

An algorithm for computing the eigenvalues of block companion matrices

Steven Delvaux Katrijn Frederix
Marc Van Barel

Report TW 538, April 2009



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

An algorithm for computing the eigenvalues of block companion matrices

Steven Delvaux Katrijn Frederix
Marc Van Barel

Report TW 538, April 2009

Department of Computer Science, K.U.Leuven

Abstract

In this paper we propose a method for computing the roots of a monic matrix polynomial. To this end we compute the eigenvalues of the corresponding block companion matrix C . This is done by implementing the QR algorithm in such a way that it exploits the rank structure of the matrix. Because of this structure, we can represent the matrix in Givens-weight representation. A similar method as in [S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu. A fast QR algorithm for companion matrices. *Operator Theory: Advances and Applications*, 179:111–143, 2007], the bulge chasing, is used during the QR iteration. For practical usage, matrix C has to be brought in Hessenberg form before the QR iteration starts. During the QR iteration and the transformation to Hessenberg form, the property of the matrix being unitary plus low rank numerically deteriorates. A method to restore this property is used.

Keywords : Givens-weight representation, companion matrix, rank structured matrix.

MSC : Primary : 65F, Secondary : 65F30, 65F15, 15A03.

An algorithm for computing the eigenvalues of block companion matrices

Steven Delvaux*, Katrijn Frederix†, Marc Van Barel†

April 1, 2009

Abstract

In this paper we propose a method for computing the roots of a monic matrix polynomial. To this end we compute the eigenvalues of the corresponding block companion matrix C . This is done by implementing the QR algorithm in such a way that it exploits the rank structure of the matrix. Because of this structure, we can represent the matrix in Givens-weight representation. A similar method as in [4], the bulge chasing, is used during the QR iteration. For practical usage, matrix C has to be brought in Hessenberg form before the QR iteration starts. During the QR iteration and the transformation to Hessenberg form, the property of the matrix being unitary plus low rank numerically deteriorates. A method to restore this property is used.

Keywords: Givens-weight representation, companion matrix, rank structured matrix.

AMS subject classifications: Primary: 65F, Secondary: 65F30, 65F15, 15A03.

1 Introduction

It is generally known that the roots of a monic matrix polynomial, $p(x) = \sum_{i=0}^n A_i x^i$ with $A_i \in \mathbb{C}^{p \times p}$ ($i = 0, \dots, n$) and $A_n = I_p$, of degree n coincide with the eigenvalues of the associated block companion matrix C ($pn = N$):

$$C = \begin{bmatrix} 0 & 0 & \dots & 0 & -A_0 \\ I_p & 0 & \dots & 0 & -A_1 \\ 0 & I_p & \dots & 0 & -A_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_p & -A_{n-1} \end{bmatrix} \in \mathbb{C}^{pn \times pn}. \quad (1)$$

Therefore algorithms for computing matrix eigenvalues, $Cx = \lambda x$, can be applied for approximating the roots of $p(x)$.

When all eigenvalues have to be computed, the QR algorithm is still the method of choice for general nonsymmetric eigenvalue problems, $Cx = \lambda x$. Recently other algorithms [5, 10, 11, 15] have arisen which replace the QR algorithm in the area of symmetric eigenvalue problems. The QR algorithm for nonsymmetric matrices uses $\mathcal{O}(N^2)$ storage and runs in $\mathcal{O}(N^3)$ time. For practical usage, matrix C has to be brought into an upper Hessenberg form H and then the QR iteration is

*Department of Mathematics, Katholieke Universiteit Leuven, Celestijnenlaan 200B, B-3001 Leuven (Heverlee), Belgium. email: Steven.Delvaux@wis.kuleuven.be. The first author is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium).

†Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium. email: {Katrijn.Frederix,Marc.VanBarel}@cs.kuleuven.be. The research was partially supported by the Research Council K.U.Leuven, project OT/05/40 (Large rank structured matrix computations), CoE EF/05/006 Optimization in Engineering (OPTEC), by the Fund for Scientific Research–Flanders (Belgium), G.0423.05 (RAM: Rational modelling: optimal conditioning and stable algorithms), and by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization). The scientific responsibility rests with its authors.

carried out on H accordingly. The advantage of reducing C to H , is that the Hessenberg form is invariant under QR iterations and the computational complexity of one step of the QR algorithm on a Hessenberg matrix is an order less than for a general matrix. To reduce dramatically the number of QR iterations needed for convergence, a suitable shift has to be incorporated in each QR -step.

The Matlab function `roots`, provides approximations to the roots of a polynomial ($p = 1$) and is based on the shifted QR iteration [13, 14, 16]:

$$\begin{aligned} H_k - \alpha_k I &= Q_k R_k, \\ H_{k+1} &= R_k Q_k + \alpha_k I = Q_k^H H_k Q_k, \end{aligned} \quad (2)$$

$k = 0, 1, \dots$ applied with $H_0 = C$, suitably balanced by means of diagonal scaling. The matrix Q_k^H denotes the conjugate transpose of Q_k . In the beginning of the algorithm H_0 will be sparse and structured but after some steps, the upper triangular part of H_k is dense and apparently with no structure (except that the matrix is in Hessenberg form). In this case, the arithmetic cost of each iteration is $\mathcal{O}(N^2)$ and the QR algorithm does not seem to take advantage of the initial structure of C .

In [1, 2], it is proven that the iterate matrices H_k in (2) have off-diagonal low rank structure. In these papers, this structure property is exploited, and the corresponding algorithms use only $\mathcal{O}(N)$ storage and run in $\mathcal{O}(N^2)$ time. The method proposed in [2], is computationally appealing but numerically unstable due to the appearance of an inverse in the formula.

Another approach is discussed in [4]. This method exploits the low rank structure by writing the iterate matrices H_k in terms of their Sequentially Semi-Separable (SSS) form. However, it does not explicitly work with this representation. In fact the method works with the QR representation where the Q -factor is a product of Givens transformations and the R -factor is in SSS form. While the algorithm proceeds, the off-diagonal low rank structure of the R -factor gradually deteriorates. A method to restore the structure is proposed in [4].

In this paper, a QR algorithm will be presented, which is based on the method proposed in [4], to efficiently compute the eigenvalues of the block companion matrix C in (1). Instead of working with the SSS representation another compact representation is used, the Givens-weight representation [6]. The major contribution of this paper is that in contrast to [4], our algorithm is applicable to matrix polynomials ($p \geq 1$) or, what is the same, for unitary plus low rank (p) matrices. The algorithm consists of three steps, the first step is the construction of the block companion matrix and the related input for the algorithm, the second step is the reduction to Hessenberg form if $p > 1$ and the last step is the computation of the eigenvalues by use of the QR iteration algorithm.

During the algorithm it is taken into account that the QR iterates of a companion matrix have off-diagonal low rank structure. The underlying reason for this is that the matrix C , as well as its iterate matrices, is the sum of a unitary and a low rank matrix. It is however observed that this property numerically deteriorates during the algorithm.

A method to restore the unitary plus low rank form is presented and is based on the method described in [3]. Two possible variants of this method are discussed as well. Because of the computational cost, this restoration procedure is only applied every fixed number of iteration steps.

The paper is organized as follows. Section 2 recalls some facts about the Givens-weight representation. Section 3 describes the method for computing the eigenvalues of the matrix C . It consists of two major parts, reduction to Hessenberg form and the QR iteration step. The method to restore the unitary plus low rank structure is explained as well. Section 4 contains the numerical experiments. Section 5 states a conclusion.

2 Givens-weight representation

As stated in the introduction the method proposed in this paper will be based on the Givens-weight representation. In this section we briefly summarize the main graphical notations concerning this

representation. For a more extended description the reader is referred to [6, Chapter 7].

The Givens-weight representation will be used to represent a class of matrices that we call *rank structured*. We say that a matrix A is rank structured if it has many off-diagonal submatrices of low rank, more precisely contiguous submatrices which are situated in the left bottom matrix corner. These submatrices are also called structured blocks. A toy example with three structure blocks is shown in Figure 1(a): we have there a 9×9 matrix A such that $\text{rank } A(2 : 9, 1 : 3) \leq 2$, $\text{rank } A(5 : 9, 1 : 5) \leq 3$ and $\text{rank } A(7 : 9, 1 : 7) \leq 2^*$. Every cross in the figure represents a matrix element. The Givens-weight representation yields a compact way to represent the rank structured part of the matrix A . The representation consists of a pair $(\{Q_k\}_{k=1}^K, W)$ where $K = 3$ for the example in Figure 1.

Each Q_k is a unitary operation of the form

$$Q_k = \begin{bmatrix} I & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & I \end{bmatrix}$$

where the I denote identity matrices of appropriate size, depending on k . The square block X also depends on k and will often be given as a product of Givens rotations, but this needs not concern us right now. The operations Q_k , $k = K, K - 1, \dots, 1$, can be thought of as a sequence of unitary operations acting on the rows of the given rank structured matrix, proceeding from bottom to top, and serving to create as many zero rows as possible in the subsequent structure blocks. A main feature is that each Q_k is only applied on a restricted number of columns, in fact only on the rank structured part of the matrix; the rightmost column to which Q_k is applied is called the *action radius* of Q_k . The action radius is a function that decreases with decreasing k ; in the case of Figure 1(b)–(d) the action radii of Q_3 , Q_2 and Q_1 are given by 7, 5, and 3, respectively. In these figures, the entries which are changed by the action of the Q_k , $k = 3, 2, 1$ are shown in grey while the created zero rows are shown as blank. The operations Q_k themselves are shown as arrows, positioned to the left of the matrix, indicating those rows on which the nontrivial part of the action takes place.

On the other hand, the second ingredient of the Givens-weight representation is the matrix W which is called the *weight matrix*. This matrix contains for each structure block the elements obtained at the top border of the structure block at the moment just after applying Q_K, Q_{K-1}, \dots, Q_k where Q_k is the last relevant operation for that structure block.

This construction is illustrated in Figure 1(a)–1(d). Let us comment on this figure. The first step of the computation of the Givens-weight representation is the computation of Q_3 , having action radius 7. This operation serves to create zeros in the bottom row of the bottommost structure block, i.e., it serves to guarantee that

$$\tilde{W}(7 : 9, 1 : 7) = Q_3 A(7 : 9, 1 : 7)$$

has a zero row $\tilde{W}(9, 1 : 7)$ on the bottom. This is shown in Figure 1(b).

