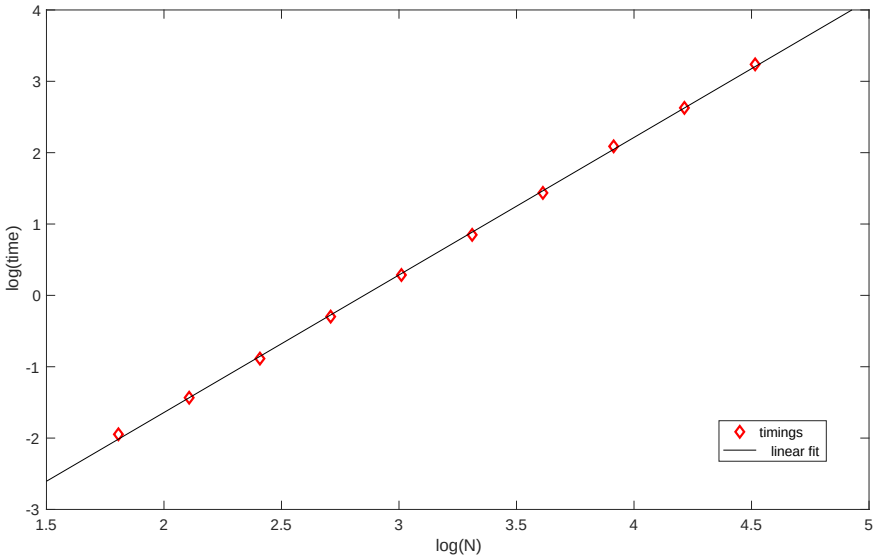

Fig. 1 This is a log-log plot of running times vs. polynomial degree N for Example 1. Here we have chosen N as powers of 2, from $N = 2^6 = 64$ to $N = 2^{15} = 32768$. The linear fit has equation $y = 1.93x - 5.50$, which is consistent with the $\mathcal{O}(N^2)$ complexity of Fast QZ.

Table 2 Timings and absolute forward errors for the Fortran implementation of Fast QZ applied to cyclotomic polynomials. Errors are computed w.r.t. "exact" roots.

N	err. Fast QZ	err. LAPACK	average n. it.	Fast QZ time	LAPACK time
100	4.65e−15	3.11e−15	1.38	3.00e−2	9.00e−3
200	5.31e−15	8.67e−15	1.25	7.90e−2	5.20e−2
300	6.76e−15	1.37e−14	1.19	1.52e−1	1.67e−1
400	1.05e−14	1.74e−14	1.16	2.82e−1	3.97e−1
500	9.49e−15	2.28e−14	1.14	4.03e−1	1.03
600	1.46e−14	2.85e−14	1.12	5.79e−1	2.01
700	1.51e−14	3.19e−14	1.12	7.85e−1	3.40
800	1.53e−14	3.90e−14	1.10	9.73e−1	5.39
900	1.93e−14	3.95e−14	1.10	1.24	8.00
1000	1.69e−14	4.84e−14	1.10	1.53	11.18
1500	3.00e−14	7.37e−14	1.09	3.35	41.47
2000	2.45e−14	1.02e−13	1.08	5.80	107.97

the "exact" roots: FastQZ turns out to be as accurate as LAPACK. Table 2 shows forward absolute errors, as well as the average number of iterations per eigenvalue and running times (in seconds). Figure 2 shows a logarithmic plot of the running times for Fast QZ, together with a linear fit.

Example 3 In this example we use a classical set of test polynomials taken from [22] . The polynomials are all of degree 20:

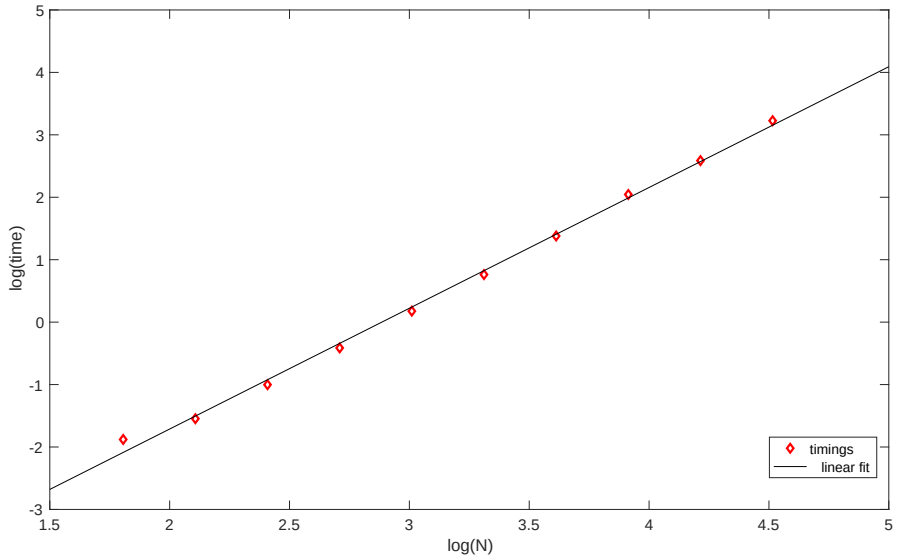

Fig. 2 Log-log plot of running times vs. polynomial degree for Example 2. The linear fit has equation $y = 1.93x - 5.58$.

1. the Wilkinson polynomial, i.e., $P(x) = \prod_{k=1}^{20}(x-k)$,
2. the polynomial with roots uniformly spaced in [-1.9, 1.9],
3. $P(x) = \sum_{k=0}^{20} x^k/k!$,
4. the Bernoulli polynomial of degree 20,
5. $P(x) = 1 + x + x^2 + \ldots + x^{20}$,
6. the polynomial with roots $2^{-10}, 2^{-9}, \ldots, 2^9$,
7. the Chebyshev polynomial of degree 20.

Table 3 shows absolute forward and backward errors for our fast QZ and for classical QZ applied to the companion pencil. For the purpose of computing forward errors we have taken as $\{\alpha_k\}_{k=1,\ldots,N}$ either the exact roots, if known, or numerical roots computed with high accuracy.

Forward errors may vary, consistently with the conditioning of the problem. However, backward errors are always of the order of the machine epsilon, which points to a backward stable behavior in practice.

Example 4 We apply here our structured algorithm to some polynomials taken from the test suite proposed by Jenkins and Traub in [15]. The polynomials are:

$p_1(x) = ((x-a)(x-1)(x+a))$, with $a = 10^{-8}$, 10^{-15}, 10^{8}, 10^{15},
$p_3(x) = \prod_{j=1}^{r}(x - 10^{-j})$, with $r = 10$, 20,
$p_4(x) = (x-0.1)^3(x-0.5)(x-0.6)(x-0.7)$,
$p_7 = (x-0.001)(x-0.01)(x-0.1)(x-0.1+ai)(x-0.1-ai)(x-1)(x-10)$,
with $a = 10^{-10}$,

Table 3 Forward and backward errors for a set of ill-conditioned polynomials. Note that the MATLAB implementation of classical QZ sometimes finds infinite roots, which prevent computation of the backward error. This behavior is denoted by the entry Inf.

