

A Hessenberg reduction algorithm for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW460, May 2006



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

A Hessenberg reduction algorithm for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW460, May 2006

Department of Computer Science, K.U.Leuven

Abstract

In this paper we show how to perform the Hessenberg reduction of a rank structured matrix under unitary similarity operations in an efficient way, using the Givens-weight representation which we introduced in an earlier paper. This reduction can be used as a first step for eigenvalue computation. We also show how the algorithm can be modified to compute the bidiagonal reduction of a rank structured matrix, the latter being a preprocessing step for computing the singular values of the matrix. Numerical experiments demonstrate the stability of this approach.

Keywords : rank structured matrix, (zero-creating) Givens-weight representation, Hessenberg reduction, eigenvalue computation, singular value computation, structure inheritance.

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

A HESSENBERG REDUCTION ALGORITHM FOR RANK STRUCTURED MATRICES

STEVEN DELVAUX *, MARC VAN BAREL *

Abstract. In this paper we show how to perform the Hessenberg reduction of a rank structured matrix under unitary similarity operations in an efficient way, using the Givens-weight representation which we introduced in an earlier paper. This reduction can be used as a first step for eigenvalue computation. We also show how the algorithm can be modified to compute the bidiagonal reduction of a rank structured matrix, the latter being a preprocessing step for computing the singular values of the matrix. Numerical experiments demonstrate the stability of this approach.

Keywords: rank structured matrix, (zero-creating) Givens-weight representation, Hessenberg reduction, eigenvalue computation, singular value computation, structure inheritance.

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

1. Introduction. In this paper we describe how for a rank structured matrix (sometimes also called quasiseparable matrix) with given Givens-weight representation, we can efficiently compute its Hessenberg form under the action of unitary similarity transformations. The algorithm will be particularly useful when the underlying matrix is Hermitian.

Hessenberg reduction is often the first step for computing the eigenvalues of a given matrix $A \in \mathbb{C}^{n \times n}$. This process transforms the given matrix into Hessenberg form by means of a unitary similarity transformation $A \mapsto Q^H A Q$, which may be based on either Givens or Householder transformations [8, 15]. Since the resulting Hessenberg matrix has the same eigenvalue spectrum as the original matrix we started from, the eigenvalue problem reduces to finding the eigenvalues of a Hessenberg matrix, for which then efficient algorithms can be applied such as the QR-algorithm [8, 11].

Classical algorithms for the Hessenberg reduction of banded matrices are contained in [13, 12]. Concerning Hessenberg reduction for rank structured matrices, we may refer to [6, 10] for the case of weakly semiseparable matrices of semiseparability rank one, and to [1] for semiseparable plus banded matrices of arbitrary semiseparability rank. But we should mention that these papers use what we call a uv -representation for representing these matrices, the latter implying some intrinsic restrictions to the class of matrices for which the algorithm can be applied. In the current paper, we will not assume such a restriction.

For the main cases of interest, the algorithm we describe in this paper is of complexity $O((r+s)n^2)$, 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.

*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.

This paper is organized as follows. In Section 2 we review some topics of the Givens-weight representation from [3], and we digress on the so-called zero-creating Givens-weight representations. Section 3 handles the Hessenberg reduction algorithm for a lower rank structured matrix. This section contains both a theoretical part concerning structure inheritance, as well as a practical part concerning the exploitation of these inheritance results. Section 4 deals with some variants of the algorithm, showing how to exploit symmetry or general rank structure in the upper triangular part during the algorithm. We also explain here how the algorithm can be modified to compute the bidiagonal reduction of a given rank structured matrix. Section 5 deals with some numerical experiments.

2. Givens-weight representation. Before proceeding to the algorithm, in the present section we provide and recall some topics concerning the Givens-weight representation.

2.1. General definitions. In this subsection we recall the main ideas of the Givens-weight representation. This subsection follows the exposition in [3], except that Definition 1 is more general here, and so the reader familiar with these ideas might wish to move directly to Section 2.2.

First we define rank structured matrices.

DEFINITION 1. (See [2]:) We define a rank structure \mathcal{R} on $\mathbb{C}^{m \times n}$ as a collection of so-called structure blocks $\mathcal{R} = \{\mathcal{B}_k\}_k$. Each structure block \mathcal{B}_k is characterized as a 4-tuple

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

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

$$\text{rank } A_k(i_k : m, 1 : j_k) \leq r_k, \quad \text{where } A_k = A - \lambda_k I.$$

Thus after subtracting the shift element λ_k from the diagonal entries, we must get a low rank block.

As a special case, when a structure block \mathcal{B}_k has shift element equal to zero, or when it is situated strictly below the main diagonal, then we call it pure. We sometimes denote such a structure block by $\mathcal{B}_{\text{pure},k}$.

Figure 2.1 shows an example with two structure blocks.

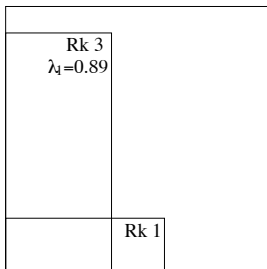


FIGURE 2.1. Example of a rank structure with two structure blocks. The left structure block \mathcal{B}_1 intersects the diagonal and has shift $\lambda_1 = 0.89$, while the second structure block \mathcal{B}_2 is 'pure'. The notation 'Rk r ' denotes that the structure block is of rank at most r .

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.

Now we recall the Givens-weight representation from [3]. First we assume that the rank structure is pure, i.e., that the shift elements are zero.

We will assume in what follows that we are working with a pure 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 [3]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a pure 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 2.2, serving to create zeros in the subsequent $\text{Rk } r_k$ 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 W_k obtained at the top border of the rank structure at the moment just after applying U_k . See Figure 2.2.

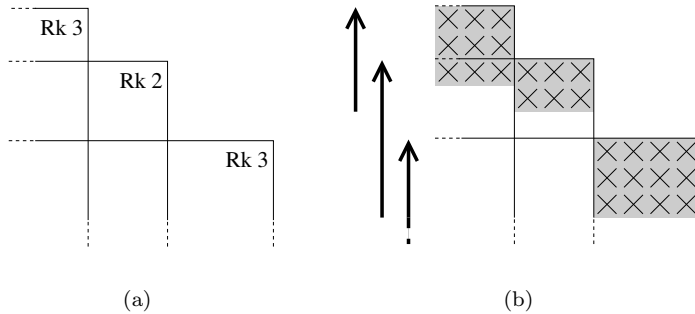


FIGURE 2.2. For the pure rank structure in the left picture, the right figure shows a schematic picture of the unitary-weight representation.

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 consider the subsequent weight blocks, proceeding from top to bottom of the matrix, and multiply them with the ‘decompressing’ unitary operations U_k^{-1} , each one acting of

course only on the columns on the left of its action radius. While this process proceeds from top to bottom of the matrix, we will gradually 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 $G_{i,i+1}$, and the index i will be called the *row index* of the Givens transformation.

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 having row index precisely one more than the previous one. This means that each Givens transformation is situated precisely one position below the previous one: see Figure 2.3.

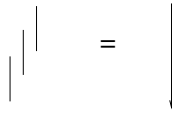


FIGURE 2.3. A Givens arrow consisting of 3 Givens transformations. Concerning this figure, we recall the reader that we consider each Givens transformation as ‘acting’ on the rows of an (invisible) matrix standing on the right of it, and hence that the Givens transformations in the figure should be evaluated 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* of the Givens arrow to be the largest and the smallest row index of the Givens transformations of which the Givens arrow consists, respectively. These notions have an obvious graphical interpretation.

DEFINITION 3. (*Givens-weight representation. See [3]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a pure 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 2.4.



FIGURE 2.4. 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. But we will not be concerned about this here, since we will work with a very special kind of Givens-weight representation, to be described next.