The second step of the procedure is the computation of Q_2 , having action radius 5, such that

$$\tilde{W}(5 : 8, 1 : 5) = Q_2 \begin{bmatrix} A(5 : 6, 1 : 5) \\ \tilde{W}(7 : 8, 1 : 5) \end{bmatrix}$$

has a zero row on the bottom, see Figure 1(c). The third and last step is the computation of Q_1 , having action radius 3, such that

$$W(2 : 7, 1 : 3) = Q_1 \begin{bmatrix} A(2 : 4, 1 : 3) \\ \tilde{W}(5 : 7, 1 : 3) \end{bmatrix}$$

has four zero rows in the bottom. This is shown in Figure 1(d) which is also the final Givens-weight representation. Note that the resulting weight matrix W consists of three non-trivial

*The notation is Matlab like notation: the colon operation has to be interpreted as follows: $i : m = [i, i+1, \dots, m]$ and $A(i : m, j : n) = A_{i:m, j:n}$ denotes the submatrix of A whose entries lie on the intersection of rows i, \dots, m and columns j, \dots, n .

Notice also that the product of two Givens transformations with the same row index k is again a Givens transformation with that row index, i.e. $G_k \bar{G}_k = \tilde{G}_k$. Throughout this paper this process is called a *fusion*. A graphical illustration is ($k = 1$):

$$\begin{array}{ccc} \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} & \longrightarrow & \begin{array}{|c|} \hline \text{---} \\ \hline \end{array} \\ G_1 & \bar{G}_1 & \tilde{G}_1 \end{array}$$

In the next sections, \bar{G}_k denotes the Givens transformation used to generate a bulge at $R_{k+1,k}$, \tilde{G}_k denotes the Givens transformation used to eliminate the bulge at $R_{k+1,k}$, $G^{(s)}$ denotes the Givens transformation which is obtained after application of the shift-through lemma and $G^{(f)}$ denotes the resulting Givens transformation after a fusion.

3.2 Initial QR factorization of C

Instead of working with the sparse matrix C we will work with its QR factorization $C = QR$ where matrix Q is represented as a product of Givens transformations and the R -factor is represented in Givens-weight representation. This representation is beneficial because of its sparse structure and despite the fact that matrix C is sparse in the beginning it will not stay like this during the algorithm. So we need to describe now how to compute the QR factors of matrix C and the Givens-weight representation of the R -factor.

To obtain the QR factorization, the block Hessenberg matrix C in (1) is brought into upper triangular form by applying successively Givens transformations to its rows. To remove the $p(n-1)$ ones on the p -th subdiagonal of matrix C , p Givens transformations of the form $(G_T)_k$ ($k = l + p - 1, \dots, l$), as defined in (3), are applied to the rows of the matrix, thereby bringing the subsequent columns l ($l = 1, \dots, p(n-1)$) in upper triangular form. For instance, for the first column ($p = 2, N = 6$):

$$(G_T)_1(G_T)_2 C(1:6, 1) = (G_T)_1(G_T)_2 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{with } (G_T)_k = \begin{bmatrix} I_{k-1} & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & I_{N-k-1} \end{bmatrix}. \quad (3)$$

After applying all these Givens transformations, the matrix is upper triangular except for its last p columns. To deal with the latter, we apply Givens transformations G_k ($k = pn-1, \dots, l$) to bring the subsequent columns $l = p(n-1) + 1, \dots, pn-1$ in upper triangular form. This results in

$$Q^H C = R = \begin{bmatrix} I_p & 0 & 0 & \dots & \tilde{A}_0 \\ 0 & I_p & 0 & \dots & \tilde{A}_1 \\ 0 & 0 & I_p & \dots & \tilde{A}_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \tilde{U}_{n-1} \end{bmatrix}, \quad (4)$$

with Q^H the product of all the applied Givens transformations and \tilde{U}_{n-1} an upper triangular matrix. The matrix R in (4) can be rewritten as $R = I + UV^H$, where

$$U = \begin{bmatrix} \tilde{A}_0 \\ \tilde{A}_1 \\ \tilde{A}_2 \\ \vdots \\ \tilde{U}_{n-1} - I_p \end{bmatrix} \in \mathbb{C}^{pn \times p} \quad \text{and} \quad V = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ I_p \end{bmatrix} \in \mathbb{C}^{pn \times p}.$$

This provides a unitary plus rank p decomposition of the R -factor.

As described before (Section 3.2), we will work with the QR decomposition of C , instead of working with matrix C itself. The matrix Q is a product of Givens transformations and matrix R is given by a column-based Givens-weight representation, as shown in Figure 2(d). In the explanation below, the R -factor is represented in full form, i.e. no structure is exploited. This is only done for clarity of exposition; in reality one should update the Givens-weight representation of R under the applied transformations and this will be described in Section 3.5.

For a general matrix $C \in \mathbb{C}^{N \times N}$, the Hessenberg reduction algorithm [13] consists of a sequence of similarity transformations

$$C_k = \bar{Q}_k^H C_{k-1} \bar{Q}_k, \quad k = 1, 2, \dots, N-2, \quad (6)$$

with $C_0 := C$, and where the unitary matrix \bar{Q}_k is a product of Givens transformations of the form $\bar{Q}_k = G_{N-1}^{(k)} \dots G_{k+1}^{(k)}$, constructed in such a way that C_k has Hessenberg structure in its first k columns. In our case the matrix C has a lower banded structure so the matrices \bar{Q}_k will usually consist of less Givens transformations.

For $k = 1$ the Givens transformations constituting \bar{Q}_1 can be read off immediately from the previously computed Q -factor in Section 3.2. For instance, the unitary transformation $Q_1^H = (G_T)_1 \dots (G_T)_p$ in (5) takes care that the first column of C becomes upper triangular. To bring the first column in Hessenberg form it suffices to put

$$\bar{Q}_1^H = (G_T)_2 \dots (G_T)_p.$$

To find the \bar{Q}_k for $k > 1$ we need to do more work. We will explain this for a 6×6 matrix with $p = 2$. The QR factorization of C is the following (according to (5)) $C = QR = Q_1 Q_2 Q_3 Q_4 Q_5 \cdot R$. Set $C_0 = C$, $R_0 = R$, $Q = G_2^{(1)} G_1^{(1)} Q_2 Q_3 Q_4 Q_5 = G_2^{(1)} G_1^{(1)} W_0$.

1. The first column is brought in Hessenberg form. As mentioned above this is achieved by setting $\bar{Q}_1 := G_2^{(1)}$. Let

$$C_1 := \bar{Q}_1^H C_0 \bar{Q}_1 = G_1^{(1)} \cdot W_0 \cdot R_0 \bar{Q}_1.$$

A graphical illustration of this factorization is

$$\begin{array}{c|c|c} & G_1^{(1)} & W_0 & R_0 \bar{Q}_1 \\ \hline 1 & | & & \times \times \times \times \times \\ 2 & | & & \times \times \times \times \times \\ 3 & | & & + \times \times \times \times \\ 4 & | & & \times \times \times \\ 5 & | & & \times \times \\ 6 & | & & \times \end{array}$$

The R -factor is not upper triangular anymore, a bulge (plus sign) appeared at position (3, 2). An extra Givens transformation $(\tilde{G}_2^{(1a)})^H$ (set $\tilde{Q}_1^H \leftarrow (\tilde{G}_2^{(1a)})^H$) has to be applied at the left of the matrix $R_0 \bar{Q}_1$ such that $\tilde{Q}_1^H R_0 \bar{Q}_1$ becomes upper triangular (set $R_1 \leftarrow \tilde{Q}_1^H R_0 \bar{Q}_1$). The extra unitary transformation \tilde{Q}_1 is incorporated into the Q -factor by use of the shift-through lemma. In fact, the term $W_0 \tilde{Q}_1$ has to be rearranged. This is first shown in formula form and then graphically in Figure 3. In the figure the bold lines denote the Givens transformations which play a role during the application of the shift-through lemma, the dotted line is the Givens transformation \tilde{Q} which has to be incorporated in the Q -factor. This notation is also

used in figures further on.

$$\begin{aligned}
C_1 &= G_1^{(1)} \cdot W_0 \tilde{Q}_1 \cdot \tilde{Q}_1^H R_0 \bar{Q}_1 \\
&\stackrel{(a)}{=} G_1^{(1)} \cdot G_3^{(2)} G_2^{(2)} \overleftarrow{G_4^{(3)} G_3^{(3)}} Q_4 Q_5 \overleftarrow{\tilde{G}_2^{(1a)}} \cdot R_1 \quad (\text{Decompose } W_0, \text{ set } R_1 \leftarrow \tilde{Q}_1^H R_0 \bar{Q}_1) \\
&= G_1^{(1)} \cdot G_3^{(2)} G_4^{(3)} \left(G_2^{(2)} G_3^{(3)} \tilde{G}_2^{(1a)} \right) Q_4 Q_5 \cdot R_1 \quad (G_4^{(3)}, \tilde{G}_2^{(1a)} \text{ pushed inward}) \\
&\stackrel{(b)}{=} G_1^{(1)} \cdot \left(G_3^{(2)} G_4^{(3)} G_3^{(s1)} \right) G_2^{(s1)} \hat{G}_3^{(s1)} Q_4 Q_5 \cdot R_1 \quad (\text{Shift-through lemma}) \\
&\stackrel{(c)}{=} G_1^{(1)} \cdot G_4^{(s2)} G_3^{(s2)} \hat{G}_4^{(s2)} \overleftarrow{G_2^{(s1)} \hat{G}_3^{(s1)}} Q_4 Q_5 \cdot R_1 \quad (\text{Shift-through lemma}) \\
&\stackrel{(d)}{=} G_1^{(1)} \cdot \underbrace{G_4^{(s2)} G_3^{(s2)}}_{\bar{Q}_2} G_2^{(s1)} \underbrace{\hat{G}_4^{(s2)} \hat{G}_3^{(s1)}}_{W_1} Q_4 Q_5 \cdot R_1 \quad (G_2^{(s1)} \text{ pushed inward}).
\end{aligned}$$

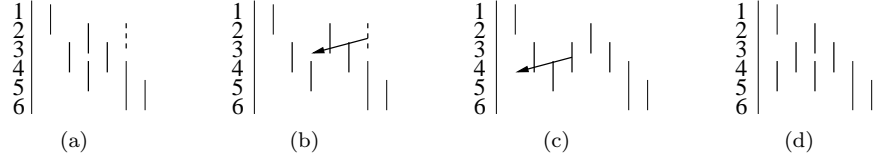


Figure 3: Graphical representation of what happens with the Q -factor: a) decompose W_0 , b) shift-through lemma, c) shift-through lemma, d) final result.