$P(x)$	f. err. (fast QZ)	f. err. (classical QZ)	b. err. (fast QZ)	b. err. (classical QZ)
1	28.73	Inf	6.52e−16	Inf
2	5.91e−13	8.07e−13	8.07e−16	1.11e−15
3	5.70	Inf	2.22e−16	Inf
4	3.76e−10	1.83e−12	1.72e−15	1.20e−15
5	3.06e−15	1.09e−15	4.52e−15	1.58e−15
6	1.09e−2	2.30e−3	2.28e−15	3.05e−15
7	5.47e−11	1.68e−11	1.08e−15	1.91e−15

Table 4 Forward and backward errors for polynomials taken from Jenkins and Traub's test suite; see Example 4.

$P(x)$	f. err. (fast QZ)	f. err. (class. QZ)	b. err. (fast QZ)	b. err. (class. QZ)
$p_1(x)$, $a = 1e{-}8$	1.52e−8	1.00e−8	2.22e−16	1.11e−16
$p_1(x)$, $a = 1e{-}15$	1.64e−8	8.08e−16	1.90e−16	1.11e−16
$p_3(x)$, $r = 10$	8.76e−6	1.00e−6	8.60e−16	3.61e−16
$p_3(x)$, $r = 15$	1.25e−6	1.37e−6	6.80e−16	9.10e−16
$p_3(x)$, $r = 20$	1.99e−4	9.90e−7	3.14e−15	8.07e−16
$p_4(x)$	9.088e−6	4.26e−6	6.66e−16	3.33e−16
$p_7(x)$, $a = 1e{-}10$	1.91e−5	6.47e−6	2.77e−16	1.11e−16
$p_{10}(x)$, $a = 1e{+}3$	2.71e−16	0	1.91e−16	0
$p_{10}(x)$, $a = 1e{+}6$	1.16e−16	8.25e−18	8.20e−17	5.83e−18
$p_{10}(x)$, $a = 1e{+}9$	1.81e−16	0	1.28e−16	1.11e−16
$p_{11}(x)$, $m = 15$	1.11e−14	9.87e−15	3.45e−14	1.80e−14

$p_{10}(x) = (x-a)(x-1)(x-a^{-1})$, with $a = 10^3, 10^6, 10^9$,
$p_{11}(x) = \prod_{j=1-m}^{m-1}(x - e^{\frac{ij\pi}{2m}}) \prod_{j=m}^{3m} 0.9e^{\frac{ij\pi}{2m}}$, with $m = 15$.

In particular, the polynomial $p_1(x)$ is meant to test whether large or small zeros may pose a difficulty, the polynomial $p_3(x)$ can be used to test for underflow, the polynomials $p_4(x)$ and $p_7(x)$ test for multiple or nearly multiple roots, whereas $p_{10}(x)$ and $p_{11}(x)$ test for deflation stability. Table 4 shows absolute forward and backward errors, computed as in the previous example, for Fast QZ and classical QZ. Note that larger values of r for $p_3(x)$ tend to slow down convergence, so for $r = 20$ we needed to increase the allowed number of iterations per eigenvalue (before an exceptional shift is applied).

When QZ is tested on the polynomial $p_1(x)$ with large values of a, normalization of the coefficients inevitably leads to a numerically zero leading coefficient and therefore to infinite eigenvalues. In this case, both fast and classical QZ retrieve the root 1 with accuracy up to machine precision. Of course one may also try using the non-normalized polynomials, in which case Fast QZ finds roots $\{1, 1, -1\}$ and classical QZ finds roots $\{1, 0, 0\}$, up to machine precision.

Example 5 The jumping polynomial. This is a polynomial of degree 20 where the coefficients are heavily unbalanced and QZ applied to the companion pencil tends to work better than computing the eigenvalues of the companion matrix (see also [5]). The polynomial is defined as $p(x) = \sum_{k=0}^{20} p_k x^k$, where $p_k = 10^{6(-1)^{(k+1)}-3}$

Table 5 Forward and backward errors for several methods applied to a polynomial with highly unbalanced coefficients (Example 5).

method	forward error	backward error
fast QZ	2.78e−15	4.94e−15
classical QZ	1.49e−15	3.22e−15
balanced QR	2.46e−9	5.86e−9
unbalanced QR	1.68e−15	2.72e−15

for $k = 0, \ldots, 20$. Table 5 shows that Fast QZ is just as accurate as classical QZ, and more accurate than the MATLAB command `roots`.

Example 6 In order to test the behavior of backward error for non-normalized polynomials (that is, for unbalanced pencils), we consider polynomials of degree 50 with random coefficients and 2-norms ranging from 1 to 10^{14}. For each polynomial p we apply QZ (structured or unstructured) without normalization to compute its roots. Then we form a polynomial $\tilde{p}$ from the computed roots, working in high precision, and define the 2-norm absolute backward error as

$$\text{backward error}_2 = \min_{\alpha \in \mathbb{R}} \|p - \alpha \tilde{p}\|_2.$$

In practice, the value of α that minimizes the backward error is computed as $\alpha = \left(\sum_{i=0}^{N} p_i \tilde{p}_i\right) / \sum_{i=0}^{N} \tilde{p}_i^2$.

Figure 3 shows that in this example the backward error grows proportionally to $\|p\|_2^2$ and its behavior when using Fast QZ is very similar to the case of classical QZ (that is, the Matlab function `eig`). See also the analysis in [1].

Our algorithm has been tested on several numerical examples resulting from the linearization of nonlinear eigenvalue problems by using Lagrange type interpolation schemes. In particular, if $f\ \Omega \subseteq \mathbb{R} \ \rightarrow \ \mathbb{R}$ is analytic then increasingly accurate approximations of its zeros can be found by rootfinding methods applied to certain polynomial approximations of f. The unique polynomial of degree less than n interpolating the function $f(z)$ at the $N-$th roots of unity $z_k = e^{2\pi(k-1)/N}$, $1 \leq k \leq N$, can be expressed as

$$p(z) = (z^N - 1) \sum_{j=1}^{n} \frac{w_j f_j}{z - z_j},$$

where

$$f_j = f(z_j), \quad w_j = \Big(\prod_{k=1, k \neq j} (z_j - z_k) \Big)^{-1} = z_j / N, \quad 1 \leq j \leq N.$$

In [6] it was shown that the roots of $p(z)$ are the finite eigenvalues of the matrix pencil $F - zG$, $F, G \in \mathbb{C}^{(N+1)\times(N+1)}$, given by

$$F = \begin{bmatrix} 0 & -f_1/\xi_1 & \ldots & -f_N/\xi_N \\ w_1\xi_1 & z_1 & & \\ \vdots & & \ddots & \\ w_N\xi_N & & & z_N \end{bmatrix}, \quad G = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}, \tag{5.29}$$

