

The explicit QR-algorithm for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW 459, May 2006



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

The explicit QR-algorithm for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW459, May 2006

Department of Computer Science, K.U.Leuven

Abstract

In this paper we show how to perform in an explicit way the shifted QR-algorithm for computing the eigenvalues of a rank structured matrix. The implementation is based on the underlying preservation of rank structure under the QR-algorithm. It will be expressed in terms of the Givens-weight/unitary-weight representation which we introduced in a previous paper. The results of some numerical experiments will be reported.

Keywords : rank structured matrix, Givens-weight/unitary-weight representation, shifted QR-algorithm, eigenvalue computation, structure inheritance.

AMS(MOS) Classification : Primary : 65F15, Secondary : 15A21, 15A03.

The explicit QR-algorithm for rank structured matrices

Steven Delvaux *, Marc Van Barel *

5th May 2006

Abstract

In this paper we show how to perform in an explicit way the shifted QR-algorithm for computing the eigenvalues of a rank structured matrix. The implementation is based on the underlying preservation of rank structure under the QR-algorithm. It will be expressed in terms of the Givens-weight/unitary-weight representation which we introduced in a previous paper. The results of some numerical experiments will be reported.

Keywords: rank structured matrix, Givens-weight/unitary-weight representation, shifted QR-algorithm, eigenvalue computation, structure inheritance.

AMS subject classifications: 65F15, 15A21, 15A03

1 Introduction

In this paper we explain how one can perform the shifted QR-iteration algorithm for eigenvalue computation on a rank structured matrix. The algorithm is *explicit* in the sense that it explicitly forms the QR-factorization of a shifted matrix $A - \lambda I$, $\lambda \in \mathbb{C}$. The implementation makes use of the Givens-weight representation [5], hereby providing a counterpart to the algorithm for symmetric, scalar quasiseparable representations handled in [8].

*Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium. email: {Steven.Delvaux,Marc.VanBarel}@cs.kuleuven.ac.be.

The research was partially supported by the Research Council K.U.Leuven, project OT/05/40 (Large rank structured matrix computations), Center of Excellence: Optimization in Engineering, by the Fund for Scientific Research-Flanders (Belgium), G.0455.0 (RHPH: Riemann-Hilbert problems, random matrices and Padé-Hermite approximation), G.0423.05 (RAM: Rational modelling: optimal conditioning and stable algorithms), and by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture, project IUAP V-22 (Dynamical Systems and Control: Computation, Identification & Modelling). The scientific responsibility rests with the authors.

A matrix will be called *rank structured* if the ranks of certain submatrices starting from its bottom left corner, as well as the ranks of certain submatrices starting from its top right corner, are small compared to the matrix size.

The determination of the eigenvalue spectrum of a rank structured matrix by means of the QR-algorithm has been investigated in several places in the literature. Let us give here a brief overview.

It is a classical result in numerical linear algebra that, when applying the QR-algorithm on an (unreduced) Hessenberg matrix, the resulting matrix is again of Hessenberg type. Based on this property, one is led to the classical $O(n)$ algorithm for applying a QR-step on a Hermitian, tridiagonal matrix, where n is the matrix size: see [10, p. 417].

In [1], some more general shapes of lower banded matrices are studied which are invariant under the QR-algorithm.

In [2], the preservation of a suitable rank structure is used to devise an $O(n)$ implementation for applying a QR-step on a Frobenius (i.e., companion) matrix.

In [3], an $O(n)$ implementation is devised for applying a QR-step for certain Hermitian-plus-rank-one rank structured matrices of semiseparability rank one and with low rank blocks situated just below the main diagonal. This structure includes the so-called arrowhead matrices, which just as Frobenius matrices are a useful class for polynomial root location.

In [12], an $O(n)$ implementation is described for applying a QR-step on a Hermitian semiseparable matrix, using the so-called Givens-vector representation. By using a preliminary similarity transformation into semiseparable form, the QR-algorithm can be used here to compute the eigenvalue decomposition of an arbitrary Hermitian matrix, in much the same way as for the classical tridiagonal case: see [11].

All the above examples deal with the preservation of rank structure under the QR-algorithm. The case of a general rank structure was handled in [4, 7] where the structure preservation and corresponding sparsity pattern of Givens transformations during the QR-factorization were analysed.

Let us mention that in theory, there can be considered a certain diagonal correction term to the rank structure, which is also preserved by the QR-algorithm. This was observed for semiseparable plus diagonal matrices in [9] and later in a more general context in [4, 7]. But an implementation of these theoretical results is beyond the scope of the present paper. In fact, we will restrict ourselves here to the case where the rank structure is situated *strictly away from the main diagonal*.

In this paper, we will exploit the above mentioned preservation results for rank structures under the QR-algorithm, using the Givens-weight representation. This will lead to an $O((r + s)^2 n)$ complexity algorithm for applying a QR-step on a Hermitian rank structured matrix, where n is the matrix size, r is some measure for the average rank index of the rank structure, and s is some measure for the bandwidth of the unstructured matrix part around the main diagonal. Hence the *total* complexity of the QR-algorithm will be of order $O((r + s)^2 n^2)$.

This paper can be seen as the counterpart of the results of [8], where an

explicit QR-algorithm was implemented using a quasiseparable representation. Let us point out that the latter paper assumes the matrix to be (i) Hermitian, and the corresponding rank structure to be (ii) situated just below the main diagonal, and (iii) dense, in the sense that the low rank blocks are following immediately one after the other. This last condition is awkward since the corresponding, ‘scalar’ quasiseparable representations suffer from an overload of parameters in case of a moderate or high rank index r [5]. Although we believe that the exposition of [8] should allow an analogue where the above conditions are removed, we note that such restrictions do not occur for the Givens-weight representation.

For further reference, let us recall here the two defining equations of the shifted QR-algorithm [10]. These equations show how to obtain from the matrix $A^{(\nu)} \in \mathbb{C}^{n \times n}$ a new iterate $A^{(\nu+1)}$:

$$A^{(\nu)} - \lambda I = QR \tag{1}$$

$$A^{(\nu+1)} = RQ + \lambda I, \tag{2}$$

where $\lambda \in \mathbb{C}$ is called the *shift*, Q is unitary and R upper triangular. As it is known, by appropriately choosing shifts the matrices $A^{(\nu)}$ converge to (block) upper triangular form, or to diagonal form in the Hermitian case, and hence the QR-algorithm can be used to determine the eigenvalues of a given matrix $A = A^{(0)}$.

The remainder of this paper is organized as follows. In Section 2 we recall the basic ideas of the Givens-weight representation. Section 3 recalls the algorithm for computing the QR-factorization of a rank structured matrix. Section 4 provides and recalls some results about the preservation of rank structure under the QR-algorithm. Section 5 describes the QR-algorithm in the Hermitian case. Section 6 deals with the QR-algorithm in case of arbitrary rank structure in the upper triangular part. Sections 7, 8 consider an alternative scheme for the Hermitian plus low rank case. Use is made here of some general tools for updating the Givens-weight representation under the influence of a low rank correction term, described in Section 7. Finally, some numerical experiments are reported in Section 9.

2 Givens-weight representation

In this section we review the basic ideas of the Givens-weight representation. This section follows the exposition in [5], except that Definition 4 did not appear yet in explicit form in the latter paper.

Definition 1 (See [4]:) *We define a pure rank structure \mathcal{R} on $\mathbb{C}^{m \times n}$ as a collection of so-called pure structure blocks $\mathcal{R} = \{\mathcal{B}_k\}_k$. Each pure structure block \mathcal{B}_k is characterized as a 3-tuple*

$$\mathcal{B}_k = (i_k, j_k, r_k),$$

where i_k is the row index, j_k the column index, r_k the rank upper bound. We say a matrix $A \in \mathbb{C}^{m \times n}$ to satisfy the pure rank structure \mathcal{R} if for each k ,

$$\text{Rank}A(i_k : m, 1 : j_k) \leq r_k.$$

The above definition uses the word *pure* to distinguish from the more general rank structures which were handled in [4]. Since these more general structures do not occur in the present paper, we will simplify notation by just dropping the word *pure* everywhere from the notation.

Note that by definition, all structure blocks have to start from the lower left matrix corner.

In practice, it often happens that also the block *upper* triangular part is rank structured, i.e., that also the matrix A^T satisfies rank structure in the sense of Definition 1. By abuse of notation, we will indiscriminately use the term *rank structure* also in this case.

We will assume in what follows that we are working with a rank structure \mathcal{R} for which there are no structure blocks that are ‘contained’ in each other, i.e., for which the structure blocks \mathcal{B}_k can be ordered such that both their row and column indices i_k and j_k increase in a strictly monotonic way.