- The second column is brought in Hessenberg form. Set $\bar{Q}_2 \leftarrow G_4^{(s2)} G_3^{(s2)}$, then

$$C_2 := \bar{Q}_2^H C_1 \bar{Q}_2 = G_1^{(1)} G_2^{(s1)} \cdot W_1 \cdot R_1 \bar{Q}_2.$$

The matrix $R_1 \bar{Q}_2$ is not upper triangular, three bulges appear at the positions (4, 3), (5, 3) and (5, 4).

$$\begin{array}{c}
G_1^{(1)} G_2^{(s1)} W_1 \quad R_1 \bar{Q}_2 \\
\begin{array}{c|c}
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{array}{c} | \\ | \\ | \\ | \\ | \\ | \end{array} \\
\hline
\begin{array}{c} | \\ | \\ | \\ | \\ | \\ | \end{array} & \begin{array}{c} \times \times \times \times \times \times \\ \times \times \times \times \times \\ \times \times \times \times \\ + \times \times \times \\ + + \times \times \\ \times \end{array}
\end{array}
\end{array}$$

To restore the upper triangular form of the R -factor a sequence of extra (at most p) Givens transformations are applied such that $\bar{Q}_2^H R_1 \bar{Q}_2$ is upper triangular (set $\tilde{Q}_2 \leftarrow \tilde{G}_4^{(2a)} \tilde{G}_3^{(2b)}$). The extra Givens transformations $\tilde{G}_4^{(2a)}$ and $\tilde{G}_3^{(2b)}$ are incorporated into the Q -factor by use of the shift-through lemma. In fact, the terms in $W_1 \bar{Q}_2$ have to be rearranged. This is first

shown in formula form and then graphically in Figure 4.

$$\begin{aligned}
C_2 &= G_1^{(1)} G_2^{(s1)} \cdot \hat{G}_4^{(s2)} \hat{G}_3^{(s1)} G_5^{(4)} G_4^{(4)} G_5^{(5)} \tilde{G}_4^{(2a)} \tilde{G}_3^{(2b)} \cdot \tilde{Q}_2^H R_1 \bar{Q}_2 \\
&= G_1^{(1)} G_2^{(s1)} \cdot \hat{G}_4^{(s2)} \hat{G}_3^{(s1)} G_5^{(4)} \left(G_4^{(4)} G_5^{(5)} \tilde{G}_4^{(2a)} \right) \tilde{G}_3^{(2b)} \cdot R_2 \quad (\text{Set } R_2 \leftarrow \tilde{Q}_2^H R_1 \bar{Q}_2) \\
&\stackrel{(a)}{=} G_1^{(1)} G_2^{(s1)} \cdot \hat{G}_4^{(s2)} \hat{G}_3^{(s1)} \underline{G_5^{(4)} G_5^{(s3)} G_4^{(s3)} \hat{G}_5^{(s3)} \tilde{G}_3^{(2b)}} \cdot R_2 \quad (\text{Shift-through lemma}) \\
&\stackrel{(b)}{=} G_1^{(1)} G_2^{(s1)} \cdot \hat{G}_4^{(s2)} \hat{G}_3^{(s1)} \underline{G_5^{(f1)} G_4^{(s3)} \hat{G}_5^{(s3)} \tilde{G}_3^{(2b)}} \cdot R_2 \quad (\text{Fusion}) \\
&= G_1^{(1)} G_2^{(s1)} \cdot \hat{G}_4^{(s2)} G_5^{(f1)} \left(\hat{G}_3^{(s1)} G_4^{(s3)} \tilde{G}_3^{(2b)} \right) \hat{G}_5^{(s3)} \cdot R_2 \quad (G_5^{(f1)}, \tilde{G}_3^{(2b)} \text{ pushed inward}) \\
&\stackrel{(c)}{=} G_1^{(1)} G_2^{(s1)} \cdot \left(\hat{G}_4^{(s2)} G_5^{(f1)} G_4^{(s4)} \right) G_3^{(s4)} \hat{G}_4^{(s4)} \hat{G}_5^{(s3)} \cdot R_2 \quad (\text{Shift-through lemma}) \\
&\stackrel{(c)}{=} G_1^{(1)} G_2^{(s1)} \cdot G_5^{(s5)} G_4^{(s5)} \hat{G}_5^{(s5)} \underline{G_3^{(s4)} \hat{G}_4^{(s4)} \hat{G}_5^{(s3)}} \cdot R_2 \quad (\text{Shift-through lemma}) \\
&\stackrel{(d)}{=} G_1^{(1)} G_2^{(s1)} \cdot \underbrace{G_5^{(s5)} G_4^{(s5)} G_3^{(s4)}}_{\bar{Q}_3} \underbrace{\hat{G}_5^{(s5)} \hat{G}_4^{(s4)} \hat{G}_5^{(s3)}}_{W_2} \cdot R_2 \quad (G_3^{(s4)} \text{ pushed inward and fusion}).
\end{aligned}$$

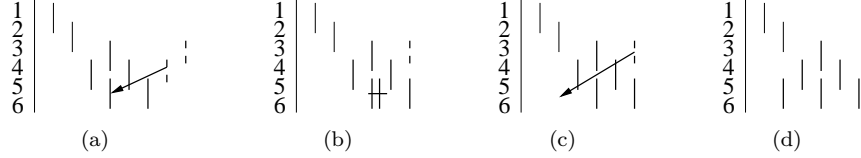


Figure 4: Graphical representation of what happens with the Q -factor: a) decompose W_1 , and apply shift-through lemma, b) fusion, c) shift-through lemma, f) final result.

3. The third column is brought in Hessenberg form. Set $\bar{Q}_3 \leftarrow G_5^{(s5)} G_4^{(s5)}$ then

$$C_3 := \bar{Q}_3^H C_2 \bar{Q}_3 = G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot W_2 \cdot R_2 \bar{Q}_3.$$

The matrix $R_2 \bar{Q}_3$ is not upper triangular, three bulges appear at positions (5, 4), (6, 4) and (6, 5).

$$\begin{array}{c}
G_1^{(1)} G_2^{(s1)} G_3^{(s4)} W_2 \quad R_2 \bar{Q}_3 \\
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6
\end{array}
\begin{array}{c}
| \\
| \\
| \\
| \\
| \\
|
\end{array}
\begin{array}{c}
\text{---} \\
\text{---} \\
\text{---} \\
\text{---} \\
\text{---} \\
\text{---}
\end{array}
\begin{array}{c}
\times \times \times \times \times \\
\times \times \times \times \times \\
\times \times \times \times \times \\
\times \times \times \times \\
+ \times \times \\
+ + \times
\end{array}
\end{array}$$

To restore the upper triangular form of the R -factor a sequence of extra Givens transformations are applied such that $\tilde{Q}_3^H R_2 \bar{Q}_3$ is upper triangular (set $\tilde{Q}_3 \leftarrow \tilde{G}_5^{(3a)} \tilde{G}_4^{(3b)}$). How these Givens transformations $\tilde{G}_5^{(3a)}$ and $\tilde{G}_4^{(3b)}$ are incorporated in the Q -factor ($W_2 \tilde{Q}_3$) is

first shown in formula form and then graphically in Figure 5.

$$\begin{aligned}
C_3 &= G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot \hat{G}_5^{(s5)} \hat{G}_4^{(s4)} \hat{G}_5^{(s3)} \tilde{G}_5^{(3a)} \tilde{G}_4^{(3b)} \cdot \tilde{Q}_3^H R_2 \bar{Q}_3 \\
&\stackrel{(a)}{=} G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot \hat{G}_5^{(s5)} \hat{G}_4^{(s4)} \underline{\hat{G}_5^{(s3)} \tilde{G}_5^{(3a)} \tilde{G}_4^{(3b)}} \cdot R_3 \quad (\text{Set } R_3 \leftarrow \tilde{Q}_3^H R_2 \bar{Q}_3) \\
&\stackrel{(b)}{=} G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot \hat{G}_5^{(s5)} \left(\hat{G}_4^{(s4)} G_5^{(f2)} \tilde{G}_4^{(3b)} \right) \cdot R_3 \quad (\text{Fusion}) \\
&\stackrel{(c)}{=} G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot \underline{\hat{G}_5^{(s5)} G_5^{(s6)}} G_4^{(s6)} \hat{G}_5^{(s6)} \cdot R_3 \quad (\text{Shift-through lemma}) \\
&\stackrel{(d)}{=} G_1^{(1)} G_2^{(s1)} G_3^{(s4)} \cdot \underbrace{G_5^{(f3)}}_{\bar{Q}_4} G_4^{(s6)} \underbrace{\hat{G}_5^{(s6)}}_{W_3} \cdot R_3 \quad (\text{Fusion}).
\end{aligned}$$

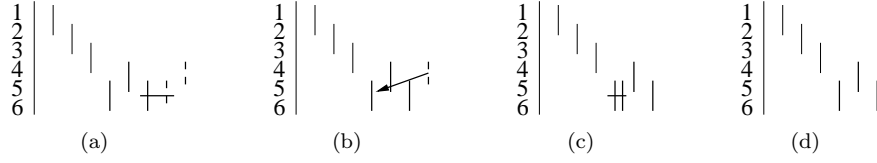


Figure 5: Graphical representation of what happens with the Q -factor: a) fusion, b) shift-through lemma, c) fusion, d) final result.

4. The fourth column is brought in Hessenberg form. Set $\bar{Q}_4 \leftarrow G_5^{(f4)}$ then

$$C_4 := \bar{Q}_4^H C_3 \bar{Q}_4 = G_1^{(1)} G_2^{(s1)} G_3^{(s4)} G_4^{(s6)} \cdot W_3 \cdot R_3 \bar{Q}_4.$$

The matrix $R_3 \bar{Q}_4$ is not upper triangular, a bulge appeared at position (6, 5):

$$\begin{array}{c}
G_1^{(1)} G_2^{(s1)} G_3^{(s4)} G_4^{(s6)} W_3 \quad R_3 \bar{Q}_4 \\
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6
\end{array} \left| \begin{array}{cccccc}
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & &
\end{array} \begin{array}{c}
\times \times \times \times \times \\
\times \times \times \times \times \\
\times \times \times \times \\
\times \times \times \\
\times \times \\
+ \times
\end{array}
\end{array}$$

The upper triangular form is restored by applying an extra Givens transformation $\tilde{G}_5^{(4a)}$ (set $\tilde{Q}_4 \leftarrow \tilde{G}_5^{(4a)}$). In fact, the matrix $\tilde{Q}_4^H R_3 \bar{Q}_4$ has to be upper triangular.

$$\begin{aligned}
C_4 &= G_1^{(1)} G_2^{(s1)} G_3^{(s4)} G_4^{(s6)} \cdot \underline{\hat{G}_5^{(s6)} \tilde{G}_5^{(4a)}} \cdot \tilde{Q}_4^H R_3 \bar{Q}_4 \\
&= G_1^{(1)} G_2^{(s1)} G_3^{(s4)} G_4^{(s6)} G_5^{(f4)} \cdot R_4 \quad (\text{Set } R_4 \leftarrow \tilde{Q}_4^H R_3 \bar{Q}_4 \text{ and fusion}).
\end{aligned}$$

Now the matrix C has been brought in Hessenberg form. Set (leave out the superscripts)

$$H = C_4 = G_1 G_2 G_3 G_4 G_5 \cdot R_4. \quad (7)$$

Remark 1: During the algorithm, due to rounding errors the unitary plus low rank structure of the R -factor can gradually be lost. In Section 3.6, a method is elaborated to restore this property.