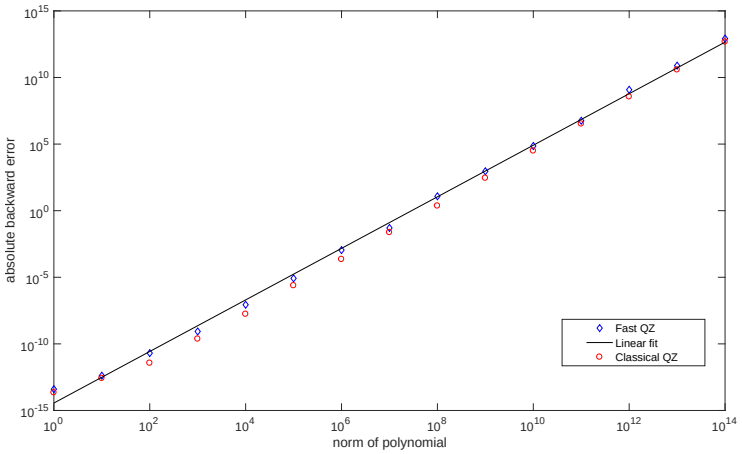

Fig. 3 Absolute backward error vs. polynomial norm for Example 6. The black line is a linear fit for the backward error of Fast QZ. Its equation is $y = 1.94x - 14.4$, which suggests that the absolute backward error grows proportionally to $\|p\|_2^2$.

where $\xi_1, \ldots, \xi_N$ are nonzero real numbers used for balancing purposes. Observe that since the size of the matrices is $N+1$ we obtain at least two spurious infinite eigenvalues. By a suitable congruence transformation $F \to F_1 = QFQ^*$ and $G \to G_1 = QGQ^*$ with Q orthogonal, we generate an equivalent real matrix pair (F_1, G_1) where $G_1 = G$ and F_1 is arrowhead with 2×2 orthogonal diagonal blocks. Then the usual Hessenberg/triangular reduction procedure can be applied by returning a final real matrix pair $(\tilde{A}, \tilde{B})$. One infinite eigenvalue can immediately be deflated by simply performing a permutation between the first and the second rows of $\tilde{A}$ and $\tilde{B}$ by returning a final matrix pair (A, B) belonging to the class $\mathcal{P}_N$. It can be shown that if $\xi_i = \xi$ for all i then this latter matrix pair is the companion pencil associated with the interpolating polynomial expressed in the power basis. Otherwise, if ξ_i are not constant then A is generally a dense Hessenberg matrix which can be represented as a rank one modification of an orthogonal matrix.

In the following examples we test the application of Fast QZ to the barycentric Lagrange interpolation on the roots of unity in order to find zeros of functions or solve eigenvalue problems. We point out here some implementation details:

- Scaling. The first row and column of the matrix F can be scaled independently without modifying G. We consistently normalize them so that $\|F(1,:)\|_2 = \|F(:,1)\|_2 = 1$, which makes the pencil more balanced.
- Deflation of infinite eigenvalues. Spurious infinite eigenvalues can be eliminated by applying repeatedly the permutation trick outlined above for the pencil $(\tilde{A}, \tilde{B})$. This leaves us, of course, with the problem of choosing a suitable deflation criterion. In practice, we perform this form of deflation when $|\tilde{A}(1,1)| < \varepsilon\sqrt{N}$, where ε is the machine epsilon.
- Reduction of the arrowhead pencil to Hessenberg/triangular form: this can be done in a fast (e.g., $O(N^2)$) way via Givens rotations that exploit structure, see e.g. [17], Section 2.2.2.

Table 6 Approximations of the zeros of $f(z) = \sin(z-0.3)\log(1.2-z)$. Here N is the number of interpolation points. See Example 7.

N	approx. of 0.2	approx. of 0.3
20	0.2153	0.2841
30	0.2014	0.2986
40	0.20016	0.29983
50	0.200021	0.299978
60	0.2000028	0.2999970
100	0.2000000011	0.2999999988
200	0.199999999999894	0.300000000000120

Example 7 This example is discussed in [2]. Consider the function $f(z) = \sin(z - 0.3)\log(1.2-z)$. We seek the zeros of f in the unit disk; the exact zeros are 0.2 and 0.3. Table 6 shows the computed approximations of these zeros for several values of N (number of interpolation points). The results are consistent with findings in [2], where 50 interpolation points yielded an accuracy of 4 digits.

Example 8 This is also an example from [2]. Define the matrix

$$A = \begin{pmatrix} 3.2 & 1.5 & 0.5 & -0.5 \\ -1.6 & 0.0 & -0.4 & 0.6 \\ -2.1 & -2.2 & 0.2 & -0.1 \\ 20.7 & 9.3 & 3.9 & -3.4 \end{pmatrix}.$$

We want to compute its eigenvalues by approximating the zeros of the polynomial $p(\lambda) = \det(A - \lambda I)$. The exact eigenvalues are 0.2, 0.3, 1.5 and -2. Interpolation plus Fast QZ using 6 nodes yields all the correct eigenvalues up to machine precision.

One may also apply a similar approach to the computation of the eigenvalues in the unit circle for a larger matrix. See Figures 4 and 5 for tests on two 100×100 matrices with random entries (uniformly chosen in [-1,1]).

Example 9 We consider some nonlinear eigenvalue problems taken from [3]:

1. `mobile_manipulator`: this 5×5 quadratic matrix polynomial is close to being nonregular;
2. `gen_tpal2`: a real T-palindromic quadratic matrix polynomial of size 16×16 whose eigenvalues lie on the unit circle;
3. `closed_loop`: the eigenvalues of this 2×2 parameterized quadratic polynomial lie inside the unit disc for a suitable choice of the parameter;
4. `relative_pose_5pt`: a 10×10 cubic matrix polynomial which comes from the five point relative pose problem in computer vision. See Figure 6 for a plot of the eigenvalues.

Table 7 shows the distance, in ∞-norm, between the eigenvalues computed via interpolation followed by Fast QZ and the eigenvalues computed via `polyeig`.

Example 10 Random matrix polynomials: we use matrix polynomials with random coefficients (given by the Matlab function `rand`). Table 8 shows errors with respect to `polyeig` for several values of the degree and of the size of the polynomial.

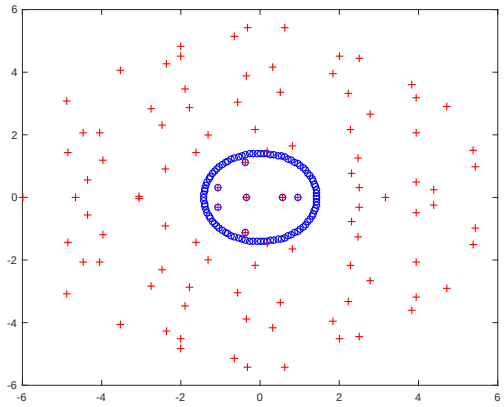

Fig. 4 Eigenvalues of a 100×100 random matrix; see Example 8. The blue circles are the eigenvalues computed via interpolation, the red crosses are the eigenvalues computed by `eig`. Here 120 interpolation nodes were used.