2.2. Zero-creating Givens-weight representation. The algorithm to be described in this paper requires a very special kind of Givens-weight representation. Namely, we must assume it to be in ‘zero-creating’ form, which means loosely speaking that each Givens transformation has to create a zero in the matrix. If this condition is not satisfied yet, then it has to be imposed, and the way how to do this will be the subject of the present subsection.

A first step is the following definition.

DEFINITION 4. (*Canonical Givens-weight representation. See [3]:*) Under the same conditions as in Definition 3, let there be given a Givens-weight representation such that

- the tails of the subsequent Givens arrows altogether are strictly monotonically proceeding upwards.

Then the Givens-weight representation is said to be canonical (of type 1).

Note that this definition implies that the Givens transformations can be stored in a 2-dimensional array in a natural way, by storing at the (i, j) th block entry of this array, the j th Givens transformation of the Givens arrow with tail index i . This property is convenient for implementation purposes.

Definition 4 is expressed only in terms of Givens transformations. We will need a condition on the weight matrix too.

DEFINITION 5. (*Zero-creating Givens-weight representation:*) Under the same conditions as in Definition 3, a Givens-weight representation is called zero-creating if (i) it is canonical of type 1, and (ii) for two subsequent weight blocks W_{k-1}, W_k of the weight matrix W for which $r_{k-1} < r_k$, the bottom $r_k - r_{k-1}$ rows of W_k must be in upper triangular form, $k = 1, \dots, K$. Here we assumed that a pseudo-structure block $\mathcal{B}_0 : (i, j, r) = (1, 0, 0)$ has been added to the rank structure.

The above condition on the weight matrix might seem strange at first, but this is nothing but a natural consequence of the zero-creating character. Note that this condition implies in particular that the top weight block W_1 must be upper triangular.

In fact we will need only a weaker version of this condition: we will need that (ii’) the first column of the top left weight block W_1 must be in upper triangular form. When this condition (ii’) is satisfied instead of the above condition (ii), we will speak of a *weakly zero-creating* Givens-weight representation.

Let us transform a given Givens-weight representation into (weakly) zero-creating form. Part of this process has already been sketched in [3], but we will provide here a detailed description. First we recall the pull-through lemma from [3].

LEMMA 6. (*Pull-through lemma:*) Given a unitary 3 by 3 matrix Q which is factorized as

$$Q = G'_{1,2} G_{2,3} G_{1,2},$$

then there exists a refactorization

$$Q = \tilde{G}'_{2,3} \tilde{G}_{1,2} \tilde{G}_{2,3}.$$

See Figure 2.5.

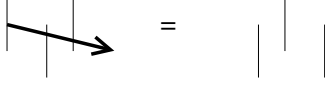


FIGURE 2.5. Pull-through lemma applied in the downward direction. One could imagine that the leftmost Givens transformation is really ‘pulled through’ the two rightmost Givens transformations.

Now in order to transform an arbitrary Givens-weight representation into a zero-creating one, we start on top of the matrix. We are concerned with condition (ii), and so we apply extra Givens transformations to bring the top weight block W_1 into upper triangular form. These Givens transformations are then simply ‘concatenated’ to the top unitary transformation U_1 . Next we restore condition (i) by applying the pull-through lemma a maximal number of times in the downward direction, inside this unitary transformation U_1 : see Figure 2.6.

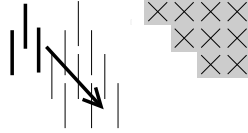


FIGURE 2.6. After applying Givens transformations to bring the top weight block in upper triangular form (indicated by the thick black lines), we apply pull-through operations to bring the top unitary operation U_1 of the Givens-weight representation to canonical form of type 1.

Note that these operations have led to the top unitary operation U_1 being in canonical form of type 1. We can then split $U_1 = U_{1,\text{new}} U_{1,2}$, where $U_{1,2}$ consists of the bottom r_2 Givens arrows, i.e., those which act entirely on the rows of the next weight block W_2 . These Givens transformations are then ‘transferred’ to the unitary operation U_2 by enlarging their action radius. This means that these transformations are applied to all the columns lying between their current and their future radius, as indicated by the thick black vertical lines in Figure 2.7.

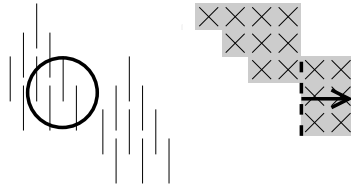


FIGURE 2.7. Having brought U_1 into zero-creating form in Figure 2.6, its bottom Givens arrows (which are circled by the thick black line) can be transferred to the next unitary operation U_2 by enlarging their action radius as indicated.

Suppose then that in the reduction process to zero-creating form, we have arrived at the unitary transformation U_k for certain $2 \leq k \leq K$. We can then do the same as before: first we apply extra Givens transformations to bring the bottom $r_k - r_{k-1}$ rows

of the weight block W_k into upper triangular form, provided this number is greater than or equal to two. These Givens transformations are then just concatenated to the unitary transformation U_k ; this is valid since it can be shown that for these bottom $r_k - r_{k-1}$ rows, there is no overlap with the Givens transformations of the unitary operations U_{k-1}, \dots, U_1 above. We can then again restore condition (i) by applying the pull-through lemma a maximal number of times in the downward direction, inside this unitary transformation U_k . We can then split $U_k = U_{k,\text{new}}U_{k,k+1}$, as before, and so on.

At the end of this process we will obtain a Givens-weight representation in zero-creating form. A piece of this is shown in Figure 2.8. Note that the tails of the subsequent Givens arrows in this figure (indicated by the thick black lines) are indeed strictly monotonically proceeding upwards.

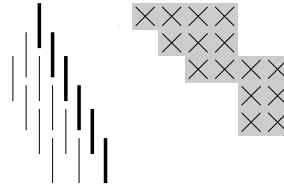


FIGURE 2.8. Part of a zero-creating Givens-weight representation.

To explain the name zero-creating, it can be shown that each of the Givens transformations of a zero-creating Givens-weight representation has to create a zero in an appropriate place of the matrix. We will need only a weaker version of this result.

LEMMA 2.1. (*Weakly zero-creating:*) *If a weakly zero-creating Givens-weight representation is given, then the tails of the subsequent Givens arrows create zeros in the first column of the matrix.*

PROOF. Suppose that we compress the given matrix by means of the unitary operations of the Givens-weight representation. At the moment that we applied the tail of the first Givens arrow $G_{k,k+1}$, we can apply the remaining Givens transformations of which the Givens arrow consists. But instead of doing this, we can already apply now the tail of the next Givens arrow $G_{k-1,k}$, because this operation acts on rows $k-1, k$ which are strictly disjoint from the rows $k+1, k+2, \dots$ on which the remaining part of the current Givens arrow acts. Repeating this argument, we can rearrange the compressing unitary operations as V_2V_1 , where V_1 consists of the succession of all the subsequent tails of the Givens arrows, and V_2 contains the remaining parts of the Givens arrows. Since the weakly zero-creating character implies that the first column of the top weight block W_1 must be in upper triangular form, and because V_2 only acts on rows strictly below the top non-zero entry of this column, we see that these zeros in the first column must have been created purely by means of the tails of the subsequent Givens arrows. \square

This lemma states that the tails of the Givens arrows, which are indicated by the thick black lines in Figure 2.8, have to create zeros in the first column of the underlying matrix. Both the meaning and the proof of this lemma can be illuminated by interpreting them in terms of this figure.

A similar property holds for the actual zero-creating Givens-weight representation. The idea is that the corresponding Givens transformations must create zeros in the

subsequent columns of the matrix. Due to the presence of the low rank blocks, extra zeros will be induced automatically in the further columns during this process. We can then look further, and create zeros in the first column which has not been zeroed out yet. It is possible to give an exact expression for the subsequent places where the zeros have to be created, but this expression is rather complicated, and we believe that it does not provide extra insight. Moreover, we will not need to know these details for the current paper.