3.4 Fast QR iteration: single shift case

3.4.1 Chasing principle

In this section, we describe an implementation of the following implicit single shift QR iteration on the Hessenberg matrix H , where $\alpha_k \in \mathbb{C}$,

$$\begin{aligned}
H_k - \alpha_k I &= Q_k R_k, \\
H_{k+1} &= R_k Q_k + \alpha_k I = Q_k^H H_k Q_k,
\end{aligned} \quad (8)$$

$k = 0, 1, \dots$ applied with $H_0 = H$. Choosing the appropriate shifts α_k , the matrix H_{k+1} converges to upper triangular form. Hence the algorithm can be used to determine the eigenvalues of a given matrix. It is also possible to adjust the proposed method to a method with double shift, the principles stay the same.

As in [4], each of the cycles $H_k \mapsto H_{k+1}$ in the QR iteration (8) will be implemented by means of a bulge chasing procedure. It suffices to describe the first transformation $H_0 \mapsto H_1$. We will explain this in detail for the previous 6×6 matrix with $p = 2$. At the beginning of the QR iteration, we have according to decomposition (7) (set $R_0 \leftarrow R_4$):

$$H_0 = G_1 G_2 \dots G_5 \cdot R_0.$$

1. Initiate bulge chasing: According to Wilkinson's criterion [13] the shift element α_1 is computed. This shift element determines the first Givens transformation $\bar{G}_1^{(w)}$ ((w) stands for Wilkinson). Let

$$\hat{H}_1 := (\bar{G}_1^{(w)})^H H_0 \bar{G}_1^{(w)} = \bar{G}_1 G_2 \dots G_5 \cdot R_0 \bar{G}_1^{(w)},$$

where we set $\bar{G}_1 \leftarrow (\bar{G}_1^{(w)})^H G_1$. We can write this factorization as follows:

$$\begin{array}{cc} \bar{G}_1 G_2 G_3 G_4 G_5 & R_0 \bar{G}_1^{(w)} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \left| \begin{array}{c} | \\ | \\ | \\ | \\ | \\ | \end{array} \right. & \begin{array}{c} \times \times \times \times \times \\ + \times \times \times \times \times \\ \times \times \times \times \times \\ \times \times \times \\ \times \times \\ \times \end{array} \end{array}$$

At position (2, 1) a bulge appears which we are going to remove by applying an extra Givens transformation \tilde{G}_1^H such that $\tilde{G}_1^H R_0 \bar{G}_1^{(w)}$ is upper triangular. How this Givens transformation \tilde{G}_1 is incorporated in the Q -factor, is first shown in formula form and then graphically in Figure 6.

$$\begin{aligned} \hat{H}_1 &= \bar{G}_1 G_2 G_3 G_4 G_5 \bar{\underline{G}}_1 \cdot \tilde{G}_1^H R_0 \bar{G}_1 \\ &\stackrel{(a)}{=} \left(\bar{G}_1 G_2 \bar{\underline{G}}_1 \right) G_3 G_4 G_5 \cdot R_1 \quad (\text{Set } R_1 \leftarrow \tilde{G}_1^H R_0 \bar{G}_1, \bar{\underline{G}}_1 \text{ pushed inward}) \\ &\stackrel{(b)}{=} G_2^{(s1)} \cdot G_1^{(s1)} \hat{G}_2^{(s1)} G_3 G_4 G_5 \cdot R_1 \quad (\text{Shift-through lemma}). \end{aligned} \tag{9}$$

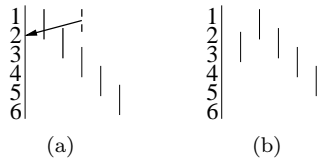


Figure 6: Graphical representation of what happens with the Q -factor: a) shift-through lemma, b) final result.

2. Second chasing: For the second chasing the left most Givens transformation $\bar{G}_2 \leftarrow G_2^{(s1)}$ of (9) is used. Let

$$\hat{H}_2 := \bar{G}_2^H \hat{H}_1 \bar{G}_2 = G_1^{(s1)} \hat{G}_2^{(s1)} G_3 G_4 G_5 \cdot R_1 \bar{G}_2.$$

Explicitly computing $R_1 \bar{G}_2$ results in:

$$\begin{array}{cc}
G_1^{(s1)} \hat{G}_2^{(s1)} G_3 G_4 G_5 & R_1 \bar{G}_2 \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \left| \begin{array}{c} | \\ | \\ | \\ | \\ | \\ | \end{array} \right. & \begin{array}{c} \times \times \times \times \times \\ \times \times \times \times \times \\ + \times \times \times \times \\ \times \times \times \times \\ \times \times \\ \times \end{array}
\end{array}$$

The bulge has been chased from position (2, 1) to (3, 2). An extra Givens transformation \tilde{G}_2^H is applied to make $\tilde{G}_2^H R_1 \bar{G}_2$ upper triangular. How \tilde{G}_2 is incorporated in the Q -factor is first shown in formula form and then graphically in Figure 7.

$$\begin{aligned}
\hat{H}_2 &= G_1^{(s1)} \hat{G}_2^{(s1)} G_3 G_4 G_5 \tilde{G}_2 \cdot \tilde{G}_2^H R_1 \bar{G}_2 \\
&= G_1^{(s1)} \left(\hat{G}_2^{(s1)} G_3 \tilde{G}_2 \right) G_4 G_5 \cdot R_2 \quad (\text{Set } R_2 \leftarrow \tilde{G}_2^H R_1 \bar{G}_2, \tilde{G}_2 \text{ pushed inward}) \\
&\stackrel{(a)}{=} G_1^{(s1)} \underbrace{G_3^{(s2)}}_{\leftarrow} G_2^{(s2)} \hat{G}_3^{(s2)} G_4 G_5 \cdot R_2 \quad (\text{Shift-through lemma}) \\
&\stackrel{(b)}{=} G_3^{(s2)} \cdot G_1^{(s1)} G_2^{(s2)} \hat{G}_3^{(s2)} G_4 G_5 \cdot R_2 \quad (G_3^{(s2)} \text{ pushed inward}).
\end{aligned}$$

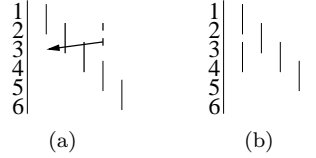


Figure 7: Graphical representation of what happens with the Q -factor: a) shift-through lemma, b) final result.

3. Third and fourth chasing: The same principle as with the second chasing is used. The bulge is chased from position (3, 2) to (4, 3) and from position (4, 3) to (5, 4).
4. Final step of bulge chasing: For the final chasing the left most Givens transformation is used, $\bar{G}_5^H \leftarrow (G_5^{(s4)})^H$. Let

$$\hat{H}_5 := \bar{G}_5^H \hat{H}_4 \bar{G}_5 = G_1^{(s1)} G_2^{(s2)} G_3^{(s3)} G_4^{(s4)} \hat{G}_5^{(s4)} \cdot R_4 \bar{G}_5.$$

The bulge at position (6, 5) of $R_4 \bar{G}_5$ has to be chased away:

$$\begin{aligned}
\hat{H}_5 &= G_1^{(s1)} G_2^{(s2)} G_3^{(s3)} G_4^{(s4)} \underline{\hat{G}_5^{(s4)} \tilde{G}_5} \cdot \tilde{G}_5^H R_4 \bar{G}_5 \\
&= G_1^{(s1)} G_2^{(s2)} G_3^{(s3)} G_4^{(s4)} G_5^{(f1)} \cdot R_5 \quad (\text{Set } R_5 \leftarrow \tilde{G}_5^H R_4 \bar{G}_5, \text{ and fusion}).
\end{aligned}$$

Let $H_1 = \hat{H}_5$. A cycle of the QR -iteration with single shift is completed.

Remark 2: During the algorithm, due to rounding errors the unitary plus low rank structure of the R -factor can gradually be lost. In Section 3.6, a method is elaborated to restore this property.

3.4.2 Deflation

An important concept in the practical implementation of the QR iteration method is deflation. This concept consists in setting small subdiagonal entries of the Hessenberg matrix equal to zero. In this way, the matrix is divided into two or more different submatrices whose eigenvalues can be computed separately. Different deflation criteria exist. In our implementation, the following

deflation criterion is used, $|h_{i,i-1}| < 1e - 16$. Because H is not explicitly stored, the entry $h_{i,i-1}$ needs to be computed as follows

$$\begin{bmatrix} h_{i-1,i-1} & \times \\ h_{i,i-1} & h_{i,i} \end{bmatrix} = \begin{bmatrix} -s_{i-2} & \bar{c}_{i-2}c_{i-1} & \times \\ & -s_{i-1} & \bar{c}_{i-1}c_i \end{bmatrix} \begin{bmatrix} r_{i-2,i-1} & \times \\ r_{i-1,i-1} & r_{i-1,i} \\ & r_{i,i} \end{bmatrix} \quad (10)$$

so that $h_{i,i-1} = -s_{i-1}r_{i-1,i-1} = (\bar{G}_{i-1})_{i,i-1}r_{i-1,i-1}$.

In our method, it is not possible to divide the matrix, which is in Givens-weight representation, into two disjoint Givens-weight representations and then compute the eigenvalues of both submatrices separately. Instead, the full representation is used and only the indices of the subblocks are stored. For instance, in the beginning the full matrix is considered: $\begin{bmatrix} 1 & N \end{bmatrix}$, when the first deflation has occurred ($|h_{i,i-1}| < 1e - 16$ for certain i), the block is split and we keep track of the indices of the subblocks $\begin{bmatrix} 1 & i-1 \\ i & N \end{bmatrix}$. The algorithm is then applied from the beginning to the ending index of each of the subblocks which can be handled separately.

3.5 Updating the Givens-weight representation of the R -factor

In Subsections 3.3-3.4, similarity transformations are applied. In fact, in every step of the method the R -factor represented in full form is multiplied with a Givens transformation \bar{G} at the right which created bulges. Because the R -factor is in fact in its compact Givens-weight representation, this representation has to be updated under this action of the transformation \bar{G} . Different techniques exist to achieve this update. Practically these techniques are important because they can be used in several matrix algorithms, e.g. QR factorization, reduction to Hessenberg form, and QR algorithm. We will consider the "generalized regression process" and the "concatenation process". The description below will be very brief; for more information about the applied updating techniques the reader is referred to [6, Section 7].