Table 7 Distance between the eigenvalues computed by interpolation+Fast QZ and `polyeig`, for some problems taken from the NLEVP suite (Example 9). Here N is the number of interpolation points. The error for the fourth problem is computed on all the eigenvalues (fourth line) and on the eigenvalues in the unit disk (fifth line, error marked by an asterisk.)

problem	error	N
`mobile_manipulator`	2.53e−15	20
`gen_tpal2`	1.61e−9	50
`closed_loop`	1.22e−15	10
`relative_pose_5pt`	2.81e−10	40
`relative_pose_5pt`	8.99e−15$^{(*)}$	40

Table 8 Distance between the eigenvalues computed by interpolation+Fast QZ and `polyeig`, for random matrix polynomials of different degrees and sizes (Example 10). The error is computed on the eigenvalues contained in the disk of center 0 and radius 2.

degree	size	error
10	5	1.12e−11
10	10	1.11e−9
10	20	2.93e−5
15	5	6.98e−9
15	10	5.15e−9
15	20	3.24e−4
20	5	2.13e−10
20	10	4.50e−9

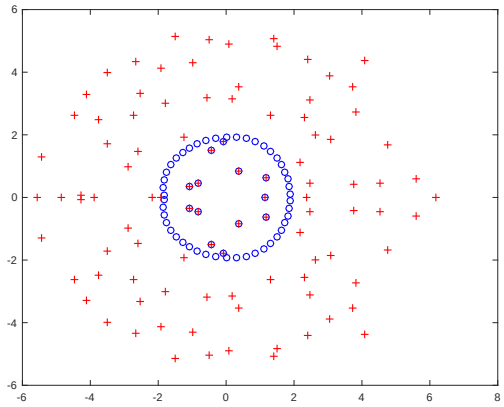

Fig. 5 Eigenvalues of a 100×100 random matrix; see Example 8. The blue circles are the eigenvalues computed via interpolation, the red crosses are the eigenvalues computed by `eig`. Here 60 interpolation nodes were used.

Example 11 We consider here a nonlinear, non polynomial example: the Lambert equation

$$w^6 \exp(w^6) = 0.1. \tag{5.30}$$

This equation has two real solutions

$$w = \pm W(0, 0.1) \approx \pm 0.671006$$

and complex solutions of the form

$$\begin{aligned} w &= \pm \left(W(\nu, x)\right)^{1/6}, \\ w &= \pm (-1)^{1/3} \left(W(\nu, x)\right)^{1/6}, \\ w &= \pm (-1)^{2/3} \left(W(\nu, x)\right)^{1/6}, \end{aligned}$$

where $W(\nu, x)$, with $\nu \in \mathbb{Z}$ and $x \in \mathbb{C}$, denotes the ν-th branch of the product-log function (Lambert function) applied to x.

Such solutions can be computed in Matlab using the `lambertw` function: in the following we will consider them as the "exact" solutions. We want to test the behavior of the "interpolation+QZ" approach in this case. Experiments suggest the following remarks:

– As expected, interpolation only "catches" roots inside the unit disk: see Figure 7. Since the roots of (5.30) are mostly outside the unit disk, we introduce a scaled version of the equation:

$$\alpha^6 w^6 \exp(\alpha^6 w^6) = 0.1, \tag{5.31}$$

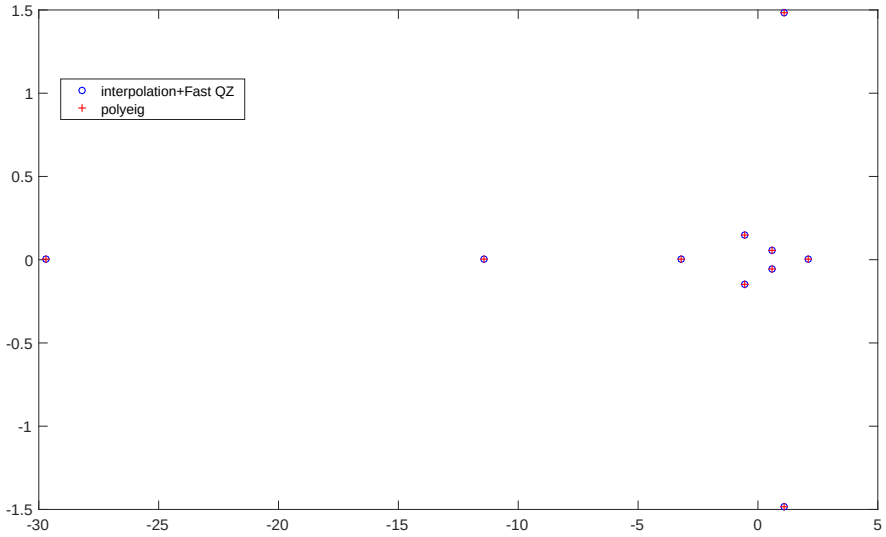

Fig. 6 Eigenvalues of the matrix polynomial `relative_pose_5pt`; see Example 9. The blue circles are the eigenvalues computed via interpolation, the red crosses are the eiganvalues computed by `polyeig`.

where $\alpha \geq 1$ is a scaling parameter. The drawback is that, as α grows, the matrix pencil becomes more unbalanced.

– A large number of interpolation nodes is needed (considerably larger than the number of approximated roots).

See Figure 8 for an example.

Tables 9, 10 and 12 show the accuracy of the approximation for several values of α and of the number of nodes. Here by "distance" we denote the distance in ∞-norm between the roots of (5.31) inside the unit disk and their approximations computed via interpolation followed by structured or unstructured QZ.

Table 9 Example 11: we take $\alpha = 1$ and there are 6 roots inside the unit circle. The number of nodes is denoted by N, taken as $N = 6k+1$ for some $k \in \mathbb{N}$, for symmetry. The accuracy in the approximation of the roots is the same for structured and unstructured QZ. An accuracy of about 10^{-15} is reached using 103 nodes.

N	distance
7	1.88e−1
19	2.53e−2
31	1.18e−3
43	2.80e−5
55	3.88e−7
67	3.52e−9

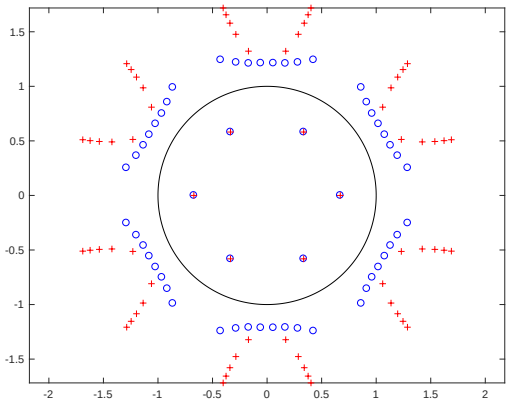

Fig. 7 This is a plot of the roots of (5.30) in the complex plane. Red crosses denote the "exact" roots computed by `lambertw` with ν up to 5. Blue circles are the roots computed via interpolation+QZ with 60 nodes. Note that only the 6 roots inside the unit circle (plotted in black for reference) are correctly approximated.

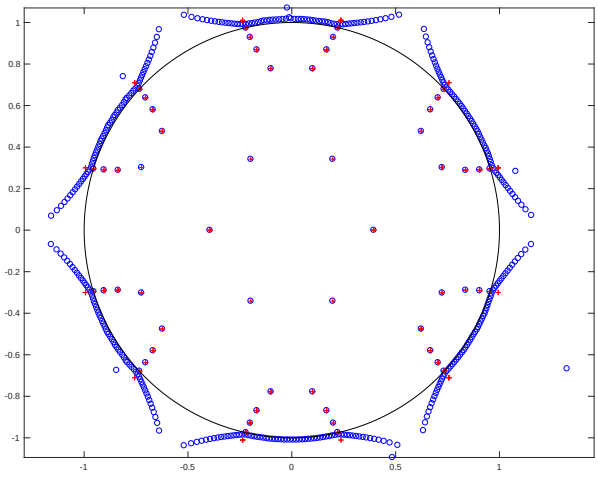

Fig. 8 This is a plot of the roots of (5.31) in the complex plane, with $\alpha = 1.7$, so that 54 roots are inside the unit circle (plotted in black for reference). Red crosses denote the "exact" roots computed by `lambertw` with ν up to 5. Blue circles are the roots computed via interpolation+QZ. Here we have taken 446 interpolation nodes.