Finally we describe here an alternative construction of zero-creating Givens-weight representations. The starting point is that we must possess over a ‘dual’ Givens-weight representation, which we describe now.

DEFINITION 7. (*Rank-decreasing:*) *A Givens-weight representation is called rank-decreasing if*

- *the tops of the subsequent Givens arrows altogether are strictly monotonically proceeding upwards.*

Note that this definition is the same as Definition 4 above, except that the word tails has been replaced by tops. Rank-decreasing Givens-weight representations were called *canonical of type 2* in [3].

Just as zero-creating Givens-weight representations are ‘efficient’ in the sense that each Givens transformation effectively creates a zero in the matrix, the efficiency of the rank-decreasing version consists in the fact that each Givens arrow effectively, i.e., definitively eliminates a row of the matrix, which is not touched by the next Givens arrows anymore, due to the monotonicity of the tops of the Givens arrows. The reader should try to see this. Moreover, this feature is important since it allows rank-decreasing Givens-weight representations to arise in several, natural ways: see [3].

We recall here the general fact that Givens-weight representations can be constructed also by means of unitary *column* instead of row operations. In particular, the rank-decreasing concept can be translated to the case of a column-based Givens-weight representation as well, by replacing in Definition 7 the word upwards by rightwards.

Concerning this last paragraph, we recall that there has been described a so-called *swapping process* in [3] to transform a column-based into a row-based Givens weight representation, or vice versa. This process will also be illustrated in what follows. Let us motivate now our interest in this process.

THEOREM 8. (*Duality theorem:*) *Rank-decreasing and zero-creating Givens-weight representations are each other ‘duals’ under the swapping process described in [3]. This means that a rank-decreasing, column-based Givens-weight representation is ‘swapped’ by the swapping process into a zero-creating, row-based Givens-weight representation, and vice versa.*

PROOF. We consider here the case of a rank-decreasing, column-based Givens-weight representation. The swapping process is illustrated in Figure 2.9.

Let us comment on this figure. We start by bringing the bottom weight block W_K in upper triangular form by a series of auxiliary row operations. We can obviously do this in a ‘zero-creating’ way, a fact which is indicated by highlighting the tails of the subsequent Givens arrows in Figure 2.9(a); see also Figure 2.10.

Next, we spread out the bottom weight block W_K by applying the ‘decompressing’ unitary operation U_K^H to it. This operation U_K^H is shown in Figure 2.9(b), where we assume that the subsequent decompressing Givens transformations have to be applied from top to bottom. Since we assumed the given Givens-weight representation to be rank-decreasing, it can be checked that the operation U_K^H is such that its left-

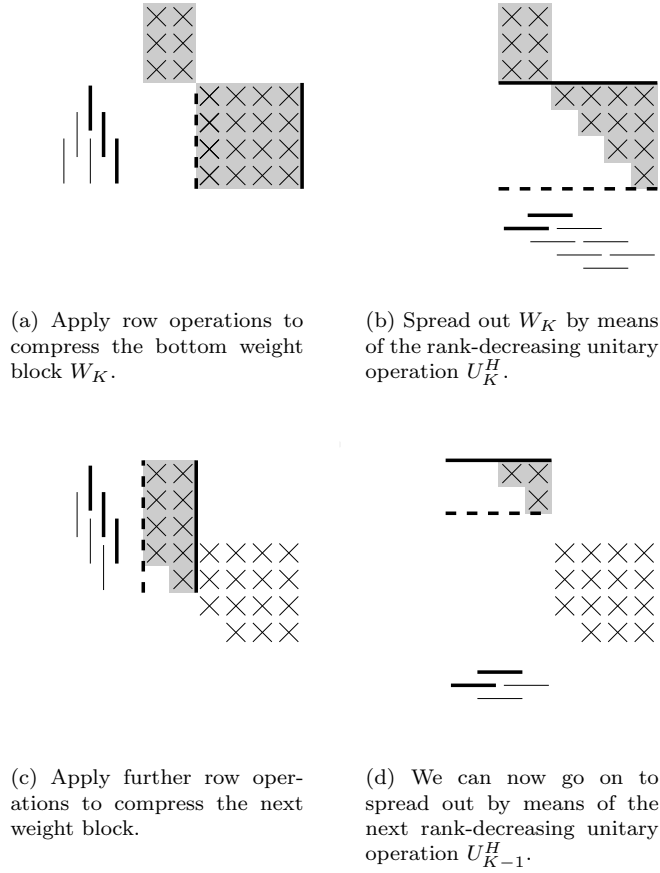


FIGURE 2.9. *Swapping from a rank-decreasing, column-based into a zero-creating, row-based Givens-weight representation.*

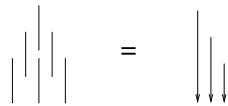


FIGURE 2.10. *Zero-creating character of Figure 2.9(a).*

most Givens transformations, highlighted in Figure 2.9(b), monotonically proceed leftwards. This property implies the crucial observation that the upper triangularity of the current weight block is *completely preserved* during the spreading-out process.

In particular, this preservation of upper triangularity implies condition (ii) in the definition of zero-creating Givens-weight representation to hold, i.e., when $r_k > r_{k-1}$, then the $r_k - r_{k-1}$ bottommost rows of the swapped weight block W_k are upper triangular: see e.g. the vanishing of the (7,3) element in Figure 2.9(c).

We have now arrived at the next weight block W_{K-1} . We compress this weight block by means of some new auxiliary row operations. By the fact that the upper triangularity of the previous weights was preserved, we can do this by just continuing

the zero-creating pattern of Givens transformations on the rows: see Figure 2.9(c).

It follows from the above observations that the swapped Givens-weight representation must indeed be zero-creating. The final weight matrix is shown in Figure 2.11.

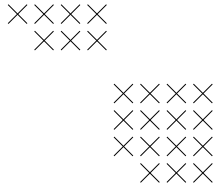


FIGURE 2.11. *Final weight matrix of the resulting zero-creating, row-based Givens-weight representation at the end of the swapping process.*

Conversely, we can swap from a zero-creating, row-based into a rank-decreasing, column-based Givens-weight representation. Since this process is ‘dual’ to the one above, it will not be shown anymore. \square

Summarizing, we have described now two ways of constructing (weakly) zero-creating Givens-weight representations, namely, either by the use of *pull-through* techniques, as in Figures 2.6 and 2.7, starting from an arbitrary Givens-weight representation, or by the use of *swapping* techniques, as in Figure 2.9, assuming that a rank-decreasing Givens-weight representation is available. The pull-through scheme will be the one that we use in practice.

In the next section, we will come to the main theme of this paper.

3. Hessenberg reduction. Assuming now that we have a weakly zero-creating Givens-weight representation at our disposal, we can start the reduction algorithm into Hessenberg form.

3.1. Structure propagation. First we will consider the structures which are propagated during the reduction of a matrix into Hessenberg form. First, by the fact that we use only unitary similarity transformations, it is easy to see that the properties of being Hermitian plus low rank, unitary plus low rank, and so on will be preserved by the reduction process. This corresponds to what were called ‘polynomial structures’ in [2].

Still following the analogy with [2], we would like also rank structures to be preserved during the Hessenberg reduction process. To this end, we have to make the assumption that the reduction process is based on Givens transformations, i.e., that in the k th step of this process, $k = 1, \dots, n - 1$, the k th column of the matrix is brought in Hessenberg form by an upward pointing sequence of Givens transformations $G_{n-1,n}, \dots, G_{k+1,k+2}$. In order to preserve the eigenvalue spectrum, we multiply then with the Hermitian transposes of these Givens transformations $G_{n-1,n}^H, \dots, G_{k+1,k+2}^H$ to the columns, hereby not destroying the created zeros anymore. This process is illustrated for a 4 by 4 example in Figure 3.1.