We will explain how to update the Givens-weight representation under the multiplication with a Givens transformation G_k to the right. This multiplication will influence the structure block which spans the columns $k+1, \dots, N$ (assuming that there is such a structure block). First a generalized regression process is applied, this corresponds to the fact that we remove the bottommost row from the structure block but the rank remains intact. Secondly, a concatenation process is applied, meaning that the Givens transformation is concatenated to the Givens-weight representation and also that the Givens transformation is applied to the elements outside the structure blocks. The concatenation corresponds to the structure block being extended with one column and with rank increased by one.

The process is explained in Figure 8. Figure 8(a) gives the situation where the R -factor is represented in its column-based Givens-weight representation and one Givens transformation was already added. Assume that a Givens transformation \bar{G} (dashed arrow) has to be applied to columns 2, 3 of the matrix R as in step 2 of the single shift QR -iteration algorithm. In particular we want to know how the submatrix $R(2 : 3, 2 : 3)$ changes under this operation, because this is where the bulge on the subdiagonal will appear. To get this submatrix $R(2 : 3, 2 : 3)$ in its real form, we must first 'regress' the action radius of the unitary component which is shown by the bold arrow acting on columns 3, 4, 5 of Figure 8(b). By this operation, the rank of the structure block remains intact, but it loses one row ($R(1 : 2, 3 : N) \rightarrow R(1, 3 : N)$): see Figure 8(c).

The next step, is to apply the new Givens transformation \bar{G} below the rank structure, in this case to the submatrix $R(2 : 3, 2 : 3)$, where it creates a bulge at the position $(3, 2)$. Of course we should also update the rank structured part; this is done by concatenating \bar{G} to the representation: see Figure 8(d). This process is called the concatenation process [6], and by this operation, the structure block gets an extra column and its rank increases by one.

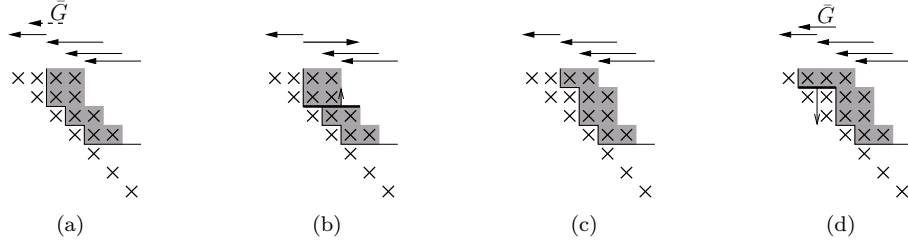


Figure 8: Updating the Givens-weight representation of the R -factor: (a) starting situation, applying \bar{G} is not possible yet because of the mixture of the weights and real elements, (b) regress the action radius of the bold unitary transformation, (c) remove one row of the rank structure, (d) apply the new Givens transformation \bar{G} to $R(2:3, 2:3)$ and concatenate it to the representation.

3.6 Restoring the compact representation of R

During the reduction to Hessenberg form and the QR -iteration method, the representation of the R -factor becomes more ‘heavy’ because of the accumulation of weights in the rank structure. This can be seen from Figure 8(c), when the action radius is regressed an extra element appears in the weight matrix at position (2, 5). This corresponds to an increase of the ranks of the structure blocks in the Givens-weight representation. In contrast, the mathematical ranks of the off-diagonal blocks of the R -factor: $R_{n-1} \leftarrow \bar{Q}_{n-1}^H R_{n-2} \bar{Q}_{n-1} = \bar{Q}^H R_0 \bar{Q}$, do not increase starting from $n = 2$. The reason is that R_{n-1} can be expressed as a rank p modification to a unitary matrix

$$R_{n-1} = \hat{Q}^H C_{n-1} = \hat{Q}^H \bar{Q}^H C_0 \bar{Q} = \hat{Q}^H \bar{Q}^H (Q + \tilde{U}V^H) \bar{Q} = \hat{Q}^H \bar{Q}^H Q \bar{Q} + (\hat{Q}^H \bar{Q}^H \tilde{U})(\bar{Q}^H V)^H,$$

where $C_{n-1} = \bar{Q}^H C_0 \bar{Q}$, $C_0 = Q + \tilde{U}V^H$, and $\bar{U} = \hat{Q}^H \bar{Q}^H \tilde{U}$ and $\bar{V} = \bar{Q}^H V$. Hence when the algorithm proceeds the ranks in the Givens-weight representation are an overestimation of the ‘true’ ranks. It is clear that this has a bad influence on the efficiency of the algorithm.

In this section an approach to restore the correct ranks in the Givens-weight representation is described. Because of the high computational cost of this restoration procedure, it should be applied not too intensively. In case of the reduction to Hessenberg form we found it convenient to apply it, every time that $2p$ columns have been brought in Hessenberg form and during the computation of the eigenvalues, it is applied, every p cycles of the QR -iteration. In this way the cost of the restoration procedure and the cost of the actual algorithm are more or less balanced.

The restoration procedure is based on the rationale proposed in [3]. The idea is to bring matrix R back to a unitary plus low rank form, $R = \tilde{Q} + \bar{U}\bar{V}^H$, with \tilde{Q} a unitary matrix, $\bar{U} \in \mathbb{C}^{m \times p}$ and $\bar{V} \in \mathbb{C}^{n \times p}$. This is done by computing the QR factorization of $R - \bar{U}\bar{V}^H = \tilde{Q}\tilde{S}$ [3] where \tilde{Q} is unitary and \tilde{S} is upper triangular with positive diagonal elements. Theoretically, we have $\tilde{Q} = \bar{Q}$ and $\tilde{S} = I$ as \bar{Q} is assumed to be unitary. Hence the matrix Δ defined by $\tilde{S} = I + \Delta$ is expected to have small norm. To avoid the accumulation of errors in the matrices Q and R which deteriorates their structural properties, the new iterate R is defined by $R = \tilde{Q} + \bar{U}\tilde{V}^H$ where $\tilde{V}^H = \bar{V}^H \tilde{S}^{-1}$. The elements of \tilde{V} are determined without explicitly computing the inverse of \tilde{S} , instead p linear systems are solved. Then the matrix R is unitary plus low rank whereas at the same time \tilde{Q} is numerically unitary. Practically this goes like follows.

- Compute the Givens-weight representation of $R - \bar{U}\bar{V}^H$. This is done separately for the lower and upper triangular part which are then simply combined. More details can be found in Section 3.6.1.
- Compute the QR -factorization of $R - \bar{U}\bar{V}^H$ by means of the algorithm in [9]: $R - \bar{U}\bar{V}^H = \tilde{Q}\tilde{S}$, ($\tilde{S} \approx I$).
- Compute the Givens-weight representation of \tilde{Q} .

- Compute the Givens-weight representation of the new $R := \tilde{Q} + \tilde{U}\tilde{V}^H$ with $\tilde{V}^H = \tilde{V}^H \tilde{S}^{-1}$. The matrix \tilde{V}^H can be obtained without explicitly computing matrix \tilde{S}^{-1} but by solving p linear systems by means of the algorithm in [9]. Notice that the matrix R has to be upper triangular.

Further details about the first step in the restoration procedure are given in Section 3.6.1. In Section 3.6.2 we describe two variants of the method.

3.6.1 Computing the Givens-weight representation for $R - UV^H$

The computation of a column based Givens-weight representation for $R - UV^H$ (we omit the bar sign on the matrices in this section) consists of two parts: computation of the lower part and the strictly upper part of the matrix. The computation of the lower part will not be discussed here because this part can be obtained directly from $-UV^H$ because R is upper triangular. The computation of the upper part is more involved since it must combine the Givens-weight representation of R with the $-UV^H$ term. The way to do this is explained by means of Figure 9.

Figure 9(a) shows the beginning of the method, at the left the matrix R in its Givens-weight representation and the rank p matrix $-UV^H$ ($p = 1$) are shown, and at the right the result of the combining process is presented. In general, the product $-UV^H$ is of rank p , but for simplicity we assumed that the rank is one in the example. The algorithm considers every row one by one, starting at the bottom of the matrix. So first, the elements outside the rank structure are computed. To compute these elements, three elements of $-UV^H$ have to be computed and added to the elements of R . In Figure 9(a), these elements for are denoted in a bold box. The result is shown at the right, the first weights of the Givens-weight representation of $R - UV^H$ are shown. Notice that the rank structure blocks of $R - UV^H$ will be the same as the rank structure blocks of the R -factor.

The next step is the computation of the weights and the unitary transformations of the Givens-weight representation of $R - UV^H$. We will start with the outer most block of the rank structure and use the unitary transformations of the R -factor also in the Givens-weight representation of $R - UV^H$. That is why in the first step the unitary transformation Q_3 has to be applied to the corresponding elements in V^H : $V^H(4 : 6) := V^H(4 : 6)Q_3$. This is shown in Figure 9(b) with an arrow above V^H . Then to compute the weights, the elements of both terms have to be added. To compute the elements of $-UV^H$, multiply the corresponding elements in V^H with the elements in $-U$ on the corresponding row: $-U(3)V^H(4 : 6)$. This is then added to the weights of R : $W(3, 4 : 6) + (-U(3)V^H(4 : 6))$, the result is shown at the right of Figure 9(b).

To compute the weights and unitary transformations of the second block of the rank structure, again the corresponding unitary transformation Q_2 is applied to V^H : $V^H(3 : 4) := V^H(3 : 4)Q_2$. Compress the two last columns of the vector V^H by a unitary transformation \tilde{Q}_2 , otherwise the representation of $R - UV^H$ will not be compact: $V^H(5 : 6) := V^H(5 : 6)\tilde{Q}_2$. Now the new weights of the representation can be computed: $W(2, 3 : 5) + (-U(2)V^H(3 : 5))$. The result is shown at the right of Figure 9(c). The last step for the top row is in fact the same as the previous one, apply first the unitary transformation Q_1 to V^H , secondly compress the $V^H(4 : 5)$ by a unitary transformation \tilde{Q}_1 and then add both terms $W(1, 2 : 4) + (-U(1))V^H(2 : 4)$, the final result which is the Givens-weight representation for the upper part of $R - UV^H$ is shown in Figure 9(d).

3.6.2 Two variants of the restoration procedure

In addition to the above method, we also implemented two variants of the restoration procedure. The first variant ignores the unitary plus low rank structure but only approximates the ranks in the Givens-weight representation by their numerical values, as in [1]. In our case this compression is done by using an algorithm in [7], approximating a compressed matrix. This method approximates a matrix with available Givens-weight representation, by one with lower ranks in its underlying rank structure. The method is the same as the swapping procedure [7], but with an additional compression of the weight matrix to its actual rank.