Definition 2 (*Unitary-weight representation. See [5]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a rank structure $\mathcal{R} = \{\mathcal{B}_k\}_{k=1}^K$, where the structure blocks are ordered from top left to bottom right. A unitary-weight representation of the matrix A according to the structure \mathcal{R} consists of a pair $(\{U_k\}_{k=1}^K, W)$. Here the U_k , $k = K, \dots, 1$ form a sequence of unitary transformations proceeding from bottom to top of the matrix, as indicated by the fat arrows in Figure 1, serving to create zeros in the subsequent rank- r_k structure blocks \mathcal{B}_k except for their top r_k rows. On the other hand, the matrix $W \in \mathbb{C}^{m \times n}$ is called the weight matrix, and it contains the blocks of elements obtained at the top border of the rank structure at the moment just after applying U_k . See Figure 1.

The basic idea of this definition is to compress the given rank structured matrix by means of subsequent unitary transformations, hereby proceeding from bottom to top of the matrix, and storing each time the elements just before they reach the top border of the rank structure.

Note that this definition leads to an *internal* representation of the rank structure, in the sense that it involves no information about the matrix part lying outside the reach of the structure blocks. Moreover, it implies that each unitary operation U_k has a certain *action radius* in the sense that it acts only on a limited number of columns. This action radius is a monotonically decreasing function when the U_k proceed from bottom to top of the matrix.

If a unitary-weight representation of a matrix is given, we can restore the full matrix by *spreading out* the representation. This means that we gradually have to consider the subsequent weight blocks and multiply them with the ‘decompressing’ unitary operations U_k^{-1} , each one acting only on the columns on the left of its action radius. Note that this process proceeds from top to bottom

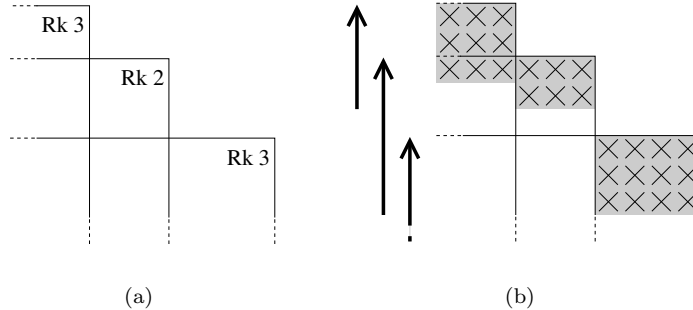


Figure 1: For the rank structure in the left picture, the right figure shows a schematic picture of the unitary-weight representation. The notation ‘Rk r ’ denotes that the structure block is of rank at most r .

of the matrix. At the end of this process we will retrieve the original, full matrix which we started from.

We can now specify from unitary-weight to Givens-weight representations. In what follows, we will use the term *Givens transformation* to denote a unitary operation which differs from the identity matrix only in two subsequent rows i and $i + 1$. This transformation will sometimes be denoted as $G_{i,i+1}$.

Rather than individual Givens transformations, it will be useful to work with *Givens arrows*: these are defined as collections of subsequent Givens transformations, each of them being situated precisely one position below the previous one: see Figure 2.

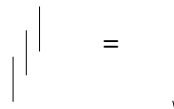


Figure 2: A Givens arrow $G_{i+2,i+3}G_{i+1,i+2}G_{i,i+1}$ consisting of 3 Givens transformations. Concerning this figure, we recall the reader that we consider each Givens transformation as ‘acting’ on the rows of an (invisible) matrix standing on the right of it, and hence that the Givens transformations in the figure should be evaluated from right to left, hereby explaining the downward direction of the Givens arrow.

The number of Givens transformations of which a Givens arrow consists will be called the *width* of the Givens arrow. Moreover, we define the *top* and the *tail indices* of the Givens arrow in an obvious way.

Definition 3 (*Givens-weight representation. See [5]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a rank structure $\mathcal{R} = \{\mathcal{B}_k\}$, where the structure blocks are ordered

from top left to bottom right. A Givens-weight representation of A according to the structure \mathcal{R} is a unitary-weight representation where additionally each unitary component U_k is decomposed into a product of Givens arrows, such that

- each of the Givens arrows has width at most r_k ,
- both the tops and the tails of the subsequent Givens arrows of each U_k are monotonically proceeding upwards. For the tails, we assume that this monotonicity is strict.

See Figure 3.

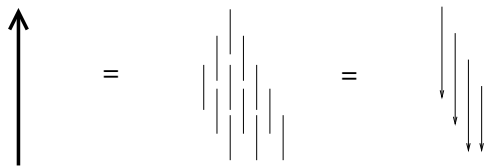


Figure 3: Suppose that the current structure block is $\text{Rk } 3$, and that the corresponding unitary transformation U_k spans over 6 rows. Then we assume for this unitary transformation a decomposition into a product of Givens arrows of width at most 3.

We should still explain why the assumption is made that each Givens arrow in the decomposition of U_k has width at most r_k . To this end, recall that the unitary transformation U_k serves to create zeros in a certain $\text{Rk}(r_k)$ submatrix, except for its top r_k rows. No matter if we do this by means of a singular value decomposition or by a pivoted QR-factorization or any other unitary operation, this effect can always be realized by a succession of Givens arrows as prescribed.

Note that by decomposing each unitary transformation U_k as specified in Definition 3, we formally obtain a decomposition into a product of *too many* Givens transformations, in the sense that the beginning and trailing Givens transformations of two subsequent unitary transformations U_k may overlap. Such an overlap can be avoided by assuming the representation to be *efficient* in the sense that it consists of approximately rn Givens transformations, for a practical choice of rank structure. Some general techniques for transforming a Givens-weight representation to efficient form have been described in [5]. However, the algorithms in this paper will work for any Givens-weight/unitary-weight representation.

We give an additional definition.

Definition 4 A unitary-weight representation is said to be interlaced if each U_k is given as a decomposition $U_k = \tilde{V}_k V_k$, where

- V_k is a unitary transformation situated on ‘top’ of U_k ,
- \tilde{V}_k is a unitary transformation situated on the ‘bottom’ of U_k ,

- the decompositions of the different U_k are consistent in the sense that the top row of \tilde{V}_k has index one more than the bottom row of V_{k-1} , for all k .

See Figure 4.

We note that interlaced unitary-weight representations arise in a natural way if one applies ‘rank-increasing’ operations to the unitary-weight representation, such as the concatenation and generalized swapping processes [5]. However, a note of caution is due to the fact that these representations are always unitary-weight, but in general not Givens-weight representations. Indeed, it often happens that both V_k and \tilde{V}_k have a decomposition into Givens arrows in the style of Figure 3, but in general this does not mean that their product $U_k = \tilde{V}_k V_k$ has such a decomposition too. Anyway, this observation will not harm us for the remainder of this paper.

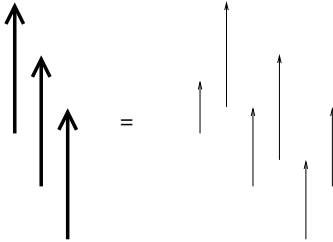


Figure 4: Schematic picture of an interlaced unitary-weight representation according to Definition 4. Note that each unitary component is given in the form $U_k = \tilde{V}_k V_k$, satisfying the restrictions in Definition 4.

3 QR-factorization

In this section we review the algorithm to compute the QR-factorization of a rank structured matrix. This section contains essentially a review of [6], but it is also important since it sets up the main example which will be used throughout the remainder of this paper (Figure 8).

For the remainder of this section, we will assume that there is given a *unitary-weight* representation for the given rank structured matrix A . (For the case where one is working with a *Givens-weight* representation, we will indicate the corresponding modifications at the appropriate places in the algorithm.)

We will interpret the QR-equation $A = QR$ as follows: the matrix Q^H will be considered as a product of unitary transformations acting on the matrix A , hereby transforming it to upper triangular form: $R = Q^H A$.

This process proceeds in two phases. The first phase is called the *preparative phase*. It is based on the observation from Section 2 that for a rank structured matrix A , a sequence of unitary operations can be applied to transform the given

$\text{Rk } r_k$ structure blocks of A into blocks of zeros, except for their top r_k rows. Note that the preparative phase proceeds from bottom to top of the matrix.

The second phase is called the *residual phase*. It serves to zero out the remaining non-zero elements in the strictly lower triangular part of the matrix resulting at the end of the preparative phase. This process is achieved by means of a sequence of upward pointing Givens arrows, one for each column. Note that the residual phase proceeds from top to bottom of the matrix.

For further use, let us recall the following definition.

Definition 5 (*Sparsity pattern. See [6]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a structure block $\mathcal{B} = (i, j, r)$. We say a QR-factorization $A = QR$ to satisfy the sparsity pattern induced by the structure block \mathcal{B} if Q^H can be written as a decomposition $Q^H = Q_3^H Q_2^H Q_1^H$, with

- Q_1^H acting on rows i, \dots, m (serving to transform \mathcal{B} into a block of zeros, except for its top r rows),
- Q_2^H acting on rows $1, \dots, i + r - 1$ (serving to create zeros above and in the top r rows of \mathcal{B}),
- Q_3^H acting on rows $j + 1, \dots, m$ (serving to create zeros on the right of \mathcal{B}).

See Figure 5.