Clearly, it will be sufficient if we can characterize the structure propagation after the Hessenberg reduction of the *first* column.

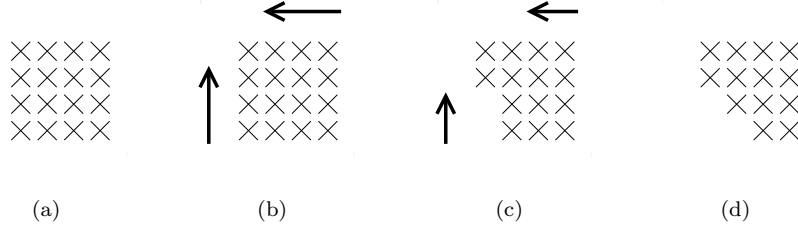


FIGURE 3.1. Hessenberg reduction of a 4 by 4 matrix.

THEOREM 9. (*Structure propagation:*) Let $A \in \mathbb{C}^{n \times n}$ be a matrix satisfying a structure block $\mathcal{B} = (i, j, r, \lambda)$ with $i \geq 2$, such that the first column of $A|_{\mathcal{B}}$ is not equal to the zero vector. (Here $A|_{\mathcal{B}}$ denotes the submatrix of A which is ‘cut out’ by \mathcal{B}). Then by reducing the first column of A into Hessenberg form by means of Givens transformations, the structure block \mathcal{B} moves one position to the bottom right position, i.e., it transforms into a new structure block $\tilde{\mathcal{B}} = (i + 1, j + 1, r, \lambda)$: see Figure 3.2.

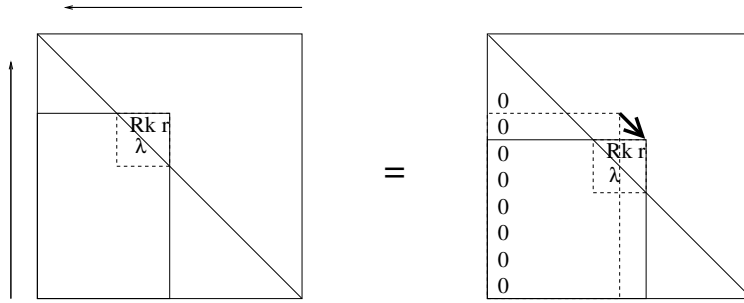


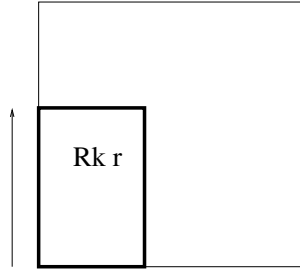
FIGURE 3.2. Structure propagation after the Hessenberg reduction of the first column.

PROOF. First, we show that the proof can be reduced to the case of a *pure* structure block. To this end, note that by definition, the matrix $A_{\text{pure}} := A - \lambda I$ satisfies the pure structure block $\mathcal{B}_{\text{pure}}$ which is obtained from \mathcal{B} by putting the shift element equal to zero. Now let us denote with Q^H the unitary row transformation which reduces the first column of A into Hessenberg form. Since Q^H does not involve the top row of the matrix, it must reduce also the first column of $A_{\text{pure}} = A - \lambda I$ into Hessenberg form. Moreover, it holds that $Q^H A Q = Q^H A_{\text{pure}} Q + Q^H (\lambda I) Q = Q^H A_{\text{pure}} Q + \lambda I$. Hence indeed, it will suffice to prove the theorem for the matrix A_{pure} .

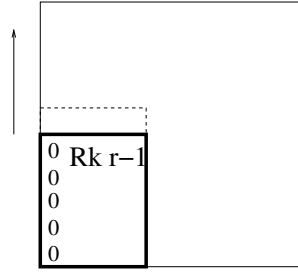
To prove this remaining case, we use an easy argument based directly on the Givens transformations, as shown in Figure 3.3.

Let us comment on this figure. The figure starts by applying Givens transformations to the rows, in order to make the first column Hessenberg. Thus at the moment that we are at the point of ‘leaving’ the structure block \mathcal{B} , the first column of $A|_{\mathcal{B}}$ will be entirely zero, except for its top element, which is non-zero by the assumptions in the theorem. The key observation is then that the top row of $A|_{\mathcal{B}}$ can not be written as a linear combination of the further rows, and hence the structure block obtained by removing this top row from \mathcal{B} must be of rank at most $r - 1$: see Figures 3.3(a) and 3.3(b).

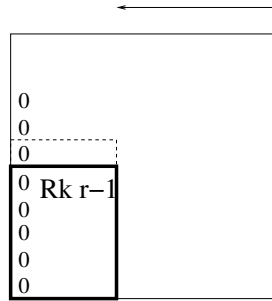
The proof can now be finished by remarking that this $\text{Rk}(r - 1)$ structure block can not be destroyed anymore by the next row operations. On the other hand, the application of the column operations will result in the structure block being enlarged by one column, and with rank increased by one: see Figures 3.3(c) and 3.3(d). \square



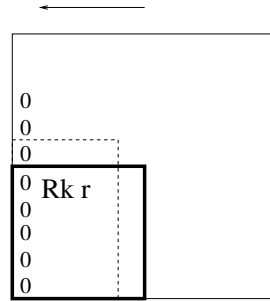
(a) Apply the first Givens transformations to the rows. They will transform the given $\text{Rk } r$ block into a $\text{Rk}(r - 1)$ structure block with one row less.



(b) Apply the remaining Givens transformations to the rows.



(c) Apply the transposed Givens transformations to the columns, until the $\text{Rk}(r - 1)$ structure block is reached.



(d) Enlarge the $\text{Rk}(r - 1)$ block into a $\text{Rk } r$ structure block with one more column, and apply the remaining Givens transformations to the columns.

FIGURE 3.3. The figure shows the propagation of pure structure blocks by the Hessenberg reduction process directly in terms of the Givens transformations.

It should be noted that the above theorem holds only under the condition that the first column of $A|_{\mathcal{B}}$ is nonzero. Although this condition is very mild, it excludes the case of rank zero structure blocks; in this case, one should work with the top induced rank one structure block instead: see Figure 3.4.

Let us note some more topics concerning the above proof.

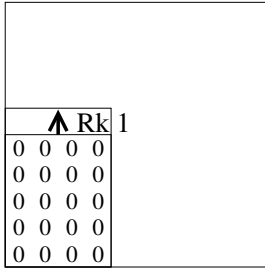


FIGURE 3.4. In case of a rank zero structure block, one should work with the top induced Rk 1 structure block which is indicated in the figure.

REMARK 10.

1. By repeating the argument in Figures 3.3(a) and 3.3(b) several times, it follows that the given structure block can be transformed into a new structure block with k rows less, and with rank diminished by k , by means of a sequence of Givens arrows of width k creating zeros in the first k columns: see also [4].
2. It is clear that in finite precision arithmetic, creating zeros in the first column and then expecting the rank to decrease, as in Figures 3.3(a) and 3.3(b), is likely to be an unstable procedure. This is because if the first column is close to the zero vector, then it can severely deviate from the overall column space of the low rank block, due to round-off errors. The solution to this problem will be to compute the zero-creating Givens-weight representation by means of the stable techniques which were described in Section 2.2.

Although Theorem 9 was stated for general shift elements λ , we will first exploit it for *pure* rank structures in what follows. This process will be described in the next subsection.

3.2. Hessenberg reduction algorithm. In this subsection we describe the algorithm for the Hessenberg reduction of the given rank structured matrix. We start with an algorithm for bringing the first column into Hessenberg form. The upper triangular part of the matrix will currently be assumed to be unstructured.