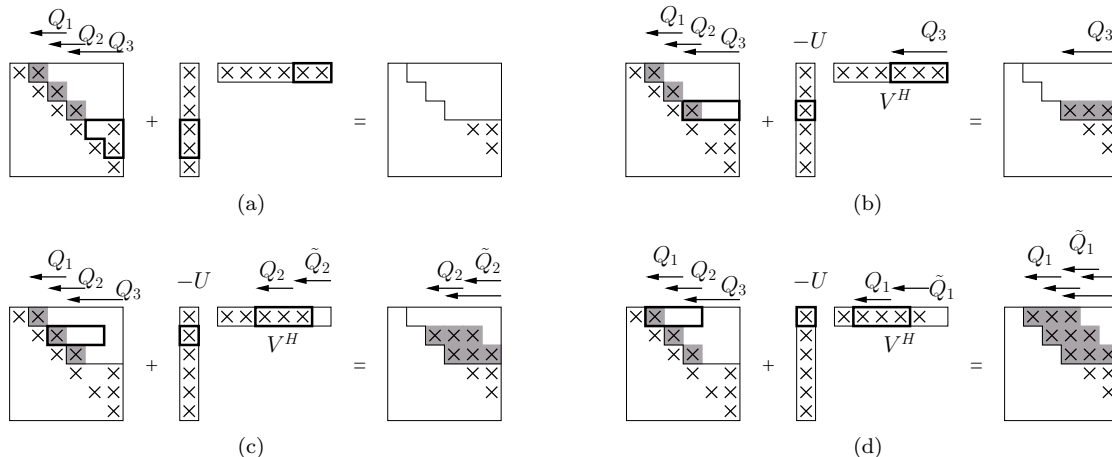


Figure 9: Updating the Givens-weight representation under a low rank update: (a) compute the elements outside the rank structure, (b) apply the unitary transformation Q_3 to V^H and compute the weights of the third row of $R - UV^H$, (c) apply the unitary transformation Q_2 to V^H and compress the two last columns of V^H by a unitary transformation \tilde{Q}_2 and compute the weights of the second row of $R - UV^H$, (d) same process as in step (c) for row one, at the right the final GWR of $R - UV^H$.

A drawback of this method is that the matrix does not necessarily stay unitary plus low rank. It only guarantees that the low rank structure of the matrix is restored.

The second variant combines parts of the above methods. The first two steps are the same as in the original restoration method. The third step, the computation of the Givens-weight representation of the new matrix R , is different. This time it is based on matrix \tilde{S}^{-1} and not on the matrix \tilde{Q} as in the original method. First the inverse \tilde{S}^{-1} is explicitly computed, then the Givens-weight representation of the product $R\tilde{S}^{-1}$ is obtained. Finally, the ranks of the obtained Givens-weight representation are approximated by their numerical values. This is done again by using the algorithm in [7]. Practically this goes like follows.

- Compute the Givens-weight representation of the new $R \leftarrow R\tilde{S}^{-1}$.
 - Compute the Givens-weight representation of the inverse matrix \tilde{S}^{-1} .
 - Compute the Givens-weight representation of the matrix product $R\tilde{S}^{-1}$, by intermingling the Givens transformations of both matrices.
 - Compress the representation by approximating the ranks by their numerical value, see the first variant.

Our numerical experiments indicate that the alternative restoration methods described in Section 3.6.2, are either less accurate (first variant) or substantially slower (second variant) than our original restoration procedure.

3.7 Balancing strategy

Because of the possible variation in magnitude of coefficients of the polynomial, normally a pre-processing step is applied to the matrix [4]. In fact this is a balancing strategy, meaning that the matrix is balanced such that the new matrix has a smaller norm. Generally DCD^{-1} is computed with D a diagonal matrix. To obtain again a unitary plus low rank matrix, the diagonal elements

have to be chosen as follows $d_i = \beta^i$ (for some β). This results in

$$DCD^{-1} = \beta \begin{bmatrix} 0 & 0 & \dots & 0 & -\frac{a_0}{\beta^n} \\ 1 & 0 & \dots & 0 & -\frac{a_1}{\beta^{n-1}} \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ddots & \ddots & 0 & -\frac{a_{n-2}}{\beta^2} \\ 0 & 0 & \dots & 1 & -\frac{a_{n-1}}{\beta} \end{bmatrix} = \beta \hat{C}.$$

For an appropriate β the norm of the matrix \hat{C} is much better. The eigenvalues of matrix C are then obtained by computing the eigenvalues of matrix \hat{C} and then multiplying them by the factor β .

3.8 Extension of the method

The method explained above is used to compute the eigenvalues of the corresponding system $Cx = \lambda x$. But it is also possible to apply the same techniques to the generalized eigenvalue problem $Cx = \lambda Bx$, which corresponds to non-monic matrix polynomial eigenvalue problems. Then the polynomials are of the following form $p(x) = \sum_{i=0}^n A_i x^i$ with $A_i \in \mathbb{C}^{p \times p}$, ($i = 0, \dots, n$). Then C is the block companion matrix as represented in (1) and B has the following form

$$B = \begin{bmatrix} I_p & 0 & \dots & 0 \\ 0 & I_p & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A_n \end{bmatrix}.$$

3.9 Overview of the method

Input: The coefficients $A_0, A_1, \dots, A_{n-1} \in \mathbb{C}^{p \times p}$.

Output: Eigenvalues of the blocks companion matrix $C \in \mathbb{C}^{pn \times pn}$.

1. Initialization

- QR factorization of C : $C = QR = Q_1 Q_2 \dots Q_{N-1} \cdot R$ with Q_j defined in Section 3.2.
- Compute U, V such that $R = Q^H C = I + UV^H$.
- Compute Givens-weight representation of R .

2. When $p > 1$ bring matrix C into Hessenberg form (Hessenberg reduction)

Set $C_0 = C$, $R_0 = R$, $Z_0 = Q = G_p^{(1)} \dots G_1^{(1)} Q_2 \dots Q_{N-1}$.

- for $j = 1, \dots, N - 2$
 - Set \bar{Q}_j as the product of the first $\min(p + j - 2, N - j - 1)$ Givens transformations in Z_{j-1} .
 - Apply similarity transformations: $Z_j \leftarrow \bar{Q}_j^H Z_{j-1}$ and $R_j \leftarrow R_{j-1} \bar{Q}_j$.
 - Find \tilde{Q}_j such that $\tilde{Q}_j^H R_j$ is upper triangular (eliminate the bulges by applying Givens transformations on the left).
 - Update $R_j \leftarrow \tilde{Q}_j^H R_j$.
 - Update $Z_j \leftarrow Z_j \tilde{Q}_j$ and rearrange by means of the shift-through lemma, see Section 3.3.
 - If $j \bmod 2p = 0$, restore the unitary plus rank p form of the R form.

end

Result: $Z_{N-2} \leftarrow G_1 \dots G_{N-1}$ and $C_{N-2} \leftarrow Z_{N-2} R_{N-2}$.

3. QR iteration with single shift

- Bulge chasing:
 - Set $H_0 = C_{N-2}$, $R_0 = R_{N-2}$, $Q_0 = Z_{N-2}$.
 - Determine shift to use (Wilkinson), this will give the first Givens transformation \bar{G}_1 .
 - for $j = 1, \dots, N-1$
 - * Apply similarity transformations: $Q_j \leftarrow \bar{G}_j^H Q_{j-1} = G_1 \dots G_{N-1}$ and $R_j \leftarrow R_{j-1} \bar{G}_j$
 - * Find \tilde{G}_j such that $\tilde{G}_j^H R_j$ is upper triangular (eliminate the bulge).
 - * Update $R_j \leftarrow \tilde{G}_j^H R_j$
 - * Update $Q_j \leftarrow G_1 \dots G_{N-1} \tilde{G}_j = \bar{G}_{j+1} \cdot G_1 \dots G_{N-1}$ by shift-through lemma, see Section 3.4.
 - end
- Test for deflation.
- Occasionally, restore the unitary plus rank p form of the R factor.
- Repeat the above steps until the matrix is completely deflated into blocks of sufficiently small size, whose eigenvalues can be computed directly.

As one can see, the two parts of the algorithm, the Hessenberg reduction and the QR iteration consist of the same kind of steps. The difference is that in the Hessenberg reduction not one bulge is chased but several bulges. That is also the reason why in the Hessenberg reduction the unitary transformations to remove and generate the bulges are denoted with \tilde{Q} and \bar{Q} , respectively. These unitary transformations are products of several Givens transformations.

4 Numerical experiments

To check the accuracy and the numerical stability of the algorithm, several numerical experiments are performed in Matlab[†]. The software to perform these numerical experiments can be requested from the authors.

Two different error measurements are considered in the three experiments. The maximum eigenvalue error (Max Eig Err) which is defined as the distance between the sets $\lambda(C)$ and $\tilde{\lambda}(C)$ by:

$$\text{dist}(\lambda(C), \tilde{\lambda}(C)) := \max\left\{ \max_{\tilde{\lambda} \in \tilde{\lambda}(C)} \|\tilde{\lambda} - \lambda(C)\|, \max_{\lambda \in \lambda(C)} \|\lambda - \tilde{\lambda}(C)\| \right\},$$

where $\|\lambda - \tilde{\lambda}(C)\| = \min_{\tilde{\lambda} \in \tilde{\lambda}(C)} |\lambda - \tilde{\lambda}|$, $\tilde{\lambda}(C)$ are the eigenvalues computed with our algorithm and $\lambda(C)$ the eigenvalues computed by the Matlab function `eig` (assume that these eigenvalues are exact). The maximal componentwise relative error in the coefficients of the polynomials ($p = 1$) (Max Coef Err) is defined as :

$$\max \left(\frac{|A_i - \tilde{A}_i|}{|A_i|} \right),$$

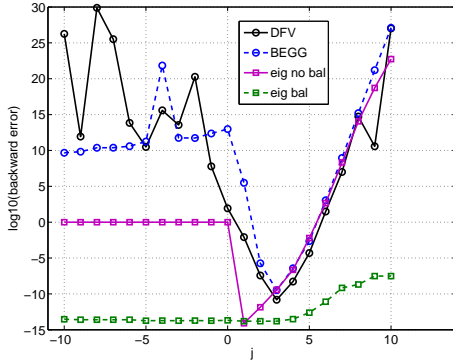
where \tilde{A}_i, A_i are the coefficients of the polynomial satisfied by the computed and exact eigenvalues, respectively.