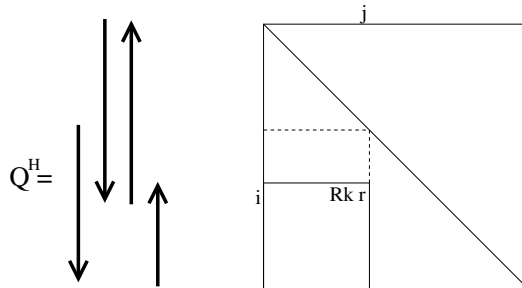


Figure 5: For the structure block shown on the right, the left picture shows the corresponding sparsity pattern of the matrix Q^H . The picture shows a decomposition into preparative (upward) and residual (downward) unitary transformations.

We should stress that the above definition was formulated from the point of view of a *single* structure block. In practical situations, there will probably be more than one structure block, causing each of the unitary components Q_i^H , $i = 1, 2, 3$ to have an additional decomposition into a sparse product of unitary transformations. To stress this point, we indicated in Figure 5 the connection with the preparative and residual unitary transformations.

Now we recall the inheritance of structure by the R-factor of the QR-decomposition of A . We will restrict here to the case where A has a structure block in its lower, and one in its upper triangular part, which are lying on the same horizontal line. Then the R-factor inherits structure as shown in Figure 6.

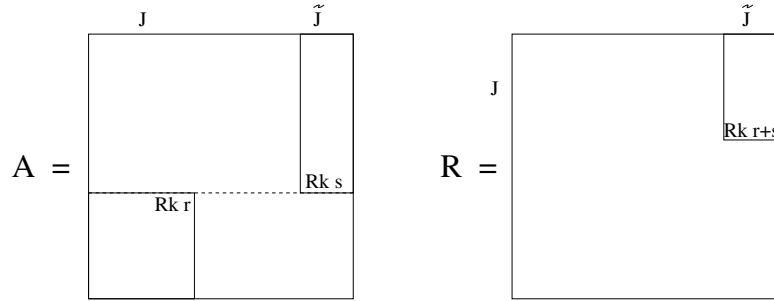


Figure 6: Inheritance of structure by the R-factor.

Let us now exploit these preservation results and compute the QR-factorization of A in a practical way. We start from the rank structure which is shown in Figure 7.

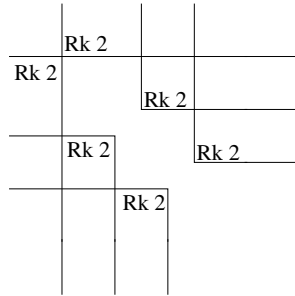


Figure 7: Starting rank structure.

The corresponding, starting unitary-weight representation is shown in Figure 8(a). Note that this figure shows a unitary-weight representation for the structured lower triangular, as well as for the structured upper triangular part. The meaning of the latter representation can be obtained in a straightforward way by ‘transposing’ the corresponding notions of Section 2.

Let us comment on the remaining part of Figure 8. The algorithm for computing the QR-factorization starts by applying the *preparative* unitary transformations. Since these unitary transformations have already been precomputed in the unitary-weight representation, this will lead to what we call a *peeling-off process* in terms of the unitary-weight representation for the lower triangular

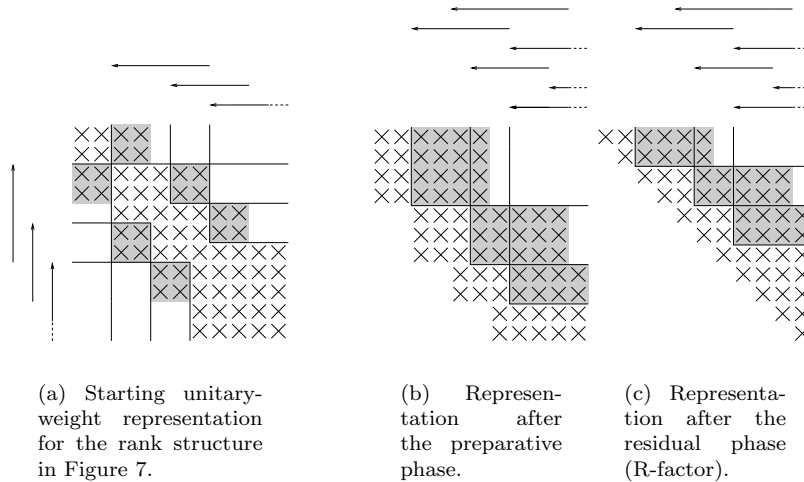


Figure 8: The figure shows the unitary-weight representation in the beginning and at the end of the preparative and residual phases for the rank structure in Figure 7.

part. However, in the meantime, the upper triangular part of the matrix should be updated too. This is achieved by a *generalized swapping process* as described in [5]. In practice, it results in the fact that one has to compute auxiliary unitary transformations \tilde{V}_k to bring the ‘disturbing’ elements, which are brought into the upper triangular part by the upward movement of the preparative phase, as much as possible to the left, hereby avoiding a fill-in of the unitary-weight representation of the upper triangular part during the preparative phase. The original unitary column transformations V_k are then replaced by $V_k \tilde{V}_k$, i.e., they are interlaced with the auxiliary column transformations \tilde{V}_k . (See Definition 4, where the order of the factors V_k, \tilde{V}_k must be reversed now since we are working with a column-based instead of a row-based unitary-weight representation.) In terms of the underlying rank structure, this reflects the fact that the ranks in the upper triangular part increase, and that the corresponding structure blocks ‘grow’ with some extra rows. The situation at the end of the preparative phase is shown in Figure 8(b).

Next the *residual* unitary transformations are applied, i.e., the remaining elements below the main diagonal are cancelled. This is achieved by means of a *generalized regression process* as described in [5]. This process reflects the fact that the structure blocks in the upper triangular part ‘shrink’ by a few rows. The situation at the end of the residual phase is shown in Figure 8(c).

This ends our brief description of the QR-factorization algorithm. More details can be found in [6]. Moreover, some of the described techniques will be

encountered again in Section 5, where we will provide a more detailed description of the implementation.

Remark 6 1. We point out that some of the structure blocks in the upper triangular part of Figure 8(b) have grown with three extra rows w.r.t. Figure 8(a), and with rank increased by three. One could decrease this value from three to two by using the more fine-tuned Givens-weight version of the generalized swapping process, as we did in [6], rather than the unitary-weight versions used here.

2. Actually, the QR-algorithm requires the QR-factorization of the shifted matrix $A - \lambda I$ rather than the original rank structured matrix A . But this can be done by just redefining $A := A - \lambda I$. Also the corresponding unitary-weight representation can be easily updated in this way, by our assumption that the structure blocks do not involve the main diagonal.

4 Preservation of rank structure under the QR-algorithm

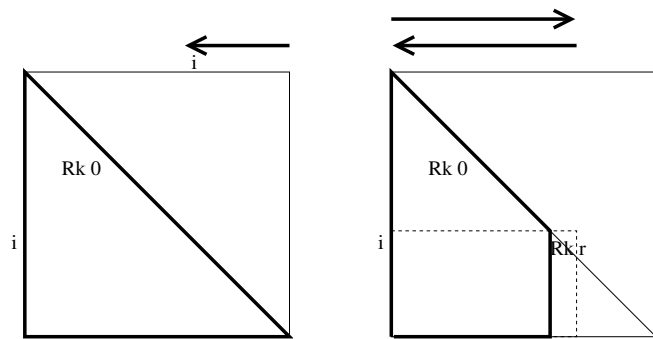
In this section we provide and recall some facts concerning the preservation of rank structure under the QR-algorithm. We will deal with the following result.

Theorem 7 Rank structure is preserved under the QR-algorithm, i.e., suppose that $A \in \mathbb{C}^{n \times n}$ is a matrix satisfying some structure block $\mathcal{B} = (i, j, r)$, situated strictly below the main diagonal. Then by computing the QR-factorization $A = QR$ according to the sparsity pattern of Definition 5, the new QR-iterate RQ also satisfies the structure block \mathcal{B} .

PROOF. This result is a consequence of [4, 7], where more general preservation results were established. Indeed: we mention that the pattern of the Givens transformations in the latter paper was more ‘canonically’ determined than in the current paper, but this fact was not essential for the ‘continuity arguments’ in [7] to hold.

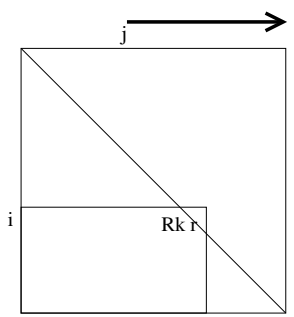
However, we want to proceed here by an alternative proof, which allows for algorithmic exploitation. We note that the QR-algorithm applies to the columns of the R-factor the unitary operation Q to obtain the new QR-iterate RQ . This process can then be split up in two phases, which we call the *preparative-columns* and the *residual-columns* phases, defined as the Hermitian transposes of the corresponding phases of Section 3. The restoration of the original rank structure is then shown to be an easy consequence of the sparsity pattern of the matrix Q^H : see Figure 9.