Note that the Givens transformations that bring the first column in Hessenberg form have already been *precomputed*, precisely by the concept of weakly zero-creating Givens-weight representation. Indeed, Lemma 2.1 guarantees that these transformations are nothing but the tails of the subsequent Givens arrows, as highlighted by the thick black lines in Figure 2.8. This means that the algorithm will suffice with ‘peeling off’ these tails from the subsequent Givens arrows, in a sense to be made exact further on.

The idea of the algorithm will be illustrated for the starting rank structure shown in Figure 3.5(a). The corresponding weakly zero-creating Givens-weight representation is shown in Figure 3.5(b). Note that this figure shows rather a *unitary*-weight than a *Givens*-weight representation, in order not to overload the figure.

The application of the Givens transformations $G_{k,k+1}$ to the rows will be achieved by what we call a *peeling-off process*. On the other hand, the application of the Givens transformations $G_{k,k+1}^H$ to the columns makes use of the general techniques for updating the Givens-weight representation under the influence of Givens transformations reported in [3], in the form of what we called there a *generalized swapping process*. These processes are shown in Figure 3.6.

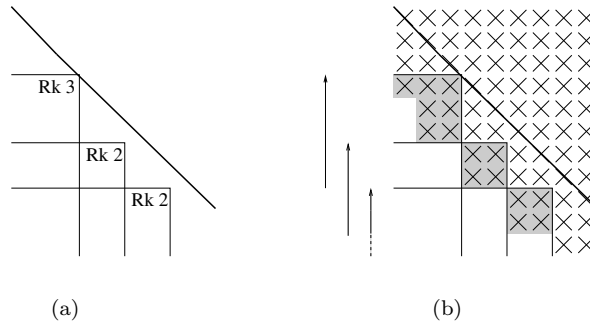


FIGURE 3.5. Starting rank structure with corresponding weakly zero-creating Givens-weight representation.

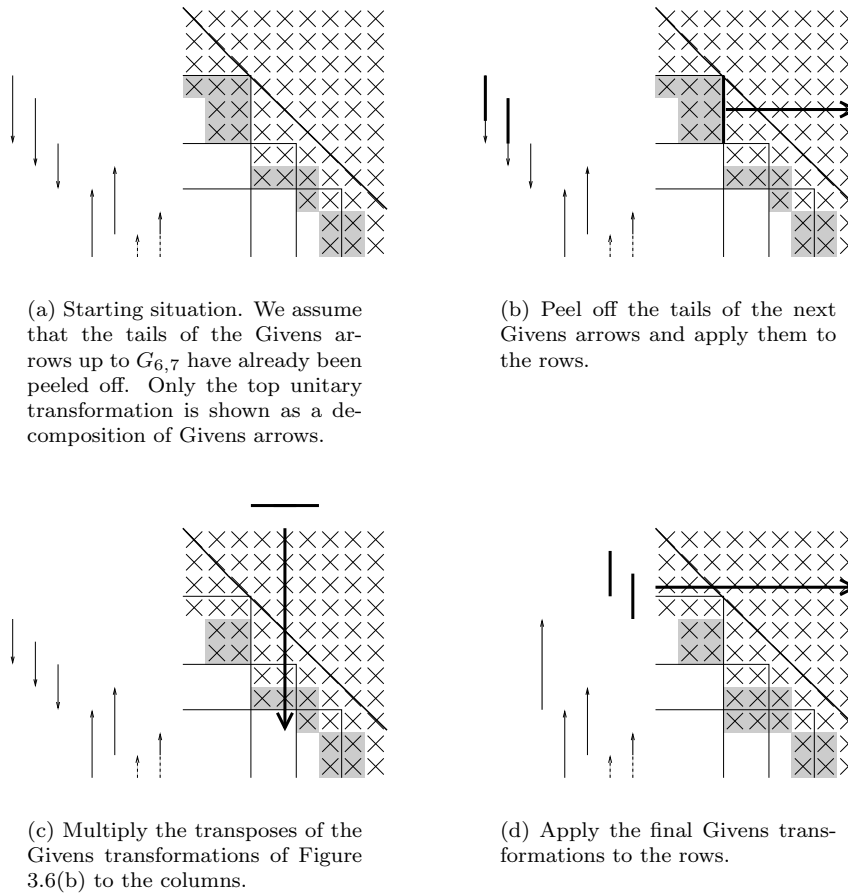


FIGURE 3.6. Hessenberg reduction of the first column (a-d).

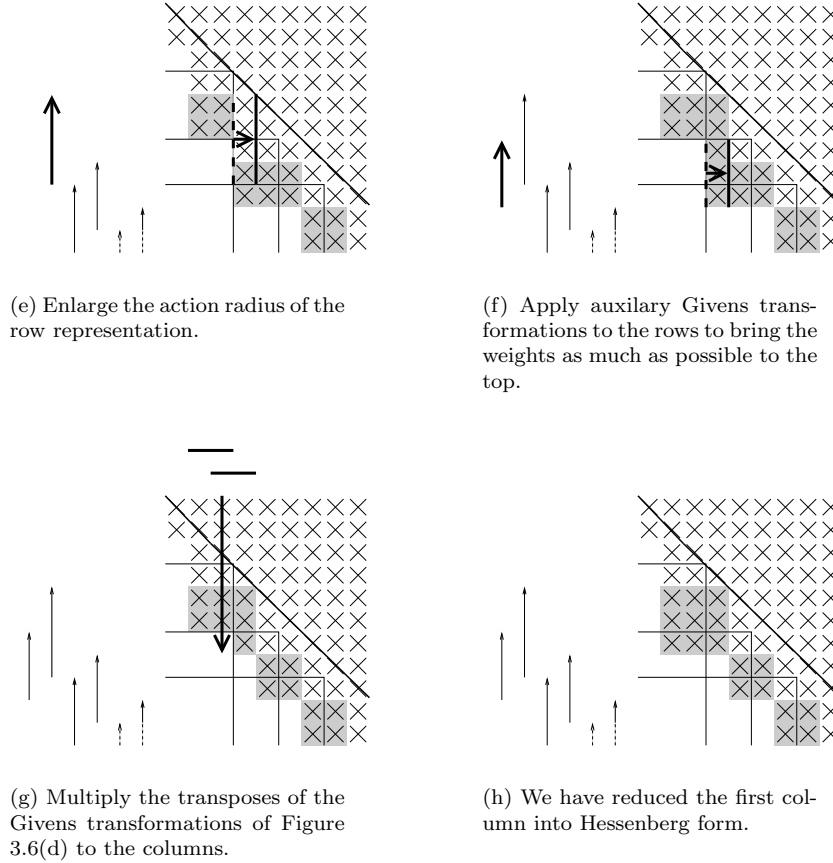


FIGURE 3.6. *Hessenberg reduction of the first column (e-h).*

Let us comment on this figure. Figure 3.6(a) shows the starting Givens-weight representation. It is assumed here that the first Givens transformations have already been peeled off from the structure, up to the transformation $G_{6,7}$, and that they have been applied to the rows and the columns.

The application of the *row* operations has led to the fact that the Givens arrows in several unitary transformations have ‘shrunk’ because they have ‘lost their tail’. Correspondingly, some of the originally grey elements of the two bottom structure blocks have turned white in Figure 3.6(a), since there is no Givens transformation anymore which acts on them. We note that this peeling-off process corresponds to the $\text{Rk } r$ structure blocks turning into $\text{Rk}(r - 1)$ structure blocks with one row less, as explained before.

The application of the *column* operations has led to the fact that the original unitary operations U_k are now interlaced with some auxiliary Givens arrows which were computed during the algorithm. Furthermore, some of the elements, such as the (9,6) element in Figure 3.6(a), are assumed to be disturbances coming from the application of the previous Givens transformations to the columns.

Now we are at the moment of peeling off the next Givens transformations. This

will cause the corresponding Givens arrows to lose their tail. While applying these Givens transformations, we exploit the fact that the result after their application has already been partially precomputed. This is why we only apply them to the columns strictly on the right of their current action radius: see Figure 3.6(b).