Experiment 1 ($p = 1$): We consider the following degree 20 monic polynomials [12, 4]:

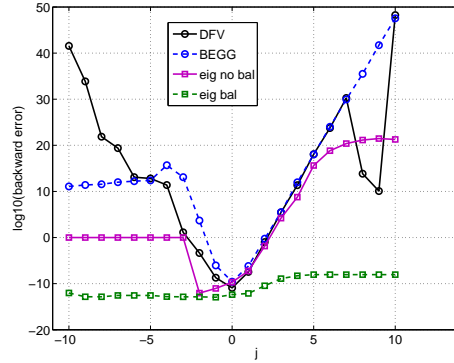
1. Wilkinson polynomial with roots: k with $k = 1, \dots, n$.
2. Monic polynomial with roots: $[-2.1 : 0.2 : 1.7]$.

[†]Matlab is a registered trademark of The MathWorks, Inc.

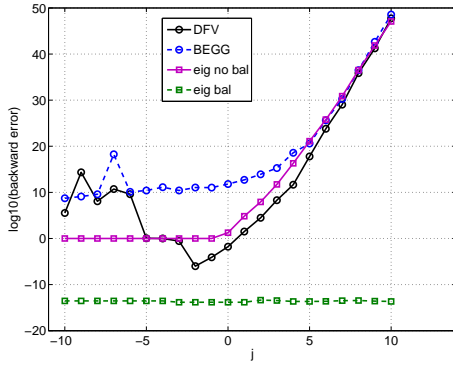
t



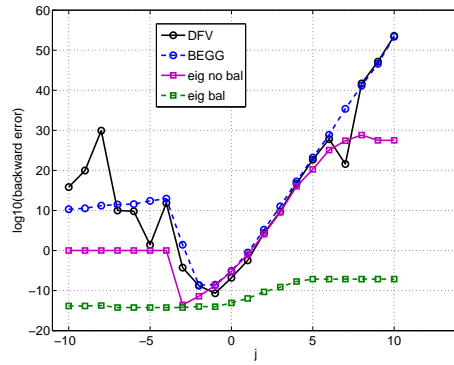
(a) Polynomial 1



(b) Polynomial 2



(c) Polynomial 3



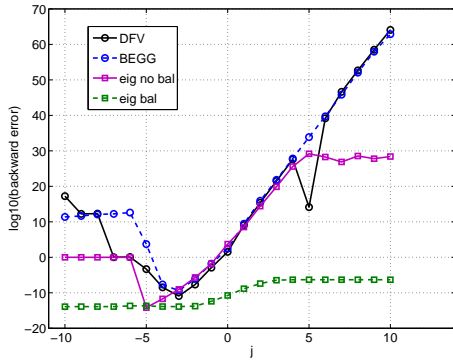
(d) Polynomial 4

Figure 10: Experiment 1: Looking for the optimal balancing factor $\beta = 2^j$ for polynomials 1 – 4.

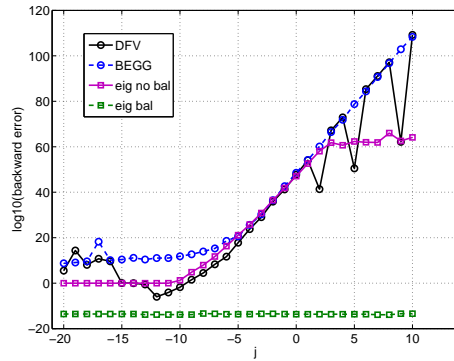
3. Monic polynomial with roots: 2^k with $k = -10, \dots, 9$.
4. Scaled Wilkinson polynomial with roots: $\frac{k}{n}$ with $k = 1, \dots, n$.
5. Reversed Wilkinson polynomial with roots: $\frac{1}{k}$ with $k = 1, \dots, n$.
6. Polynomial with well separated roots: $\frac{1}{2^k}$ with $k = 1, \dots, n$.
7. Polynomial $p(z) = (20!) \sum_{k=0}^{20} \frac{z^k}{k!}$.
8. Polynomial $p(z) = z^{20} + z^{19} + \dots + z + 1$.

For the polynomials 1) – 6) the roots are known, for 7) – 8) this is not the case. Because of the wide range of the coefficients of the polynomial, e.g. for the Wilkinson polynomial $|A_i| \in [1, 10^{20}]$, balancing, which is discussed in Section 3.7, is of high importance. Therefore different balancing factors β are applied to the matrix C . Figure 10-11, contains the result of varying the balancing factor $\beta = 2^j$, $j = -10, \dots, 10$ for the eight polynomials of degree 20. The result of the maximal componentwise error on the coefficients is shown in logarithmic scale for four different methods: DFV corresponds to the method proposed in this paper, BEGG denotes the method from [3], and eig no bal and eig bal correspond to the eig function of Matlab without and with balancing, respectively (with balancing is meant an extra balancing inside the function eig).

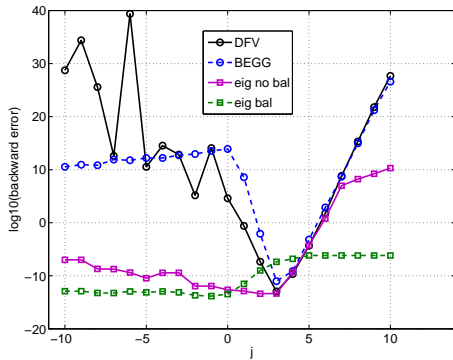
t



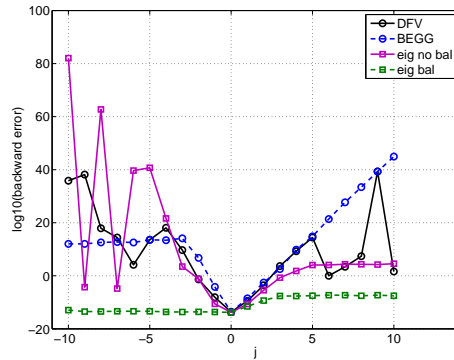
(a) Polynomial 5



(b) Polynomial 6



(c) Polynomial 7



(d) Polynomial 8

Figure 11: Experiment 1: Looking for the optimal balancing factor $\beta = 2^j$ for polynomials 5 – 8.

As one can see in the figures, only for some specific scaling factors β , an accurate solution is obtained. The function `eig bal` of Matlab always works. Note that this is an $\mathcal{O}(N^3)$ method as well as `eig` with no balancing. For polynomial 3) and 6) the proposed method DFV obtains a more accurate solution compared to, the other methods (BEGG - `eig no bal`) which work poorly. This is because the range of the coefficients is still large. Table 1 shows the range of the original coefficients and the scaled coefficients in case of DFV and BEGG for the optimal choice of β , this choice can be different for the two methods.

In Table 2–3, the componentwise error on the coefficients is shown for the optimal balancing factor β for the method of DFV and BEGG, respectively. Also the maximal componentwise error on the coefficients is shown. If one compares the two tables, the proposed method DFV obtains a slightly better accuracy. In the case of polynomials 3) and 6), the difference between the two

Index	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Range c_i	10^{19}	10^5	10^{17}	10^{10}	10^{19}	10^{63}	10^{18}	10^0
Range \hat{c}_i DFV	10^6	10^5	10^{20}	10^6	10^6	10^{20}	10^3	10^0
Range \hat{c}_i BEGG	10^6	10^5	10^{17}	10^8	10^6	10^{57}	10^3	10^0

Table 1: Experiment 1: Range of coefficients for optimal β .

Index	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1	10^{-14}	10^{-13}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-15}
2	10^{-11}	10^{-11}	10^{-6}	10^{-11}	10^{-11}	10^{-6}	10^{-13}	10^{-15}
3	10^{-12}	10^{-12}	10^{-6}	10^{-12}	10^{-12}	10^{-6}	10^{-14}	10^{-14}
4	10^{-12}	10^{-12}	10^{-6}	10^{-12}	10^{-13}	10^{-6}	10^{-14}	10^{-14}
5	10^{-12}	10^{-13}	10^{-6}	10^{-13}	10^{-13}	10^{-6}	10^{-14}	10^{-14}
6	10^{-12}	10^{-13}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
7	10^{-12}	10^{-14}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
8	10^{-12}	10^{-14}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
9	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
10	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
11	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
12	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
13	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
14	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
15	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-15}
16	10^{-12}	10^{-14}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
17	10^{-12}	10^{-12}	10^{-6}	10^{-12}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
18	10^{-12}	10^{-14}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
19	10^{-12}	10^{-14}	10^{-6}	10^{-13}	10^{-14}	10^{-6}	10^{-14}	10^{-14}
20	10^{-12}	10^{-14}	10^{-7}	10^{-13}	10^{-15}	10^{-7}	10^{-14}	10^{-14}
Max Coef Err	10^{-11}	10^{-11}	10^{-6}	10^{-11}	10^{-11}	10^{-6}	10^{-13}	10^{-14}
β	2^3	1	2^{-2}	2^{-1}	2^{-3}	2^{-12}	2^3	1

Table 2: Experiment 1: Numerical results of the proposed algorithm DFV.

methods is remarkable.

Experiment 2 ($p = 1$): To check that the computational complexity is of order $\mathcal{O}(N^2)$, we constructed random polynomials of degree n , by computing the coefficients $A_l \in \mathbb{C}$, $l = 0, \dots, n$ by using randomly distributed complex numbers. The matrix sizes were chosen as $N = pn = 25 \cdot 2^{i-1}$, with $i = 1, \dots, 7$. We assume that no balancing ($\beta = 1$) is necessary. For each of these sizes N , 5 samples were considered.

Figure 12(a) shows the time ratio T_{i+1}/T_i (T_i is duration in seconds for $i = 1, \dots, 7$) averaged over the 5 samples for the method DFV and BEGG. As one can see this is of the order $\mathcal{O}(N^2)$. Figure 12(b) shows the average number of QR iterations per cycle. For the method DFV the average is between 2 and 3. For the method BEGG it is a little bit more, between 5 and 6. This difference can be attributed to the deflation criterion.

In Table 4, the maximal componentwise error on the coefficients (Max Coef Err), the maximal eigenvalue error (Max Eig Err) and the percentage of time the algorithm takes to restore the unitarity of the matrix (Perc Rest) are shown for the method DFV and the first two for the method of BEGG. For the errors it can be seen that if N increases the errors increase slightly. Looking at the maximal eigenvalue error the results obtained with method DFV are slightly better. From the column Perc Rest for the method DFV it can be seen that around 81% of the time, the algorithm spend to restore the unitary plus low rank form of the matrix R .