Let us explain, e.g., why the dashed structure block in Figure 9(b) has rank at most r . This follows since the first and the second component of the sparsity pattern in Definition 5 overlap each other precisely in r rows, which are the top r rows of the given Rk r structure block. The other details in the figure can be

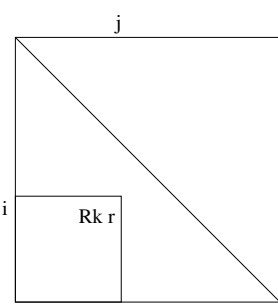


(a) Apply the first preparative-columns unitary operations to the matrix R .

(b) Apply the final preparative-columns and the first residual-columns unitary operations.



(c) Apply the final residual-columns unitary operations.



(d) We obtain now the new QR-iterate RQ .

Figure 9: The figure demonstrates the preservation of structure by the new QR-iterate by directly arguing in terms of the sparsity pattern of Definition 5.

checked similarly. □

Remark 8 (*Structure inheritance by the Q-factor:*) Figure 9 shows the preservation of structure by the new QR-iterate $A^{(\nu+1)} = RQ$ by a direct argument. Since this argument depends only on the fact that R is upper triangular, it can also be used for showing that the matrix $I_m Q = Q$ inherits the structure block \mathcal{B} , with I_m the identity matrix. Thus by working with the sparsity pattern of Definition 5, the Q-factor inherits all the structure blocks from the original matrix A : see also [4, 7].

We will provide now yet another proof for Theorem 7. In what follows, we will denote by $G_{i,\dots,j}$ a block Givens transformation, i.e., a unitary transformation behaving like the identity transformation except in a number of subsequent rows and columns i, \dots, j .

We need the following, trivial observation.

Lemma 9 *Given an upper triangular matrix R and a block Givens transformation $G_{i,\dots,j}$, there exists a refactorization*

$$RG_{i,\dots,j} = \tilde{G}_{i,\dots,j}\tilde{R}$$

where $\tilde{G}_{i,\dots,j}$ is again a block Givens transformation, and where \tilde{R} is again upper triangular: see Figure 10.

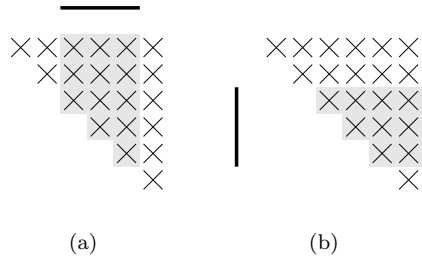


Figure 10: Propagating a block Givens transformation through a triangular shape. The block Givens transformation G acting on the columns creates a bulge in the lower triangular part of the matrix R , which can then be removed by means of a suitable block Givens transformation G^H acting on the rows.

One can interpret the above result by stating that the block Givens transformation G is really ‘propagated’ through the triangular shape of the matrix R , hereby transforming from a column into a row operation. Hence the above idea is intimately connected with the swapping process described in [5].

Now we apply this result to the current situation. Starting from the new QR-iterate RQ , we can repeatedly apply the above lemma to obtain a refactorization $\tilde{Q}\tilde{R}$ where \tilde{Q} consists of a succession of Givens transformations acting on exactly the same succession of rows as in the original matrix Q . Thus we see that the new QR-iterate allows a QR-factorization having exactly the same sparsity pattern for its Q-factor as the original matrix A . The preservation of structure could be established as a direct consequence of this.

Note that the above proof could be applied in *block* form as well, by taking as the block Givens transformations in Lemma 9, the subsequent unitary operations U_K^H, \dots, U_1^H of the Givens-weight representation. Some elements of this proof will be reflected also in the practical algorithm, to be described in the next section.

5 QR-algorithm for the Hermitian case

In this section we will describe the QR-algorithm, assuming that the given rank structured matrix A is Hermitian. We will assume that a QR -factorization $A = QR$ has been computed as explained in Section 3.

The completion of the QR-step can then be achieved by applying to the matrix R the *preparative-columns* and the *residual-columns* phases mentioned before. We recall that these two phases are the counterparts of the two phases of Section 3, with the difference that for each unitary transformation G occurring in these previous phases, we multiply now with the Hermitian transpose unitary transformation G^H to the columns.

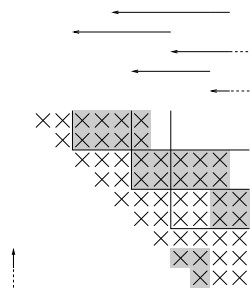
First we consider the preparative-columns phase. Since this phase contains precisely the Hermitian transposes of the preparative unitary transformations of Section 3, and since we assume the matrix A to be Hermitian, these are nothing but the unitary transformations in the unitary-weight representation for the *upper triangular* part of the original matrix A in Figure 8(a).

Moreover, we already remarked that these unitary transformations are still present in the matrix R obtained at the end of the preparative and residual phase, although they are now interlaced with some auxiliary column operations: see Figures 8(b), 8(c).

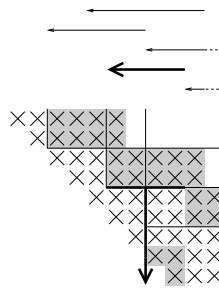
Hence we see that the preparative-columns transformations *are already occurring* in the unitary-weight representation of the matrix R . Therefore, the effect of these transformations has already been partially precomputed, and hence their application will merely consist in *peeling them off* from the representation for the upper triangular part, in much the same way as was done during the preparative phase in Section 3 (but now with the role of rows and columns reversed).

During this peeling-off process, the structured lower triangular part of the matrix should be updated too. This can be done again by a generalized swapping process [5].

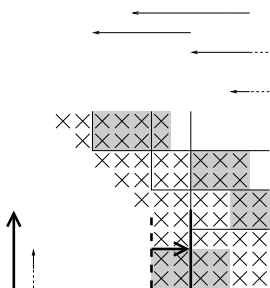
The preparative-columns phase is illustrated in Figure 11.



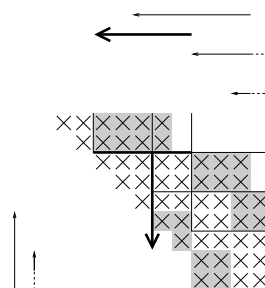
(a) Starting unitary-weight representation.



(b) Peel off the next unitary operation U_k^H . We apply it only on rows 5,6,..., since the result on rows 1,...,4 has already been pre-computed.



(c) Apply an auxiliary row operation.



(d) We could now go on to peel off the next unitary operation.

Figure 11: Preparative-columns phase

Let us comment on this figure. Figure 11(a) shows the starting unitary-weight representation. We assume here that the bottom right structure block in the upper triangular part has already ‘shrunk’ under the action of the earlier preparative-columns unitary operations. Moreover, we have already started to build up a row-based unitary-weight representation for the lower triangular part as well.

Figure 11(b) shows the peeling-off of the next unitary operation U_k^H . Since part of its application has already been precomputed, this operation has to be applied only to the rows below its action radius. Note that this will result in a small bulge below the main diagonal. To remove this bulge, we apply an auxiliary unitary row operation. We use this row operation to create zeros, only in those columns which will be influenced by the next unitary operations U_{k-1}^H, \dots, U_1^H : see Figure 11(c).

Note that the elements lying in the intersection of rows 3 and 4 and columns 6 and 7 have turned from grey into white. This corresponds to the fact that there are no unitary transformations of the unitary-weight representation anymore acting on these elements, and hence that they correspond now to real-size elements of the matrix. We could say that both the size and the rank of the structure block have ‘shrunk’ under the peeling-off process.

The peeling-off operations of the next unitary operations U_k^H are similar and hence are not shown anymore. The situation after application of the entire preparative-columns phase is shown in Figure 12.

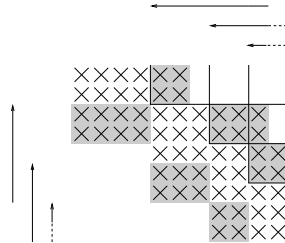


Figure 12: Unitary-weight representation after the complete preparative-columns phase.

Note that the above description of the preparative-columns phase is closely related to Lemma 9, although we were not concerned about keeping the weight matrix exactly upper triangular.

Remark 10 *The above process could be further fine-tuned by using the Givens-weight rather than the unitary-weight representation. It will then be appropriate to follow exactly Lemma 9, in the sense that for each Givens transformation creating a bulge in the lower triangular part of the weight matrix, we immediately restore the upper triangularity by means of an auxiliary row operation: see Figure 13.*

This *Givens-weight* version of the algorithm is also the one that we used in our implementation. It has the advantage that it can always be used in an efficient way, for any distribution of the structure blocks. On the other hand, the unitary-weight version of the algorithm described above is didactically easier to understand, and by the fact that it blocks the operations, it may beat the complexity of the *Givens-weight* algorithm by a constant factor, provided that the distribution of the structure blocks is well-chosen.