Table 10 Example 11: we take $\alpha = 1.5$ and there are 18 roots inside the unit circle.

N	distance (unstructured)	distance (structured)
181	1.67e−1	1.67e−1
217	7.50e−4	7.50e−4
253	4.36e−7	4.36e−7
289	7.06e−10	1.40e−9

Table 11 Example 11: we take $\alpha = 1.6$ and there are 30 roots inside the unit circle.

N	distance (unstructured)	distance (structured)
301	1.62e−3	1.62e−3
361	9.73e−8	6.03e−7

Table 12 Example 11: we take $\alpha = 1.7$ and there are 54 roots inside the unit circle.

N	distance (unstructured)	distance (structured)
379	2.11e−1	2.11e−1
433	4.59e−4	1.34e−3

Example 12 This example comes from the discretization of a nonlinear eigenvalue problem governing the eigenvibrations of a string with an elastically attached mass: see e.g., [3], [8] and [21]. The original problem is given by

$$\begin{cases} -u''(x) = \lambda u(x) \\ u(0) = 0 \\ u'(1) + k\frac{\lambda}{\lambda - k/m}u(1) = 0 \end{cases}$$

where the parameters k and m correspond to the elastic constant and to the mass, respectively. We are interested in computing the two smallest real eigenvalues λ_1 and λ_2.

The discretization is applied on a uniform grid with nodes $x_i = i/n, i = 0, \ldots, n$ and step $h = 1/n$, yielding a nonlinear matrix eigenvalue problem of the form $K(\lambda)v = 0$ with

$$K(\lambda) = A - \lambda B + k\frac{\lambda}{\lambda - k/m}C,$$

where

$$A = \frac{1}{h}\begin{pmatrix} 2 & 1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & 2 & -1 \\ & & -1 & 1 \end{pmatrix}, \quad B = \frac{h}{6}\begin{pmatrix} 4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & 4 & 1 \\ & & 1 & 2 \end{pmatrix}, \quad C = e_n e_n^T$$

and $e_n = [0, \ldots, 0, 1]^T$. Here we choose $k = 2$ and $m = 1$. In this case the eigenvalues are known to be $\lambda_1 \approx 0.572224720810327$ and $\lambda_2 \approx 6.02588212472795$. Interpolation of $\det(K(\lambda))/\det(K(\lambda))'$ on the unit circle followed by FastQZ allows us to approximate λ_1, see Table 13. The same technique applied after a suitable translation of λ (here $\lambda \to \lambda - 6$) gives approximations for λ_2.

Note that, since $K(\lambda)$ is a rational function, we should make sure that its pole $\tilde{\lambda} = k/m$ does not lie in the unit disk, otherwise interpolation might not be able to detect the eigenvalues. Experiments with $k = 0.01$ and $m = 1$, for instance, showed that the first eigenvalue (in this case $\lambda_1 \approx 9.90067 \cdot 10^{-3}$) could not be computed via interpolation.

Table 13 Example 12: the table shows the absolute errors on λ_1 and λ_2 for several values of n (the number of nodes on the discretization grid). The number of interpolation nodes is taken as $N = 100$. The distance between the approximations computed by structured and unstructured QZ is always of the order of the machine epsilon.

n	error on λ_1	error on λ_2
100	5.19e−3	8.48e−2
500	1.04e−3	1.72e−2
1000	5.23e−4	8.61e−3
5000	1.05e−4	1.72e−3

6 Conclusions

In this paper we have developed and tested a fast structured version of the double-shift QZ eigenvalue method tailored to a particular class of real matrix pencils. This class includes companion pencils, as well as pencils arising from barycentric Lagrange interpolation. Numerical tests confirm the expected complexity gains with respect to the classical method and show that our fast algorithm behaves as backward stable in practice, while retaining an accuracy comparable to the nonstructured method.

We also propose an application to nonlinear eigenvalue problems using interpolation techniques. While preliminary experiments look promising, this approach deserves further investigation, which will be the subject of further work.

Acknowledgements: Thanks to Thomas Mach for useful suggestions concerning the Fortran implementation of Givens transformations.

References

1. J. L. Aurentz, T. Mach, L. Robol, R. Vandebril, and D. S. Watkins, *Roots of polynomials: on twisted QR methods for companion matrices and pencils*, arXiv:1611.02435 [math.NA], 2016.
2. A. P. Austin, P. Kravanja, and L. N. Trefethen, *Numerical algorithms based on analytic function values at roots of unity*, SIAM J. Numer. Anal. 52 (2014), 1795–1821.
3. T. Betcke, N. J Higham, V. Mehrmann, C. Schröder, F. Tisseur, *NLEVP: a collection of nonlinear eigenvalue problems*, ACM Trans. Math. Software, 39 (2013), 7–28.

4. D. A. Bini, P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg, *A Fast Implicit QR Eigenvalue Algorithm for Companion Matrices* , Linear Algebra and Applications 432 (2010), 2006-2031.
5. P. Boito, Y. Eidelman, and L. Gemignani, *Implicit QR for companion-like pencils*, Math. Comp. 85 (2016), 1753-1774.
6. R. Corless, *Generalized companion matrices for the Lagrange basis*, Proceedings EACA, 2004.
7. C. de Boor, *An Empty Exercise*, in ACM SIGNUM Newsletter, vol. 25 (4), Oct. 1990, 2–6.
8. C. Effenberger, *Robust solution methods for nonlinear eigenvalue problems*, Ph.D. thesis, EPFL, 2013.
9. Y. Eidelman, L. Gemignani, and I. Gohberg, *Efficient eigenvalue computation for quasiseparable Hermitian matrices under low rank perturbations*, Numerical Algorithms 47 (2008), 253-273.
10. Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), 293–324.
11. Y. Eidelman, I. Gohberg and I. Haimovici, *Separable type representations of matrices and fast algorithms. Volume 1. Basics. Completion problems. Multiplication and inversion algorithms*, Operator Theory: Advances and Applications, Birkhäuser, 2013.
12. Y. Eidelman, I. Gohberg, and I. Haimovici, *Separable type representations of matrices and fast algorithms. Volume 2. Eigenvalue method*, Operator Theory: Advances and Applications, Birkhäuser, 2013.
13. M. Fiedler and T. L. Markham, *Completing a matrix when certain entries of its inverse are specified*, Linear Algebra Appl., 74 (1986), 225–237.
14. G. H. Golub and C. F. Van Loan, *Matrix computations*, Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.
15. M. A. Jenkins and J. F. Traub, *Principles for testing polynomial zerofinding programs*, ACM Trans. Math. Software, 1 (1975), 26–34.
16. P. Lancaster, P. Psarrakos, *On the pseudospectra of matrix polynomials*, SIAM J. Matrix Anal. Appl., 27 (2005), 115–129.
17. P. W. Lawrence, *Eigenvalues methods for interpolation bases*, University of West Ontario, Electronic Thesis and Dissertation Repository, paper 1359 (2013).
18. P. W. Lawrence, *Fast reduction of generalized companion matrix pairs for barycentric Lagrange interpolants*, SIAM J. Matrix Anal. Appl. **34** (2013), no. 3, 1277–1300.
19. P. W. Lawrence and R. M. Corless, *Stability of rootfinding for barycentric Lagrange interpolants*, Numer. Algorithms **65** (2014), no. 3, 447–464.
20. V. Mehrmann and D. Watkins, *Polynomial eigenvalue problems with Hamiltonian structure*, Electron. Trans. Numer. Anal.,13 (2012), 106–118.
21. S. I. Solov'Ëv, *Preconditioned iterative methods for a class of nonlinear eigenvalue problems*, Linear Algebra Appl., 415 (2006), 210–229.
22. K.-C. Toh and L. N. Trefethen, *Pseudozeros of polynomials and pseudospectra of companion matrices*, Numer. Math. 68 (1994), 403-425.
23. D. S. Watkins, *Fundamentals of matrix computations*, Pure and Applied Mathematics (New York). Wiley-Interscience [John Wiley & Sons], New York, second edition, 2002.
24. D. S. Watkins, *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*, SIAM 2007.