Note that after their application, the top row of the top weight block will have been ‘completely released’ in the sense that there are no Givens transformations anymore which act on it. This means that these elements turn from grey into white: see Figures 3.6(b) and 3.6(c).

To complete the similarity transformation, we should apply the Hermitian transposed operations to the columns. This is done in Figure 3.6(c).

We can then go on to apply the final Givens transformations for bringing the first column in Hessenberg form: see Figure 3.6(d). We would then like to complete the similarity transformation and apply these final Givens transformations on the columns too. But this means that we are at the point of contaminating the structure block in columns 2,3 with some of the elements in the column on the right of it. This contamination would lead to a mix of real-size elements and weights, which is definitely not allowed.

The solution consists in enlarging the action radius of the Givens-weight representation. This means that we bring the contaminating column, which is standing just on the right of the structured part, ‘into’ the representation: see Figures 3.6(e) and 3.6(f).

Having done this, it is now safe to apply the desired Givens transformations to the columns. But we do not do this yet, since applying all these column operations would lead to a complete fill-in in the lower triangular part. We want to minimize this fill-in as far as possible, and therefore we first apply some auxiliary Givens transformations to the rows, to bring the newly introduced weights as far as possible to the top: see Figure 3.6(f).

Having done all these preparations, we can finally apply the desired Givens transformations to the columns: see Figure 3.6(g). We have then completely reduced the first column of the matrix into Hessenberg form.

Figure 3.6(h) shows the final Givens-weight representation. Note that the weight blocks have uniformly moved one position to the bottom right position, as predicted by Theorem 9.

Note that the algorithm has led to a Givens-weight representation in *interlaced* form, i.e., a unitary-weight representation for which each unitary transformation U_k has a natural decomposition of the form $U_k = \tilde{V}_k V_k$, where V_k is a unitary transformation situated on ‘top’ of U_k , containing what is left of the original Givens-weight representation, and \tilde{V}_k is a unitary transformation situated on the ‘bottom’ of U_k , containing the chain of auxiliary Givens transformations computed during the Hessenberg reduction algorithm: see Figure 3.6(h).

Now we would like to go on by making the second column Hessenberg. This means that we should first bring the Givens-weight representation back to its weakly zero-creating form, using the pull-through techniques of Section 2.2. In fact, these techniques must be slightly modified since we are working here with a Givens-weight representation in *interlaced* form, which is not a Givens-weight representation in the strict sense anymore.

Let us briefly apply these techniques to the present situation. First we apply an extra, upwards pointing Givens arrow, in order to restore the upper triangularity of the first column of the top left weight block in Figure 3.6(h): see Figure 3.7.

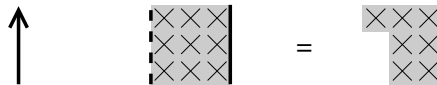


FIGURE 3.7. We apply an extra unitary operation to bring the first column of the top weight block in Figure 3.6(h) in upper triangular form.

The resulting unitary transformations, to which this extra Givens arrow is added, are shown in Figure 3.8.

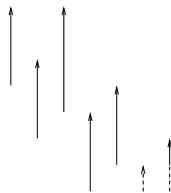


FIGURE 3.8. Resulting unitary transformations of Figure 3.6(h).

Next, we apply the pull-through lemma a maximal number of times in the downward direction, in order to bring the representation back into canonical form of type 1: see Figure 3.9.

While the pull-through process proceeds from top to bottom of the matrix, one should not forget to appropriately enlarge the action radii of the involved Givens transformations appropriately, as explained in Section 2.2. The treatment of a single step in this process is shown in Figure 3.10.

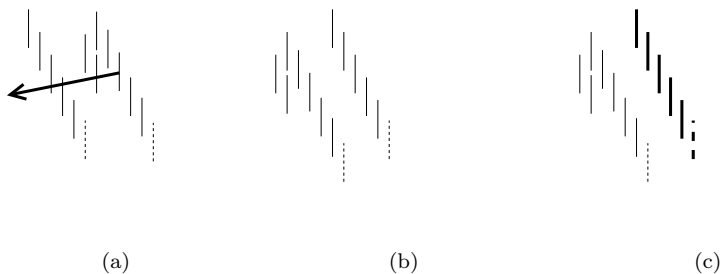


FIGURE 3.9. The left picture shows a decomposition of Figure 3.8 in terms of individual Givens transformations. It is then brought to canonical form of type 1 by repeatedly using the pull-through lemma, resulting in the situation of the middle picture.

At the end of the pull-through process, the Givens transformations indicated by the thick black lines in the Figure 3.9(c), will constitute the new tails of the Givens arrows, which will hence be peeled off during the Hessenberg reduction of the second column.

From the schematic illustration of the pull-through process in Figure 3.9, we can note the interesting fact that the (updated) chain of auxiliary Givens transformations, used for the auxiliary upward movement during the Hessenberg reduction algorithm, replaces the original tails of the Givens arrows at the end of the pull-through process:

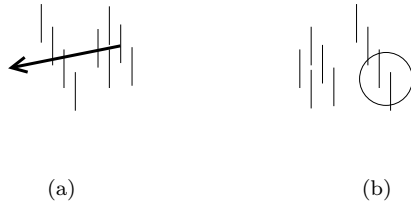


FIGURE 3.10. The picture shows a specification of Figure 3.9. After applying the pull-through operations inside the top unitary operation U_1 as shown in the left picture, the resulting unitary operation is decomposed as $U_1 = U_{1,\text{new}}U_{1,2}$, where the Givens transformations of $U_{1,2}$ are circled in the right picture. One can then enlarge the action radius of $U_{1,2}$ so that it is transferred to the next unitary operation U_2 . This scheme can then be repeated for the next unitary operations U_2, U_3, \dots

see Figure 3.9(c). Hence they will be precisely the Givens transformations to be peeled off during the Hessenberg reduction of the second column.

Finally, we have now obtained a Givens-weight representation which is back in weakly zero-creating form. We are then ready to bring the second column in Hessenberg form as well. Since this problem is completely similar to the Hessenberg reduction of the first column, it will not be explained anymore.

4. Some modifications to the algorithm. In this section we describe some variants to the Hessenberg reduction algorithm of the previous section. We start with a treatment of rank structure lying in the upper triangular part.

4.1. Exploiting rank structure in the upper triangular part. In this subsection we describe how the efficiency of the algorithm can be improved by an order of magnitude in case the matrix A has rank structure in its upper triangular part as well. The total algorithm complexity will decrease in this way from the cubic $O(n^3)$ to the quadratic $O((r + s)n^2)$, where r is some measure for the average semiseparability rank, and s is some measure for the bandwidth of the unstructured matrix part around the main diagonal.

We start with the case of rank structure originating from the fact that A is Hermitian, or more generally, when it is Hermitian plus a low rank correction. The algorithm can proceed then by just keeping track of the lower triangular part, since then the upper triangular part is known by symmetry. In the case where A is Hermitian up to some low rank correction, one should also keep track of the low rank correction matrix $\text{Rk } k$. The algorithm proceeds then by only computing each time the required superdiagonal element, by symmetry, next applying the row and column operation, and then again removing the superdiagonal element: see Figure 4.1.

It is easy to check that in this way, the complexity of the Hessenberg reduction algorithm for the k th column decreases to $O((r + s)n)$. Hence the *total* cost of the Hessenberg reduction algorithm reduces to $O((r + s)n^2)$.

Next we describe how, even if the matrix is not Hermitian up to some low rank correction, rank structure in the upper triangular part can be exploited. To this end, the reader should reacquire familiarity with the propagation mechanism for structure blocks in the *lower* triangular part in Figure 3.3. It can then be noted that the argument in Figures 3.3(a) and 3.3(b) crucially depends on the fact that zeros are created in the lower triangular part of the matrix during the Hessenberg reduction process.