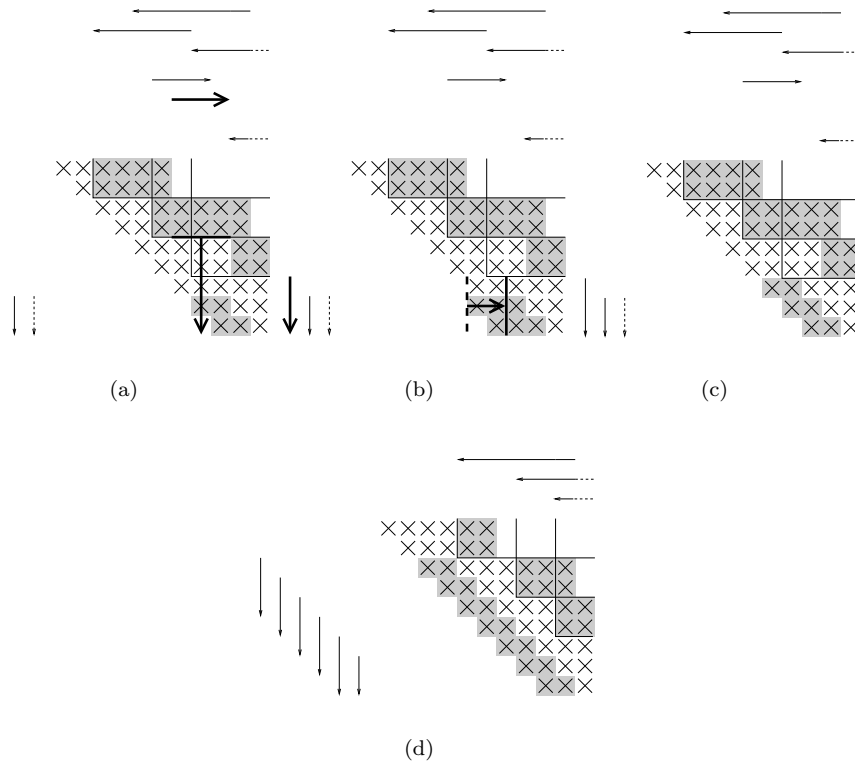


Figure 13: Preparative-columns phase using a *Givens-weight* rather than a unitary-weight representation. Parts (a-c) show a peeling-off operation of a Givens arrow from the column representation, and the corresponding auxiliary row operation. In these figures, only the relevant unitary operation U_k^H is explicitly shown as a decomposition of Givens arrows. Part (d) shows the final *Givens-weight* representation at the end of the complete preparative phase, and should be compared with Figure 12.

Now we turn to the final phase in the completion of the QR-step, namely the residual-columns phase. This phase is illustrated in Figure 14.

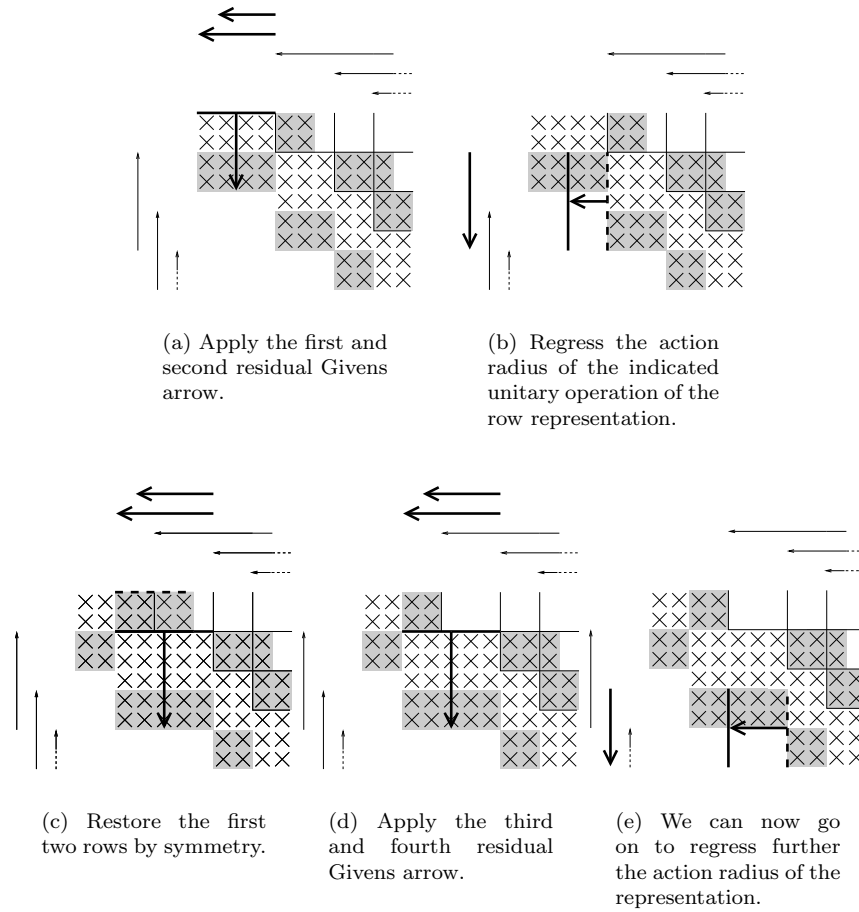


Figure 14: Residual-columns phase

Let us comment on this figure. Figure 14(a) starts by showing the application of the first and second residual Givens arrow to the columns.

We would then like to go on by applying also the next Givens arrows to the columns. But this would lead to a mixture of real-size elements and weights. To avoid this, we first regress the action radius of the required unitary component of the row representation: see Figure 14(b).

We could then go on by applying the next Givens arrows to the columns. But before doing this, we note that these Givens arrows do not involve the first two columns anymore. Therefore, this is now the right moment to exploit once more the symmetry condition under which we are working: we can obtain the elements of the first two rows from those of the first two columns. Figure 14(c) shows the corresponding update of the weight matrix. All the next operations will then be applied only on rows 3,4,..., hence not influencing these first two rows anymore.

Figure 14(d) shows the updated weight matrix, on which we can apply the third and fourth Givens arrows, hereby leaving the first two rows invariant, as just announced. The updated weight matrix is shown in Figure 14(e).

We can then go on to regress further the action of the representation, and to restore the third and fourth rows by symmetry; these operations are not shown anymore. The final situation at the end of the entire residual-columns phase is shown in Figure 15.

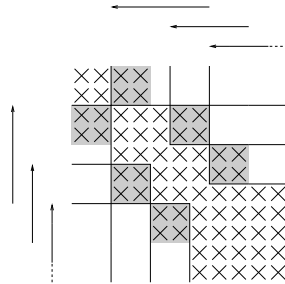


Figure 15: unitary-weight representation after the complete residual-columns phase. We have now applied a complete QR-iteration step, and the underlying rank structure is again equal to the one we started from.

Just as in the preparative-columns phase, the above process could be further fine-tuned by using the Givens-weight representation: See Figure 16.

Summarizing, we described now how to perform a QR-step on a Hermitian rank structured matrix A , which seems to be the main case of interest. In the remaining sections, we will consider some cases where the Hermitian condition is relaxed.

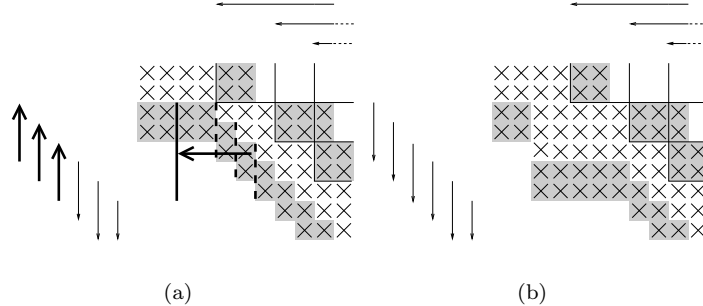


Figure 16: Residual-columns phase using a Givens-weight rather than a unitary-weight representation. The starting matrix is the one shown in Figure 13(d). Figure (a) shows then an operation of regressing the action radius, while part (b) shows the resulting situation. These two figures should be compared to Figures 14(b) and 14(c).

6 QR-algorithm for the case of arbitrary rank structure in the upper triangular part

Also in the case of *arbitrary* rank structure in the upper triangular part, one could implement the QR-algorithm, based on the general tools for updating the Givens-weight/unitary-weight representation under the influence of Givens transformations. But we should warn that in general, the ranks in the upper triangular part will soon start to increase, hereby heavily restricting the usefulness of the algorithm in this case.