Appendix

In this appendix we give a formal proof of the correctness of the algorithm stated in Section 4. Specifically, we prove the following:

Theorem 1 *Let $(A, B) \in \mathcal{P}_N$ be a matrix pair with an upper Hessenberg matrix $A = V - \boldsymbol{z}\boldsymbol{w}^*$ from the class $\mathcal{H}_N$ and an upper triangular matrix $B = U - \boldsymbol{p}\boldsymbol{q}^*$ from the class $\mathcal{T}_N$ with the unitary matrices $V \in \mathcal{V}_N, U \in \mathcal{U}_N$ and the vectors $\boldsymbol{z}, \boldsymbol{w}, \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^N$. Let $p(z) = \alpha + \beta z + \gamma z^2 \in \mathbb{R}[z]$ be a polynomial of degree at most 2. Let Q, Z be unitary*

matrices defined as in (2.8), (2.9) *where the matrices* Q_i *and* Z_i, $1 \le i \le N-1$, *are generated by the algorithm in Section 4. Then* $A_1 = Q^* * AZ$ *and* $B_1 = Q^* BZ$ *are upper Hessenberg and upper triangular, respectively, and, moreover,* $Q^* p(AB^{-1})\boldsymbol{e}_1 = \alpha \boldsymbol{e}_1$ *for a suitable scalar* $\alpha \in \mathbb{R}$.

Proof The property $Q^* p(AB^{-1})\boldsymbol{e}_1 = \alpha \boldsymbol{e}_1$ easily follows by construction. The proof of the remaining properties is constructive by showing that A_1 is upper Hessenberg and B_1 is upper triangular and then providing structured representations of the entries of their unitary components $V_1 = Q^* VZ$ and $U_1 = Q^* UZ$. We restrict ourselves to consider A_1 and V_1 since the computation of B_1 and U_1 and of the perturbation vectors can be treated in a similar way.

We treat A and V as block matrices with entries of sizes $m_i^A \times n_j^A$, $i, j = 1, \ldots, N+3$, where

$$\begin{gathered} m_1^A = \cdots = m_N^A = 1,\ m_{N+1}^A = m_{N+2}^A = m_{N+3}^A = 0, \\ n_1^A = 0,\ n_2^A = \cdots = n_{N+1}^A = 1,\ n_{N+2}^A = n_{N+3}^A = 0. \end{gathered} \tag{6.32}$$

Relative to this partition the matrix V has diagonal entries

$$\begin{gathered} d_V(1) \text{ to be the } 1 \times 0 \text{ empty matrix}, \\ d_V(k) = \sigma_{k-1}^V = \sigma_{k-1}^A + z(k)w(k-1),\ k = 2, \ldots, N, \\ d_V(N+1), d_V(N+2), d_V(N+3) \text{ to be the } 0 \times 1, 0 \times 0, 0 \times 0 \text{ empty matrices}, \end{gathered} \tag{6.33}$$

upper quasiseparable generators

$$\begin{gathered} \hat{g}_V(k) = g_V(k),\ k = 1, \ldots, N, \quad \hat{g}_V(N+1), g_V(N+2) \text{ to be the } 0 \times 0 \text{ empty matrices}, \\ \hat{h}_V(k) = h_V(k-1),\ k = 2, \ldots, N+1, \\ \hat{h}_V(N+2), \hat{h}_V(N+3) \text{ to be the } 0 \times 0 \text{ empty matrices}, \\ \hat{b}_V(k) = b_V(k-1),\ k = 2, \ldots, N, \\ \hat{b}_V(N+1), \hat{b}_V(N+2) \text{ to be the } r_N^V \times 0, 0 \times 0 \text{ empty matrices} \end{gathered} \tag{6.34}$$

and lower quasiseparable generators

$$\begin{gathered} p_V(k) = z(k),\ k = 2, \ldots, N, \\ p_V(N+1), p_V(N+2) \text{ to be the } 0 \times 1 \text{ empty matrices}, \\ q_V(1) \text{ to be the } 1 \times 0 \text{ empty matrix}, \\ q_V(k) = w(k-1),\ k = 2, \ldots, N+1, \\ a_V(k) = 1,\ k = 2, \ldots, N+1. \end{gathered} \tag{6.35}$$

Relative to the partition (6.32) the matrix A is a block upper triangular matrix with diagonal entries

$$\begin{gathered} d_A(1) \text{ to be the } 1 \times 0 \text{ empty matrix}, \\ d_A(k) = \sigma_{k-1}^A,\ k = 2, \ldots, N, \\ d_A(N+1), d_A(N+2) \text{ to be the } 0 \times 1, 0 \times 0 \text{ empty matrices}. \end{gathered} \tag{6.36}$$

Moreover using (2.1) we obtain upper quasiseparable of the matrix A relative to the partition (6.32) with orders

$$r_k^A = r_k^V + 1,\ k = 1, \dots, N, \quad r_{N+1}^A = r_{N+2}^A = 0 \tag{6.37}$$

by the formulas

$$\begin{gathered} g_A(k) = \begin{bmatrix} g_V(k) & -z(k) \end{bmatrix},\ k = 1, \dots, N, \\ g_A(N+1), g_A(N+2) \text{ to be the } 0 \times 0 \text{ empty matrices,} \end{gathered} \tag{6.38}$$

$$\begin{gathered} h_A(k) = \begin{bmatrix} h_V(k-1) \\ w(k-1) \end{bmatrix},\ k = 2, \dots, N+1, \\ h_A(N+2), h_A(N+3) \text{ to be the } 0 \times 0 \text{ empty matrix,} \\ b_A(k) = \begin{pmatrix} b_V(k-1) & 0 \\ 0 & 1 \end{pmatrix},\ k = 2, \dots, N, \\ b_A(N+1), b_A(N+2) \text{ to be the } (r_N^V + 1) \times 0, 0 \times 0 \text{ empty matrices.} \end{gathered} \tag{6.39}$$