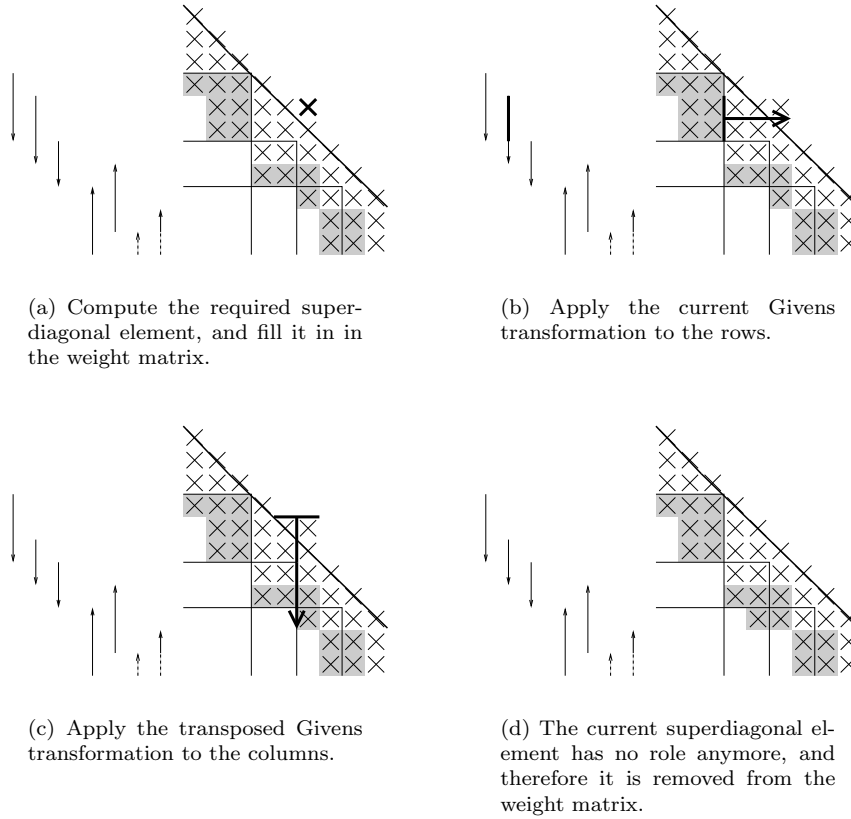


FIGURE 4.1. *Exploiting symmetry.*

The corresponding propagation of rank structure in the upper triangular part does *not* benefit from such a creation of zeros, as shown in Figure 4.2.

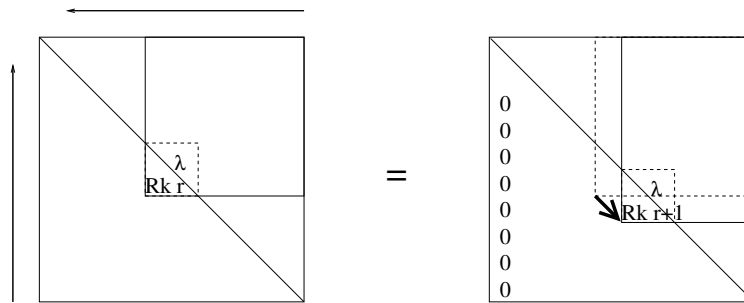


FIGURE 4.2. *Structure propagation in the upper triangular part after the Hessenberg reduction of the first column. Note that the structure block moves one position to the bottom right position, but that the original rank index r can increase to $r + 1$.*

It should be clear from Figure 4.2 that the ranks could soon start to increase. After a few columns have been transformed into Hessenberg form, there will probably be not much left of the rank structure, due to the growth of the rank indices of the

rank structure.

There is one notable exception to this idea, namely, we know that the ranks should stay constant also in case where A is a *unitary* matrix, or unitary up to some low rank correction, because then the rank structure in the upper triangular part is a direct consequence of that in the lower triangular part: see e.g. [5]. We will come back to this further on.

Anyway, also in the case of arbitrary rank structure in the upper triangular part, and where the ranks increase, we will give the tools to update the representation. These tools are based on the methods for updating a Givens-weight representation under the action of Givens transformations described in [3]. We saw there how to update the representation using what were called concatenation, pull-through, generalized swapping and generalized regression techniques.

Essentially, the only difference w.r.t. [3] is that we want to apply the disturbing Givens transformations *simultaneously* on rows and columns. This will lead to two methods being applied in the same time. We consider here the case where the Givens-weight representation of the upper triangular part is based on *row* operations. We will then simultaneously have a mixture of concatenation (for the row operations) and generalized regression techniques (for the column operations). See Figure 4.3.

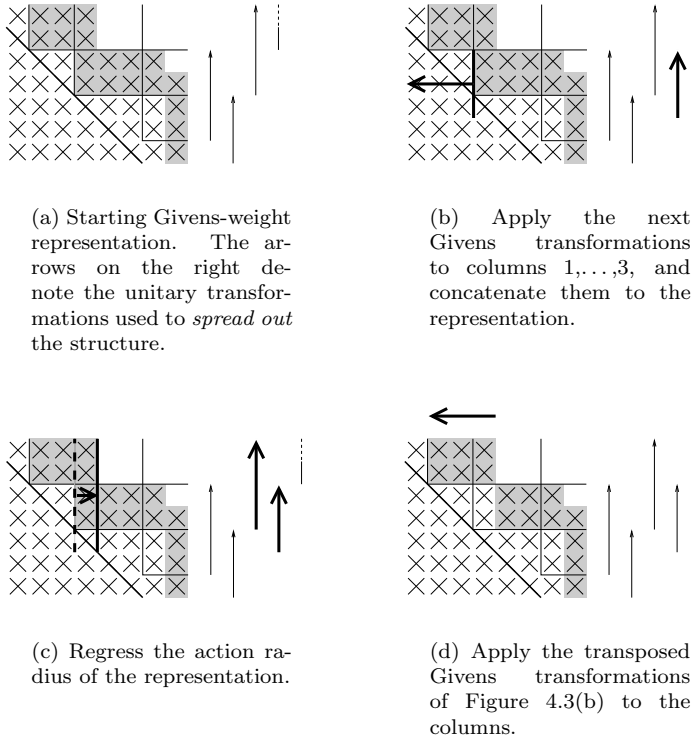


FIGURE 4.3. *Updating the structured upper triangular part during the Hessenberg reduction algorithm, assuming that it is represented by a row-based Givens-weight representation.*

Let us comment on this figure. Figure 4.3(a) shows the starting Givens-weight representation for the structured upper triangular part. Note that we preferred here to show the *decompressing* rather than the compressing unitary operations of the Givens-

weight representation, i.e., in order to obtain the full matrix, we should spread out by means of the subsequent unitary operations shown in Figure 4.3(a), evaluating from left to right. The bottommost unitary transformation is assumed to be a disturbance coming from previous steps.

We are now at the moment of applying the next Givens transformations to the rows: see Figure 4.3(b). We apply these transformations only to columns $1, \dots, 3$. For the application to the structured upper triangular part in columns $4, \dots$, we do not actually apply these Givens transformations, but instead just *concatenate* them to the Givens-weight representation. To see what this means, the reader should recall that the weight matrix contains a kind of ‘compressed’ information about the full matrix. Moreover, in order to obtain the real values of these elements, we should spread out the weight matrix by applying all the subsequent unitary operations, evaluating from left to right. The concatenation process means then simply that, at the very end of this decompressing process, one should apply the newly added Givens transformations as well, so that they can be considered from now on as being part of the Givens-weight representation.

Now we would like to apply the transpose of these Givens transformations to the columns. To do this in a valid way, we have to avoid a mixture of real-size elements and weights. Therefore we first regress the action radius of the representation: see Figure 4.3(c). We can then safely apply the Givens transformations to the columns: see Figure 4.3(d). The next operations are not shown anymore.