An exception to this are the cases of *Hermitian plus low rank* or *unitary plus low rank* matrices. Indeed, in these cases, the rank structure in the upper triangular part is merely *induced* by the one in the lower triangular part, of which we know that it is preserved by virtue of Theorem 7. We suggest then the following, general procedure:

1. compute the QR-factorization in the same way as usual (Section 3);
2. perform a swapping procedure to *swap* the column-based Givens-weight representation of the upper triangular part into a row-based representation [5];
3. compute the new QR-iterate RQ by applying the preparative-columns and residual-columns phase, in much the same way as in Section 5. Since the upper triangular part is now represented by a row-based Givens-weight representation, it can be updated by the same techniques of generalized swapping and generalized regression processes encountered earlier in this paper (but now with the role of rows and columns reversed);

4. having completed the QR-step, the coordinates of the rank structure in the upper triangular part should be set back to their theoretically predicted position. This means that one should restore the correct *ranks*, as well as the correct *shapes* of these structure blocks, by means of a numerical approximation procedure [5]. (Possibly, one could first enlarge or regress the action radii of the Givens-weight representation to obtain the correct shapes of the structure blocks; but these operations could be intermingled with the approximation procedure itself.)

We stress that the above procedure requires a numerical approximation after each QR-step. Therefore, we believe that more appropriate techniques could be used to compute the eigenvalue spectrum in the Hermitian/unitary plus low rank case, to be described in our future work. Alternatively, an adapted algorithm for the Hermitian plus low rank case is described in the next two sections, to which we turn now.

7 Computing low rank updates of the Givens-weight representation

In preparation for the QR-algorithm for Hermitian plus low rank matrices of the next section, the current section considers the problem of incorporating a low rank correction term into the Givens-weight/unitary-weight representation.

In what follows, we prefer to focus on the *upper* triangular part, which we assume to be represented by a column-based unitary-weight representation (the way how to modify the algorithm for the *Givens*-weight representation will be treated further.) It will be assumed that the low rank correction term has rank at most p , so that it can be written as uv , with $u \in \mathbb{C}^{m \times p}$ and $v \in \mathbb{C}^{p \times n}$. The problem statement is shown in Figure 17.

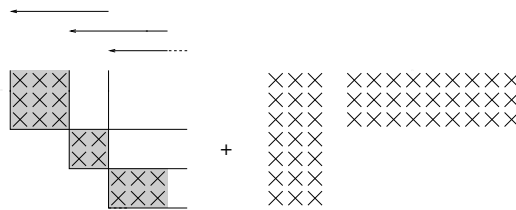
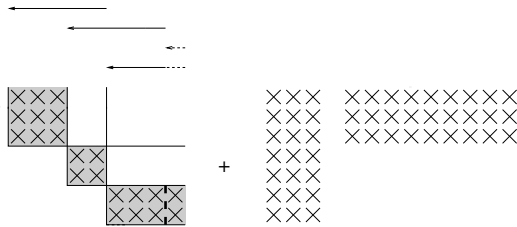
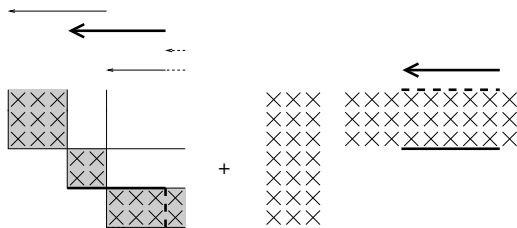


Figure 17: Given a unitary-weight representation as in the left part of the figure, the problem is to update the representation under the influence of the low rank correction term uv , having rank at most p . The crosses on the right of the figure denote the corresponding parts of the generators u, v . We have $p = 3$ in the figure.

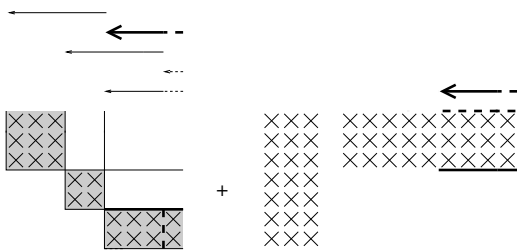
The algorithm to update the unitary-weight representation is shown in Figure 18.



(a) Starting situation. The figure shows the original rank structured matrix, which we started to compress. The weights on the right of the dashed line are due to the matrix uv .

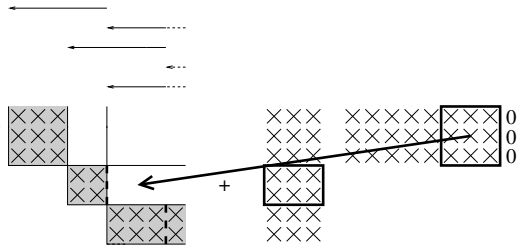


(b) Apply the next unitary column operation V_k to compress the given matrix (no practical computations have to be performed, since the effect of this operation has been completely precomputed). Apply this operation to the matrix uv as well.

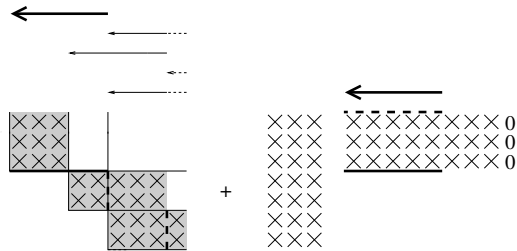


(c) Apply an auxiliary unitary column operation V_k to annihilate as many columns as possible in the matrix v . Concatenate this operation to the representation of the given matrix.

Figure 18: Updating the unitary-weight representation under the influence of a low rank correction term uv (a-c).



(d) Compute the weight block \bar{W}_k by multiplying the corresponding submatrices of u and v , and concatenate it to the representation.



(e) We could now go on to apply the next unitary column operations V_{k-1}, \dots, V_1 , hereby applying similar updating procedures as above.

Figure 18: Updating the unitary-weight representation under the influence of a low rank correction term uv (d-e).

Let us comment on this figure. Figure 18(a) shows the starting situation, where we started to perform a compression of the given rank structured matrix. Note that the unitary column operations V_k achieving this have already been precomputed, precisely by the concept of unitary-weight representation. The provenance of the extra weight block \tilde{W}_{k+1} (note the tilde), which is shown on the right of the dashed line on the bottom of the figure, will be explained further.

Now we apply the next unitary operation V_k of the unitary-weight representation, and we apply it to the low rank correction term uv as well: see Figure 18(b).

Having done this, it is now the right moment to apply an auxiliary unitary operation \tilde{V}_k to the columns (note the tilde), in order to annihilate as many columns as possible in the generator matrix v . This operation should involve only those columns of v which will not be influenced by the next unitary operations V_{k-1}, \dots, V_1 anymore: see Figure 18(c).

By the fact that we performed a compression of the matrix uv as well, we can extract from this matrix the current weight block \tilde{W}_k which is obtained in rows 4, 5 and columns 6, 7, 8 of this matrix. This extra weight block \tilde{W}_k can then be simply ‘concatenated’ on the right of the last computed weight block W_k of the original rank structured matrix: see Figure 18(d). The updated situation is then shown in Figure 18(e).

Continuing this process, at the end one will obtain the unitary-weight representation which is updated under the influence of the low rank correction. Note that this results in a unitary-weight representation having unitary operations updated in the form $V_k \tilde{V}_k$, and weight blocks updated in the form $[W_k \tilde{W}_k]$, i.e., both the original unitary operations and weight blocks are still present in the representation, but they are now interlaced with the auxiliary unitary operations and weight blocks arising from the compression of the low rank correction matrix uv . The final situation is shown in Figure 19.

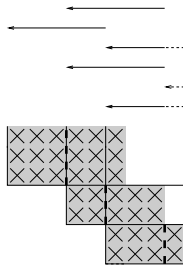


Figure 19: Updated unitary-weight representation for the matrix in Figure 17. Note that the original unitary-weight representation is now interlaced with some auxiliary compression operations, and that also additional weight blocks have been concatenated to the representation.

Note that we did not specify yet how to choose the auxiliary unitary opera-

tions \tilde{V}_k used to compress the matrix v . One could use products of Householder or Givens transformations to do this. The second possibility, e.g., amounts to working with Givens arrows of width at most p , where p is the rank of the low rank correction term: see Figure 20.

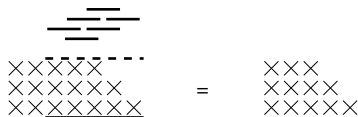


Figure 20: By working with individual Givens transformations to annihilate entries inside the matrix v , one can perform these operations in an efficient way.

By following the scheme of Figure 20, it follows that in total, not more than pn Givens transformations are involved, one for each annihilated element of v . Hence the computation of the auxiliary operations \tilde{V}_k and the corresponding weight blocks \tilde{W}_k both require a complexity of order $O(p^2n)$.

Similarly, it is easy to check that the application of the original unitary operations V_k to the matrix v requires a complexity of order $O(prn)$, assuming that the original unitary-weight representation was efficient in the sense that it contained approximately rn parameters, where r is a measure for its average semiseparability rank. The computation of the corresponding weight blocks W_k is costless since they are contained in the original unitary-weight representation, which we assume to be known.

We conclude that, under the assumptions specified above, the above algorithm has a complexity of order $O(p^2n + prn)$. Finally, note that this algorithm generalizes the process of building a Givens-weight/unitary-weight representation for a uv -representable matrix, which was described in [5].