Using (2.8) and setting

$$\begin{gathered} \tilde{S}_1^A = \tilde{S}_2^A = I_N, \quad \tilde{S}_i^A = \tilde{Q}_{i-2}^*,\ i = 3, \dots, N+1, \quad \tilde{S}_{N+2}^A = \tilde{S}_{N+3}^A = I_N, \\ \tilde{T}_1^A = \tilde{T}_2^A = \tilde{T}_3^A = I_N, \quad \tilde{T}_i^A = \tilde{Z}_{i-3},\ i = 4, \dots, N+2, \quad \tilde{T}_{N+3}^A = I_N \end{gathered} \tag{6.40}$$

we get

$$Q^* = \tilde{S}_{N+3}^A \cdots \tilde{S}_1^A, \quad Z = \tilde{T}_1^A \cdots \tilde{T}_{N+3}^A. \tag{6.41}$$

We have

$$\begin{gathered} \tilde{S}_1^A = \operatorname{diag}\{S_1^A, I_{N-1}\},\ \tilde{S}_2^A = \operatorname{diag}\{S_2^A, I_{N-2}\}; \\ \tilde{S}_k^A = \operatorname{diag}\{I_{k-2}, S_k^A, I_{N-k}\},\ k = 2, \dots, N; \\ \tilde{S}_{N+1}^A = \operatorname{diag}\{I_{N-2}, S_{N+1}^A\},\ \tilde{S}_{N+2}^A = \operatorname{diag}\{I_{N-1}, S_{N+2}^A\},\ \tilde{S}_{N+3}^A = \operatorname{diag}\{I_N, S_{N+3}^A\} \end{gathered}$$

with

$$\begin{gathered} S_1^A = 1,\ S_2^A = I_2, \quad S_k^A = Q_{k-2}^*,\ k = 3, \dots, N+1, \quad S_{N+2}^A = 1, \\ S_{N+3}^A \text{ to be the } 0 \times 0 \text{ empty matrix} \end{gathered} \tag{6.42}$$

and

$$\begin{gathered} \tilde{T}_1^A = \operatorname{diag}\{T_1^A, I_N\},\ \tilde{T}_2^A = \operatorname{diag}\{T_2^A, I_{N-1}\},\ \tilde{T}_3^A = \operatorname{diag}\{T_2^A, I_{N-2}\}; \\ \tilde{T}_k = \operatorname{diag}\{I_{k-4}, T_k^A, I_{N-k+1}\},\ k = 4, \dots, N+2; \\ \tilde{T}_{N+3}^A = \operatorname{diag}\{I_{N-1}, T_{N+3}^A\} \end{gathered}$$

with

$$\begin{gathered} T_1^A \text{ to be the } 0 \times 0 \text{ empty matrix,} \\ T_2^A = 1,\ T_3^A = I_2, \quad T_k^A = Z_{k-3},\ k = 4, \dots, N+2, \quad T_{N+3}^A = 1. \end{gathered} \tag{6.43}$$

We treat the lower Hessenberg matrix Q^* as a block matrix with entries of sizes $\tau_i^A \times m_j^A,\ i, j = 1, \dots, N+3$, where

$$\tau_1^A = \tau_2^A = 0, \quad \tau_3^A = \cdots = \tau_{N+2}^A = 1, \quad \tau_{N+3}^A = 0. \tag{6.44}$$

The matrix Q^* has the representation considered in Lemma 31.1 in [12] with the matrices S_k $(k = 1, \ldots, N+2)$ of sizes $(\tau_1^A + r_1^S) \times m_1^A$, $(\tau_k^A + r_k^S) \times (m_k^A + r_{k-1}^S)$ $(k = 2, \ldots, N+1)$, $\tau_{N+2}^A \times (m_{N+2}^A + r_{N+1}^S)$, where

$$r_1^S = 1, \quad r_k^S = 2,\ k = 2, \ldots, N), \quad r_{N+1}^S = 1,\ r_{N+2}^S = 0.$$

We treat the upper Hessenberg matrix Z as a block matrix with entries of sizes $n_i^A \times \nu_j^A$, $i, j = 1, \ldots, N+2$, where

$$\nu_1^A = \nu_2^A = \nu_3^A = 0, \quad \nu_4^A = \cdots = \nu_{N+3}^A = 1. \tag{6.45}$$

The matrix Z has the representation considered in Lemma 31.1 in [12] with the matrices T_k^A $(k = 1, \ldots, N+2)$ of sizes $n_1^A \times (\nu_1^A + r_1^*)$, $(n_k^A + r_{k-1}^*) \times (\nu_k^A + r_k^*)$ $(k = 2, \ldots, N+1)$, $(n_{N+2}^A + r_{N+1}^*) \times \nu_{N+2}^A$, where

$$r_1^* = 0,\ r_2^* = 1, \quad r_k^* = 2,\ k = 3, \ldots, N+1, \quad r_{N+2}^*.$$

Now we apply the structured multiplication algorithm for quasiseparable representations stated in Corollary 31.2 in [12] in order to determine diagonal entries $\tilde{d}_A(k)$ $(k = 3, \ldots, N+2)$ and quasiseparable generators $\tilde{q}(j)$ $(j = 3, \ldots, N+1)$; $\tilde{g}_A(i)$ $(i = 3, \ldots, N+2)$ of the matrix $A_1 = Q^* A Z$ as well as auxiliary variables $\beta_k^A, f_k^A, \phi_k^A, \varphi_k^A$. The matrix A_1 is obtained as a block one with entries of sizes $\tau_i^A \times \nu_j^A$, $i, j = 1, \ldots, N+3$.

For the variables $\beta_k = \beta_k^A, f_k = f_k^A, \phi_k = \phi_k^A$ used in Corollary 31.2 we use the partitions

$$\beta_k^A = \begin{bmatrix} f_k^A & \phi_k^A \end{bmatrix},\ \phi_k^A = \begin{bmatrix} \varphi_k^A & -\chi_k \end{bmatrix},\ k = 2, \ldots, N-1, \tag{6.46}$$

with the matrices $f_k^A, \varphi_k^A, \chi_k$ of sizes $2 \times 2, 2 \times r_k^V, 2 \times 1$. For $k = 1, \ldots, N-2$ combining Corollary 31.2 with (6.42), (6.43), (6.36) and (6.38),(6.39) we get

$$\begin{gathered}
\begin{pmatrix} \tilde{d}_A(k+3) & \tilde{g}_A(k+3) \\ \tilde{q}(k+3) & \beta_{k+3}^A \end{pmatrix} = \\
Q_{k+1}^* \begin{pmatrix} f_{k+2}^A & \phi_{k+2}^A h_A(k+2) & \phi_{k+2}^A b_A(k+2) \\ 0 & \sigma_{k+2}^A & g_A(k+3) \end{pmatrix} \begin{pmatrix} Z_k & 0 \\ 0 & I_{r_{k+3}^A} \end{pmatrix}, \\
k = 1, \ldots, N-3, \\
\begin{pmatrix} \tilde{d}_A(N+1) & \tilde{g}_A(N+1) \\ \tilde{q}(N+1) & \beta_{N+1}^A \end{pmatrix} = Q_{N-1}^* \begin{pmatrix} f_N^A & \phi_N^A h_A(N) \end{pmatrix} Z_{N-2}.
\end{gathered} \tag{6.47}$$

Using (6.46) and (6.39) we get

$$\phi_{k+2}^A h_A(k+2) = \epsilon_{k+2}^A, \quad k = 1, \ldots, N-2 \tag{6.48}$$

and

$$\phi_{k+2}^A b_A(k+2) = \begin{bmatrix} \varphi_{k+2}^A b_V(k+2) & -\chi_{k+2} \end{bmatrix}, \quad k = 1, \ldots, N-3 \tag{6.49}$$

with ϵ_{k+2}^A as in (4.14).