Experiment 3 ($p > 1$): To test the algorithm for matrix polynomials, we constructed random polynomials of degree n , by computing the coefficients $A_l \in \mathbb{C}^{p \times p}$, $l = 0, \dots, n$ by using randomly distributed complex numbers. The matrix sizes were chosen as $N = pn = 25 \cdot 2^{i-1}$, with $i = 2, \dots, 6$. Again no balancing ($\beta = 1$) is assumed. For each of these sizes N , 5 samples were considered. Different values of the size p of the coefficient matrices are considered: $p = 2, 5, 10$.

Figure 13(a) shows the ratio T_{i+1}/T_i averaged over the 5 samples for the different p -values. It

Index	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
1	10^{-10}	10^{-10}	10^0	10^{-11}	10^{-10}	10^0	10^{-11}	10^{-14}
2	10^{-11}	10^{-10}	10^0	10^{-11}	10^{-11}	10^0	10^{-12}	10^{-14}
3	10^{-13}	10^{-12}	10^0	10^{-11}	10^{-13}	10^0	10^{-13}	10^{-14}
4	10^{-12}	10^{-12}	10^0	10^{-11}	10^{-13}	10^0	10^{-13}	10^{-14}
5	10^{-13}	10^{-13}	10^0	10^{-11}	10^{-13}	10^0	10^{-13}	10^{-14}
6	10^{-13}	10^{-13}	10^0	10^{-11}	10^{-13}	10^0	10^{-13}	10^{-14}
7	10^{-13}	10^{-13}	10^0	10^{-11}	10^{-13}	10^0	10^{-14}	10^{-14}
8	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-13}	10^0	10^{-14}	10^{-14}
9	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-13}	10^0	10^{-14}	10^{-14}
10	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-14}	10^0	10^{-13}	10^{-14}
11	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-14}	10^0	10^{-13}	10^{-14}
12	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-14}	10^0	10^{-13}	10^{-14}
13	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-14}	10^0	10^{-14}	10^{-14}
14	10^{-13}	10^{-14}	10^0	10^{-12}	10^{-14}	10^0	10^{-13}	10^{-14}
15	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-13}	10^0	10^{-13}	10^{-14}
16	10^{-13}	10^{-13}	10^0	10^{-12}	10^{-13}	10^0	10^{-13}	10^{-14}
17	10^{-13}	10^{-11}	10^0	10^{-11}	10^{-13}	10^0	10^{-13}	10^{-14}
18	10^{-13}	10^{-13}	10^0	10^{-11}	10^{-14}	10^0	10^{-12}	10^{-14}
19	10^{-13}	10^{-12}	10^4	10^{-10}	10^{-13}	10^4	10^{-12}	10^{-15}
20	10^{-13}	10^{-12}	10^9	10^{-9}	10^{-12}	10^9	10^{-11}	10^{-14}
Max Coef Err	10^{-10}	10^{-10}	10^9	10^{-9}	10^{-11}	10^9	10^{-11}	10^{-14}
β	2^3	1	2^{-1}	2^{-2}	2^{-3}	2^{-20}	2^3	1

Table 3: Experiment 1: Numerical results of method BEGG.

N	DFV			BEGG	
	Max Coef Err	Max Eig Err	Perc Rest	Max Coef Err	Max Eig Err
25	2.141×10^{-14}	4.276×10^{-15}	81%	3.287×10^{-14}	1.226×10^{-14}
50	4.4378×10^{-14}	4.650×10^{-15}	81%	1.494×10^{-13}	2.150×10^{-14}
100	1.734×10^{-13}	1.106×10^{-14}	81%	6.401×10^{-13}	7.639×10^{-14}
200	5.689×10^{-13}	3.774×10^{-14}	81%	3.570×10^{-12}	3.551×10^{-13}
400	3.636×10^{-12}	1.468×10^{-13}	81%	2.016×10^{-11}	9.245×10^{-12}
800	2.970×10^{-11}	4.262×10^{-13}	81%	1.134×10^{-10}	4.367×10^{-11}
1600	2.022×10^{-10}	3.329×10^{-12}	81%	5.605×10^{-10}	3.565×10^{-10}

Table 4: Experiment 2: numerical results for method DFV and BEGG.

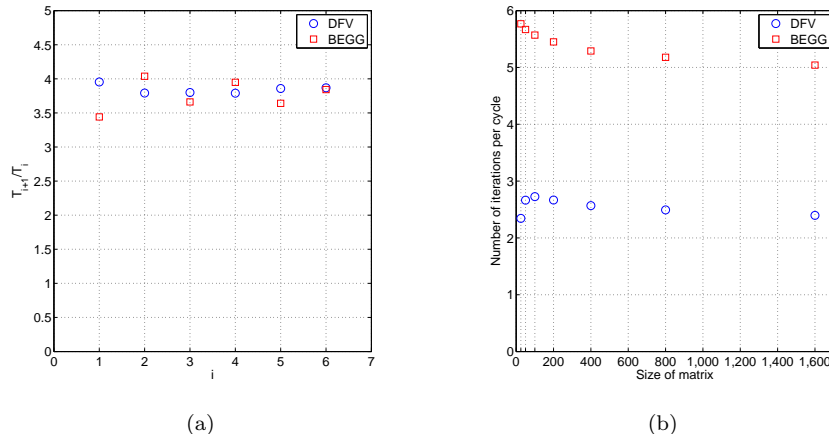


Figure 12: Experiment 2: a) time ratio, b) average of the number of iterations per cycle.

N	$p = 2$		$p = 5$		$p = 10$	
	Max Eig Err	Perc. Rest.	Max Eig Err	Perc. Rest.	Max Eig Err	Perc. Rest.
50	1.511×10^{-14}	65%	2.276×10^{-14}	55%	3.950×10^{-14}	45%
100	5.734×10^{-14}	67%	6.654×10^{-14}	58%	8.472×10^{-14}	51%
200	1.698×10^{-14}	67%	1.724×10^{-13}	59%	2.451×10^{-13}	54%
400	6.698×10^{-13}	67%	5.597×10^{-13}	59%	6.580×10^{-14}	65%
800	1.769×10^{-13}	67%	1.688×10^{-12}	67%	1.587×10^{-12}	55%

Table 5: Experiment 3: numerical results for method DFV, $p = 2$, $p = 5$ and $p = 10$.

shows that the algorithm is $\mathcal{O}(N^2)$. Figure 13(b) shows the average number of QR iterations per cycle. For all the three values of $p = 2, 5, 10$ the average is around 2.5.

In Tables 5, the same values as in Table 4 are shown, leaving out the maximal componentwise error on the coefficients (Max Coef Err). The same observation as for $p = 1$ can be stated: the maximum eigenvalue error increases slightly as N increases. The time spent to restore the companion is decreased if p increases, but this is probably because for higher values of p , the computation of the Hessenberg matrices takes more time.

5 Conclusion

In this paper an algorithm to compute the eigenvalues of a monic matrix polynomial is described. This is done by computing the eigenvalues of the corresponding block companion matrix. We showed how the QR algorithm can be adjusted such that the algorithm exploits the rank structure of the matrices involved and how the matrices can be represented in Givens-weight representation. The numerical performance of the algorithm was demonstrated by means of some numerical experiments showing the good accuracy and efficiency of the algorithm.

References

- [1] D. Bindel, S. Chandrasekaran, J. W. Demmel, D. Garmire, and M. Gu. A fast and stable nonsymmetric eigensolver for certain structured matrices. Technical report, Department of Computer Science, University of California, Berkeley, California, USA, May 2005.

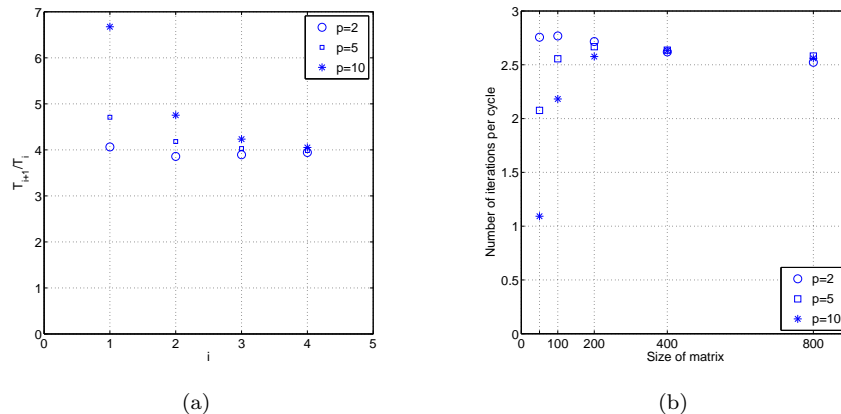


Figure 13: Experiment 3: a) time ratio, b) average of the number of iterations per cycle.

- [2] D. A. Bini, F. Daddi, and L. Gemignani. On the shifted QR iteration applied to companion matrices. *Electronic Transactions on Numerical Analysis*, 18:137–152, 2004.
- [3] D. A. Bini, Y. Eidelman, L. Gemignani, and I. C. Gohberg. Fast QR eigenvalue algorithms for Hessenberg matrices which are rank-one perturbations of unitary matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(2):566–585, 2007.
- [4] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu. A fast QR algorithm for companion matrices. *Operator Theory: Advances and Applications*, 179:111–143, 2007.
- [5] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numerische Mathematik*, 36:177–195, 1981.
- [6] S. Delvaux. *Rank Structured Matrices*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3000 Leuven (Heverlee), Belgium, June 2007.
- [7] S. Delvaux and M. Van Barel. A Givens-weight representation for rank structured matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1147–1170, 2007.
- [8] S. Delvaux and M. Van Barel. Eigenvalue computation for unitary rank structured matrices. *Journal of Computational and Applied Mathematics*, 213(1):268–287, March 2008.
- [9] S. Delvaux and M. Van Barel. A QR -based solver for rank structured matrices. *SIAM Journal on Matrix Analysis and Applications*, 30(2):464–490, 2008.
- [10] I. S. Dhillon. *A new $O(n^2)$ algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*. PhD thesis, Dept. of Computer Science, University of California, Berkeley, 1989.
- [11] J. J. Dongarra and D. C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM Journal on Scientific and Statistical Computation*, 3(2):139–154, March 1987.
- [12] A. Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64(210):763–776, April 1995.
- [13] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, USA, third edition, 1996.
- [14] B. N. Parlett. *The Symmetric Eigenvalue Problem*, volume 20 of *Classics in Applied Mathematics*. SIAM, Philadelphia, Pennsylvania, USA, 1998.

- [15] B. N. Parlett and I. S. Dhillon. Relatively robust representations of symmetric tridiagonals. *Linear Algebra and its Applications*, 309(1–3):121–151, 2000. Proceedings of the international workshop on accurate solution of eigenvalue problems University Park, PA , 1998.
- [16] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, USA, 1999.