8 QR-algorithm for the Hermitian plus low rank case

In this section we consider the QR-algorithm in case where A is a rank structured matrix satisfying

$$A - A^H = \text{Rk } p, \tag{3}$$

where $\text{Rk } p$ is a matrix of rank at most p , $p \geq 0$. Note that such an equation will be satisfied in particular when A is Hermitian plus some low rank correction. Moreover, this equation is clearly preserved under the QR-algorithm.

First we have to analyze at which places the Hermitian assumption was used in the algorithm of Section 5, so that we can see what will go wrong if this condition is relaxed to the form (3). We note that the Hermitian assumption was certainly *not* used for maintaining the lower triangular part: indeed, the preservation result of Theorem 7 does not need any symmetry at all. Also the

techniques for updating the Givens-weight representation for the lower triangular part of the new QR-iterate $A^{(\nu+1)}$, as described in Section 5, could be taken over without difficulty.

The place where the Hermitian condition *was* needed, was merely for an efficient treatment of the *upper* triangular part during the algorithm. It was used, e.g., in the residual-columns phase, where the restoration of symmetry at the end of the QR-algorithm allowed to forget about the upper triangular part. It was used also in the preparative-columns phase, where the symmetry allowed to peel off the unitary column operations from the upper triangular part, as in Figure 11(b). As noted in Section 5, the validity of this peeling-off goes back to the very beginning of the algorithm, in Figure 8(a), where the Hermitian condition guaranteed that the unitary transformations of the lower triangular part have to appear in the upper triangular part as well (in transposed form).

We want now to establish an analogue of this last property for the Hermitian plus low rank case. To this end, we claim that the representation can be chosen in such a way that the unitary operations of the lower triangular part *still* appear in the upper triangular part as well, although they may now be interlaced with some auxiliary operations due to the low rank correction term: see Figure 21.

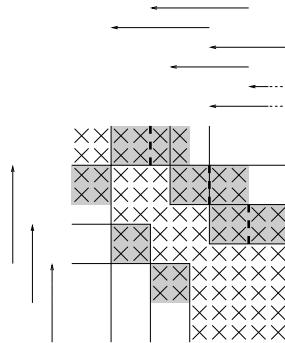


Figure 21: Modification of Figure 8(a) in case of a rank structured matrix which satisfies (3), i.e., $A - A^H = \text{Rk } p$. The unitary-weight representation for the structured upper triangular part can be obtained from that of the structured lower triangular part, in Hermitian transposed form, by interlacing it with a correction term of rank at most p ; in the figure we have $p = 2$.

To establish this claim, we note that by restricting (3) to the structured upper triangular part, it follows that the Givens-weight representation for the structured upper triangular part of A can be obtained by taking the Hermitian transpose of the representation for the structured lower triangular part, to which should be added the structured upper triangular part of the low rank correction term $\text{Rk } p$. See also [3]. Hence this updating procedure can be achieved as explained in Section 7, hereby establishing our claim.

Thus we demonstrated that indeed, the representation can be chosen in

such a way that the unitary operations of the lower triangular part appear in the upper triangular part as well, as in Figure 21. We explained above that this induces the peeling-off operations in the preparative-columns phase to be applied just as easily as in the purely Hermitian case. The algorithm can then go on by keeping track of the low rank correction term $R_k p$ whole the time, and by bringing the upper triangular part back to the form of Figure 21 at the end of each QR-step. (Since this updating requires only information about the lower triangular part and the low rank correction term, one can again forget about the upper triangular part during the residual-columns phase, in much the same way as in Section 5).

9 Numerical experiments

In this section we report on the results of some numerical experiments. The algorithms were implemented in Matlab*. The experiments were executed on an Intel PC running Matlab Version 7.0.1.24704 (R14) under Linux having 1GByte of memory and an Intel Pentium 4 processor running at 3.2 GHz. The software of these experiments can be requested from the authors.

We constructed symmetric rank structured matrices in $\mathbb{R}^{n \times n}$, with $n = 2^k$ for $k = 4, \dots, 10$. Starting from a diagonal matrix containing the desired eigenvalues, which were uniformly randomly chosen in the interval $[-1, 1]$, we applied to this matrix a similarity transformation based on a ‘disturbing’ sequence of Givens arrows of width r . This resulted in a rank structure whose structure blocks are situated just below the main diagonal, following immediately one after the other. The upper triangular part is then known by symmetry.

Since it can be argued that the above construction yields rather special rank structured matrices, we next applied a ‘randomization’ procedure. We did this by applying an additional similarity transformation based on Givens transformations, computed in a structure-preserving way. Note that symmetry is preserved during all these operations. A detailed description of this randomization method will not be given here.

We note here that all operations were performed using an efficient Givens-weight representation. More precisely, we used a Givens-weight representation which is *canonical of type 1*, as defined in [5].

For each size n , the above scheme was carried out for subsequent rank indices $r = 1, 2, 3$. For each of these sizes and each of these rank indices, there were considered 5 samples. Figure 22 shows for each size $n = 2^k$ and each rank index r the execution time $T_{k,r}$ averaged over the 5 samples of computing the eigenvalue spectrum by means of the explicit QR-algorithm. The shift element of the QR-algorithm was chosen each time in the classical way to be the bottom rightmost entry of the matrix.

To check that the computational complexity is quadratic in the size of the matrix, Figure 23 shows the fraction $T_{k+1,r}/T_{k,r}$ averaged over the 5 samples and over the ranks $r = 1, 2, 3$.

*Matlab is a registered trademark of The MathWorks, Inc.

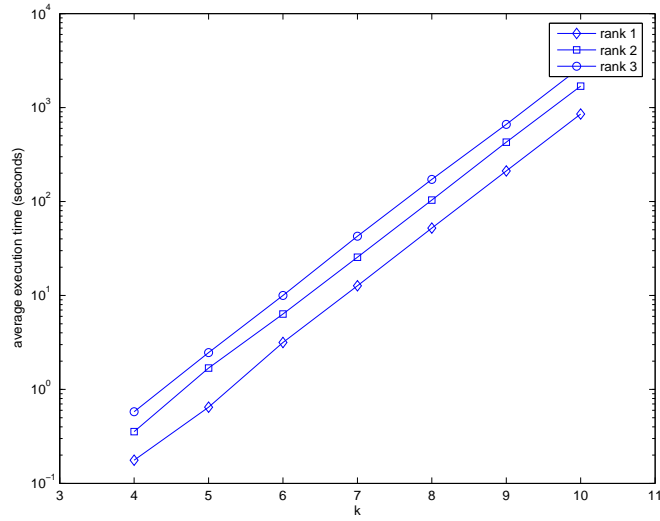


Figure 22: Average execution time for 5 random samples of size $n = 2^k$ and rank $r = 1, 2, 3$

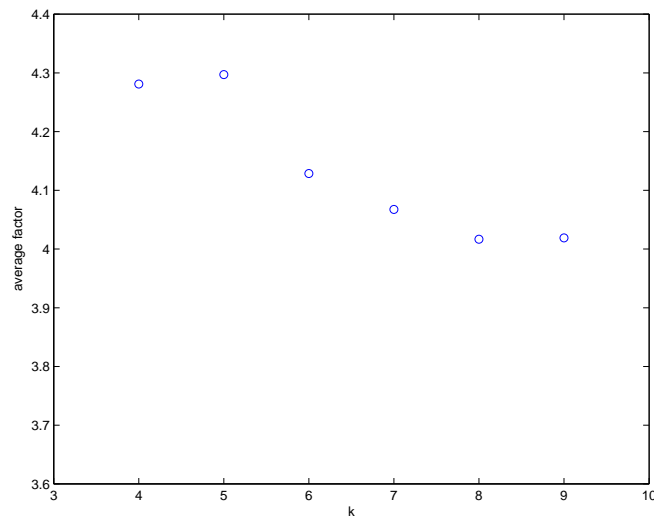


Figure 23: Fraction $T_{k+1,r}/T_{k,r}$ averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of size $n = 2^k$

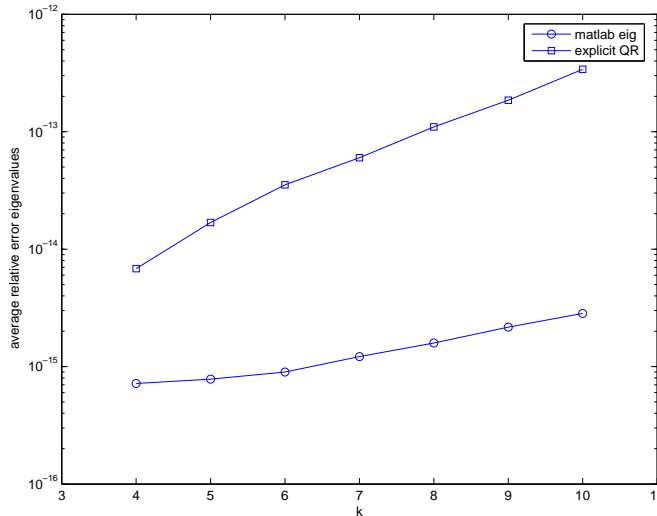


Figure 24: Relative 2-norm of the error of the eigenvalues averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of the size $n = 2^k$.