Inserting (6.48), (6.49) in (6.47) and using (6.46), (6.38) we obtain

$$\begin{gathered}\begin{pmatrix} \tilde{d}_A(k+3) & \times & \times & \times \\ \tilde{q}(k+3) & f^A_{k+3} & \varphi^A_{k+3} & -\chi_{k+3} \end{pmatrix} = \\ Q^*_{k+1} \begin{pmatrix} f^A_{k+2} & \epsilon^A_{k+2} & \varphi^A_{k+2} b_V(k+2) & -\chi_{k+2} \\ 0 & \sigma^A_{k+2} & g_V(k+3) & -z(k+3) \end{pmatrix} \begin{pmatrix} Z_k & 0 \\ 0 & I_{r^A_{k+3}} \end{pmatrix}, \\ k = 1, \ldots, N-3. \end{gathered} \tag{6.50}$$

From (6.50) using (4.15) we obtain the relations

$$\begin{pmatrix} \tilde{d}_A(k+3) \\ \tilde{q}(k+3) \end{pmatrix} = Q^*_{k+1} \begin{pmatrix} \Omega_k(1,1) \\ \Omega_k(2,1) \end{pmatrix} \tag{6.51}$$

and

$$\begin{gathered}\begin{pmatrix} \times & \times & \times \\ f^A_{k+3} & \varphi^A_{k+3} & -\chi_{k+3} \end{pmatrix} = \\ Q^*_{k+1} \begin{pmatrix} \Omega_k(1:2,2:3) & \varphi^A_{k+2} & -\chi_{k+2} \\ \Omega_k(3,2:3) & g_V(k+3) & -z(k+3) \end{pmatrix}, \quad k = 1, \ldots, N-4. \end{gathered} \tag{6.52}$$

From (6.51) using (4.16) we have

$$\tilde{d}_A(k+3) = (\sigma^A_k)^{(1)},\ k = 1, \ldots, N-2 \tag{6.53}$$

and

$$\tilde{q}(k+3) = 0,\ k = 1, \ldots, N-2. \tag{6.54}$$

The formulas (6.44) and (6.45) mean that $(\sigma^A_k)(1),\ k = 1, \ldots, N-2$ are subdiagonal entries of the matrix A_1 (treated as an usual scalar matrix). The equalities (6.54) imply that A_1 is an upper Hessenberg matrix.

Next we apply the structured multiplication algorithm stated in Lemma 31.1 in [12] to compute (block) upper quasiseparable generators $\tilde{g}_V(i)$ $(i = 1, \ldots, N+2)$, $\tilde{h}_V(j)$ $(j = 2, \ldots, N+3)$, $\tilde{b}_V(k)$ $(k = 2, \ldots, N+2)$ with orders

$$\tilde{r}^V_1 = r^V_1,\ \tilde{r}^V_2 = r^V_2 + 1, \quad \tilde{r}^V_k = r^V_k + 2,\ k = 3, \ldots, N,\ \tilde{r}^V_{N+1} = 2,\ \tilde{r}^V_{N+1} = 1$$

and diagonal entries $\tilde{d}_{V_1}(k)$ $(k = 1, \ldots, N+3)$ of the matrix $V_1 = Q^* V Z$. The matrix V_1 is obtained as a block one with entries of sizes $\tau^A_i \times \nu^A_j,\ i, j = 1, \ldots, N+3$, where the numbers τ^a_i, ν^A_j are defined in (6.44), (6.45).

Using Lemma 31.1 and (6.42), (6.43) we obtain that

$$\begin{gathered}\begin{pmatrix} \tilde{d}_V(k) & \tilde{g}_V(k) \\ \times & \mathit{\Gamma}_k \end{pmatrix} = \\ \begin{pmatrix} Q^*_{k-2} & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} f^V_{k-1} & \phi^V_{k-1} h_V(k-1) & \phi^V_{k-1} b_V(k-1) \\ z(k)\alpha_{k-1} & \sigma^V_{k-1} & g_V(k) \\ \alpha_{k-1} & w^*(k-1) & 0 \end{pmatrix} \begin{pmatrix} Z_{k-2} & 0 \\ 0 & I_{r^V_k} \end{pmatrix}, \\ \mathit{\Gamma}_k = \begin{bmatrix} f^V_k & \phi^V_k \\ \alpha_k & 0 \end{bmatrix}, \quad k = 4, \ldots, N, \end{gathered} \tag{6.55}$$

together with the relation (4.26).

From (6.55) we find that the auxiliary matrices α_k $(k = 3, \ldots, N)$ satisfy the relations

$$\alpha_3 = \begin{pmatrix} w(1) & w(2) \end{pmatrix}, \quad \begin{pmatrix} \times & \alpha_k \end{pmatrix} = \begin{pmatrix} \alpha_{k-1} & w(k-1) \end{pmatrix} Z_{k-3},\ k = 4, \ldots, N.$$

Comparing this with (4.4), (4.21) we get

$$\alpha_k = \gamma_{k-1}^*, \quad k = 3, \ldots, N. \tag{6.56}$$

Thus using (6.56) and (6.55) we obtain (4.24), (4.25).

Next we show that the auxiliary variables f_k^A, φ_k^A $(k = 3, \ldots, N)$ may be determined via relations (4.5), (4.28). Take γ_2 as in (4.4) and assume that for some k with $1 \le k \le N-2$ the relations

$$f_{k+2}^A = f_{k+2}^V - \chi_{k+2}\gamma_{k+1}^*, \quad \varphi_{k+2}^A = \phi_{k+2}^V \tag{6.57}$$

hold. By (4.15) and (6.52) we have

$$\begin{gathered} \begin{pmatrix} \tilde{d}_A(k+3) & \times & \times \\ 0 & f_{k+3}^A & \varphi_{k+3}^A \end{pmatrix} = \\ Q_{k+1}^* \begin{pmatrix} f_{k+2}^A & \epsilon_{k+2}^A & \phi_{k+2}^V b_V(k+2) \\ 0 & \sigma_{k+2}^A & g_V(k+3) \end{pmatrix} \begin{pmatrix} Z_k & 0 \\ 0 & I_{r_{k+3}^V} \end{pmatrix}. \end{gathered} \tag{6.58}$$

Using (4.23) and (4.14) we get

$$\begin{gathered} \begin{pmatrix} f_{k+2}^A & \epsilon_{k+2} \\ 0 & \sigma_{k+2}^A \end{pmatrix} = \\ \begin{pmatrix} f_{k+2}^V - \chi_{k+2}\gamma_{k+1}^* & \phi_{k+2}^V h_V(k+2) - \chi_{k+2} w(k+2) \\ z(k+3)\gamma_{k+1}^* - z(k+3)\gamma_{k+1}^* & \sigma_{k+2}^V - z(k+3)w(k+2) \end{pmatrix}. \end{gathered} \tag{6.59}$$

Thus combining (6.58) and (6.59) together and using (4.24), (4.25) and (4.27) we obtain (4.28).