To measure the accuracy of the algorithm, we computed the relative norm $\frac{\|\lambda - \lambda_0\|_2}{\|\lambda_0\|_2}$, where $\lambda_0, \lambda \in \mathbb{R}^n$ denote the vectors containing the exact and computed eigenvalues. Figure 24 shows the relative 2-norm of the error averaged over the 5 samples and the rank indices $r = 1, 2, 3$.

It can be noted from this figure that the algorithm is somewhat less accurate in comparison to the built-in command `eig` of matlab. But we believe that this is not inherent to our method. We have two reasons for believing this:

1. The performance of our algorithm turns out to be very sensitive to the routine used for the computation of the Givens transformations. The results reported here use the implementation of Givens transformations as described in [10]. But we found out that the accuracy could be improved by using the built-in Givens routine `givens` of matlab. The resulting error lies then almost in the middle of the two curves shown in Figure 24; but we preferred here to be consistent with the implementation in [10], which we used also in our previous papers.
2. The same phenomenon occurs on *full* matrices, where we noted the same discrepancy between the accuracy of the Householder and Givens implementations of the QR-algorithm, as well as the discrepancy between the different implementations for the Givens routine. This discrepancy occurred even when computing a single QR-factorization.

Let us mention that we were able to bring the accuracy of the computed eigenvalues to the same level as the matlab `eig` by means of an iterative refinement, based on the Rayleigh quotient method. This method uses the routine for multiplying the Givens-weight representation with a vector [5].

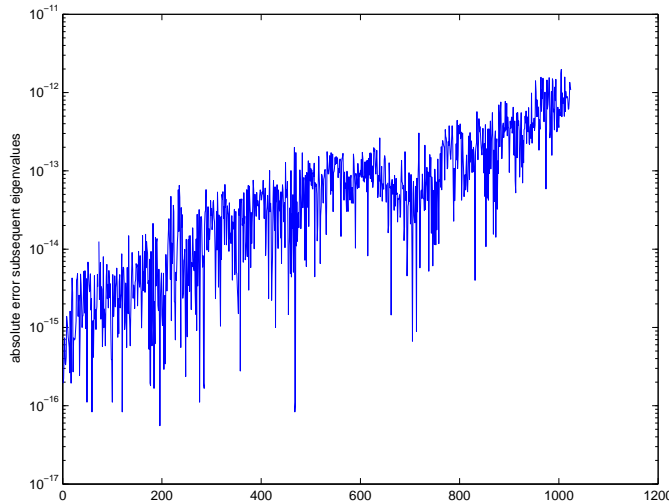


Figure 25: Absolute error of the subsequent eigenvalues as they were found for a sample of rank index $r = 3$ and size $n = 2^{10}$.

Figure 25 shows the error of the subsequent eigenvalues as they were found for a particular sample of rank index $r = 3$ and size $n = 2^{10}$. The absolute values of the corresponding eigenvalues are shown in Figure 26. It can be seen that the error tends to increase during the QR-algorithm.

To check that the computational complexity is quadratic as a function of the rank index r , we considered the execution time T_r for matrices of fixed size $n = 2^8 = 256$ and varying rank index $r = 2^l$ with $l = 0, 1, \dots, 6$. Figure 27 gives the fraction T_{2r}/T_r for subsequent rank indices. Note that the fraction tends to approximate 4 for large rank indices r but is much smaller for small values of r . In fact, the fraction tends even to decrease at the end; but this is artificial since for high ranks, the ranks become relatively large w.r.t. the size $n = 2^8$, so that the distribution of Givens transformations on the ‘borders’ of the matrix starts to have a non-negligible impact on the timings.

We note that similar experiments have been performed also for other kinds of rank structures, which are less regular than the one we took for the above numerical experiments. The results of these experiments were similar to those reported above.

10 Conclusion

In this paper we have shown how to perform the explicit QR-algorithm on a rank structured matrix, using the Givens-weight representation. The algorithm was especially useful in case where the matrix is Hermitian, or Hermitian plus some low rank correction, in which case the complexity is $O((r + s)^2 n)$ for each QR-step. The results of some numerical experiments were reported.

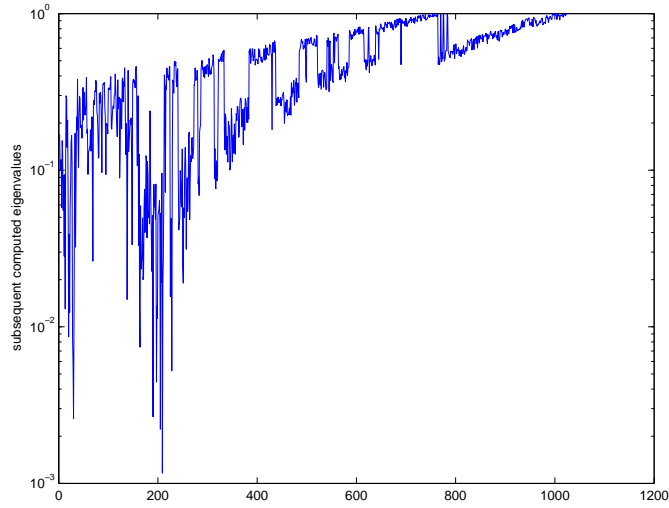


Figure 26: Absolute value of the subsequent eigenvalues as they were found for the sample in Figure 25.

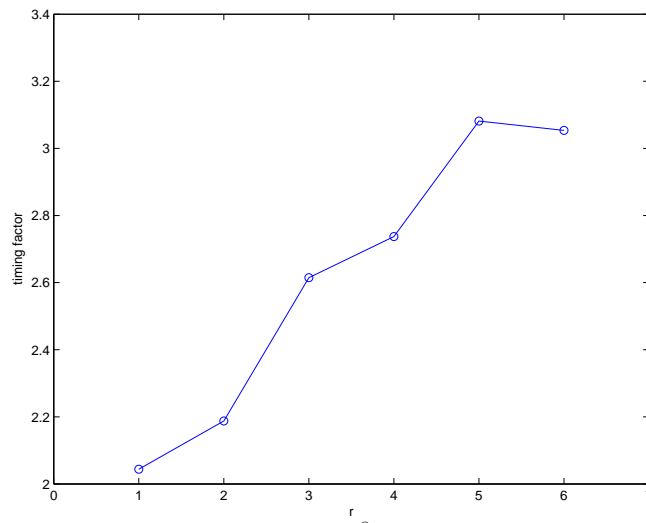


Figure 27: Fraction T_{2r}/T_r for size $n = 2^8$ in function of the rank index $r = 2^l$ with $l = 0, 1, \dots, 7$.

References

- [1] P. Arbenz and G. H. Golub. Matrix shapes invariant under the symmetric QR algorithm. *Numerical Linear Algebra with Applications*, 2(2):87–93, 1995.
- [2] D. A. Bini, F. Daddi, and L. Gemignani. On the shifted QR -iteration applied to Frobenius matrices. *Electronic Transactions on Numerical Analysis*, 18:137–152, October 2004.
- [3] D. A. Bini, L. Gemignani, and V. Y. Pan. Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations. *Numerische Mathematik*, 100(3):373–408, 2005.
- [4] S. Delvaux and M. Van Barel. Structures preserved by the QR-algorithm. *Journal of Computational and Applied Mathematics*, 187(1):29–40, 2005. DOI 10.1016/j.cam.2005.03.028.
- [5] S. Delvaux and M. Van Barel. A Givens-weight representation for rank structured matrices. Technical Report TW453, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 2006.
- [6] S. Delvaux and M. Van Barel. A QR-based solver for rank structured matrices. Technical Report TW454, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 2006.
- [7] S. Delvaux and M. Van Barel. Rank structures preserved by the QR-algorithm: the singular case. *Journal of Computational and Applied Mathematics*, 189:157–178, 2006.
- [8] Y. Eidelman, I. C. Gohberg, and V. Olshevsky. The QR iteration method for Hermitian quasiseparable matrices of an arbitrary order. *Linear Algebra and its Applications*, 404:305–324, July 2005.
- [9] D. Fasino. Rational Krylov matrices and QR-steps on Hermitian diagonal-plus-semiseparable matrices. *Numerical Linear Algebra with Applications*, 12(8):743–754, October 2005.
- [10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.
- [11] M. Van Barel, R. Vandebril, and N. Mastronardi. An orthogonal similarity reduction of a matrix into semiseparable form. *SIAM Journal on Matrix Analysis and its Applications*, 27(1):176–197, 2005.
- [12] R. Vandebril, M. Van Barel, and N. Mastronardi. An implicit QR -algorithm for symmetric semiseparable matrices. *Numerical Linear Algebra with Applications*, 12(7):625–658, 2005.