Finally, let us come back to the case where the rank structured matrix is *unitary plus low rank*. In this case, the above algorithm will yield that the ranks increase from r to $r + 1$ in each step, although we know that they should stay constant. Thus it should be possible to perform a *numerical approximation* of the structure after each step to approximate the ranks again by their actual values. If the structure blocks in the upper triangular part have been chosen wisely, this implies an extra term $O(r^2n)$ for the complexity in each step: see [3]. This *quadratic* term in the rank index can be avoided by just performing the numerical approximation procedure *only after each* $O(r)$ *steps* in the Hessenberg reduction process. The above $O(r^2n)$ term can then be distributed over $O(r)$ different steps, so that the total algorithm complexity becomes $O((r + s)n^2)$, just as in the Hermitian case.

Summarizing, we described now how the Hessenberg reduction algorithm can be modified to take advantage of the rank structure in the upper triangular matrix part. The next subsection considers a second modification to the algorithm.

4.2. Bidiagonal reduction algorithm. In this subsection we will explain how a rank structured matrix can be reduced to bidiagonal form. This means that we will apply an operation of the form $A \mapsto UAV$, where U and V are possibly different unitary transformations. Such a reduction does not preserve the eigenvalue spectrum, but it can be used as the first step for computing the *singular value decomposition* of the given matrix [7, 8].

Before starting, we recall that during the *Hessenberg* reduction process, the ranks in the lower triangular part were preserved, but those in the upper triangular part could increase from r to $r + 1$ in each step. The reason underlying this, was that the Hessenberg reduction only created zeros in the lower triangular part of the matrix, implying the argument of Figures 3.3(a) and 3.3(b) to be invalid for the structure blocks in the upper triangular part.

This conclusion does not hold anymore for the bidiagonal reduction process. Indeed, by exploiting the freedom of applying possibly different unitary transformations

U and V to rows and columns, this reduction succeeds in a creation of zeros in *both* lower and upper triangular part of the matrix. Hence it is easy to see that the structure propagation argument of Figures 3.3(a) and 3.3(b) will be valid now for *both* the lower and the upper triangular part, without any increase of ranks, in any case. On the other hand, a small price has to be paid in the sense that the structure propagation will hold now only for *pure* rank structures: see Figure 4.4.

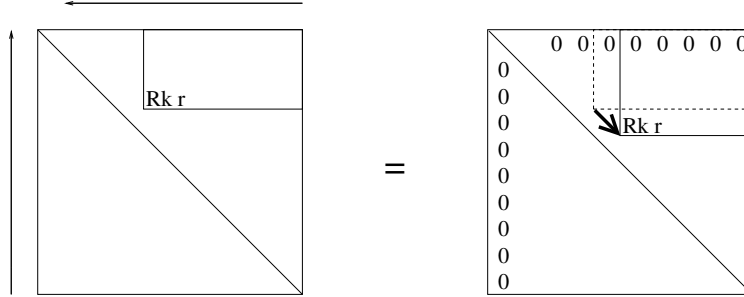


FIGURE 4.4. Structure propagation in the upper triangular part after the bidiagonal reduction of the first column and row. Note that the given, pure structure block moves one position to the bottom right corner, with rank index r being preserved. A similar result holds for the lower triangular part as well.

Let us discuss now the practical implementation of the bidiagonal reduction process. The algorithm will closely follow the Hessenberg reduction process of Section 3. The starting point will be that we have available a row-based Givens-weight representation for the lower triangular part, and a column-based Givens-weight representation for the upper triangular part of the given rank structured matrix.

By induction, let us assume then that zeros have been created already in the first $k - 1$ columns and rows of the matrix, $1 \leq k \leq n - 1$. We assume that the Givens-weight representation of the lower triangular part is in weakly zero-creating form. We can then peel off its tails $G_{n-1,n}, \dots, G_{k,k+1}$ to bring the k th column of the matrix in upper triangular form. During this process, the rank structure in the *upper* triangular part can be updated in much the same way as in Section 3, but now with the role of rows and columns interchanged. Having done this, we bring the Givens-weight representation of the upper triangular part back to its weakly zero-creating form. We can then peel off its tails $\tilde{G}_{n-1,n}^H, \dots, \tilde{G}_{k+1,k+2}^H$ and apply them to the *columns*, to create zeros in the k th *row* of the matrix. During this process, the rank structure in the lower triangular part can be updated in the same way as in Section 3, and so on.

Summarizing, we described now how the Hessenberg reduction algorithm could be modified to a bidiagonal reduction algorithm. We turn now to a final modification of the algorithm.

4.3. The case of non-pure structure blocks. We assumed in Section 3 that the structure blocks were *pure* and lying strictly below the main diagonal. In this subsection, we will explain how the Hessenberg reduction algorithm can be adapted to work for non-pure structure blocks as well. Along this line, further in this subsection we will describe how to reduce a matrix to lower semiseparable plus diagonal instead of Hessenberg form.

First we have to explain how the Givens-weight representation for such a non-pure rank structure looks like. We have to assume that the shift elements are *compatible* in the sense that two structure blocks intersecting each other on the main diagonal

have the same shift element. If the shift elements are compatible, then we can just represent the rank structured matrix as $D_\lambda + A_{\text{pure}}$, where D_λ is the diagonal matrix containing the shift elements, and A_{pure} is a matrix satisfying a pure rank structure. The latter matrix can now be represented by a usual Givens-weight representation.

Let us explain how the non-pure Givens-weight representation, as just described, can be updated during the algorithm. For simplicity, we will restrict to the case where only the lower triangular part of the matrix is rank structured.

Starting from the example of Figure 4.5, the algorithm is shown in Figure 4.6.

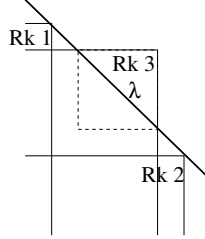


FIGURE 4.5. *Starting non-pure rank structure.*

Let us comment on this figure. The starting situation is shown in Figure 4.6(a); it is assumed here that the tails of the subsequent Givens arrows until $G_{6,7}$ have already been peeled off from the representation. Hence the next operations are at the point of entering the non-pure diagonal part of the non-pure structure block \mathcal{B} .

Still concerning this figure, let us note that the circle notation expresses that, after spreading out the matrix, the value λ should be added to each of the three originally circled entries of the structure block.

Since we are going to enter the non-pure part of \mathcal{B} , it is now the right moment to subtract the shift element λ from the entry in bottom right position of the structure block: this is indicated by the highlighted circle in Figure 4.6(a). Note that this subtraction will allow us to split a multiple of the identity matrix λI_4 from the structure.

We will now apply the next operations. First we enlarge the current action radius, and we apply an auxiliary unitary operation to bring the weights upwards: see Figures 4.6(b) and 4.6(c). Note that these operations can be performed exactly as in the shift-free case, since they only involve the *pure* part of the representation, so that the shift element λ is not involved.

We can then peel off the tails of the next Givens arrows, and apply them to rows and columns. Since the shift element λ appears only in the form of a multiple of the identity matrix λI_4 , which will clearly be preserved by this unitary similarity operation, we can carry out these operations exactly as in the shift-free case: see Figures 4.6(d) and 4.6(e).

These peeling-off operations have caused the structure block \mathcal{B} to move one position to the bottom right matrix corner. This means that the element which was originally situated in top left position of \mathcal{B} , has now come for free. We can restore this element to its real-size form by adding the shift element λ to it: see the highlighted circle in Figure 4.6(f). The next operations are not shown anymore.

Summarizing, we have now completely propagated through the non-pure structure block \mathcal{B} . During this process, the entry on bottom right of the structure block was ‘brought into’ the influence of the shift element in Figure 4.6(a), while the top left

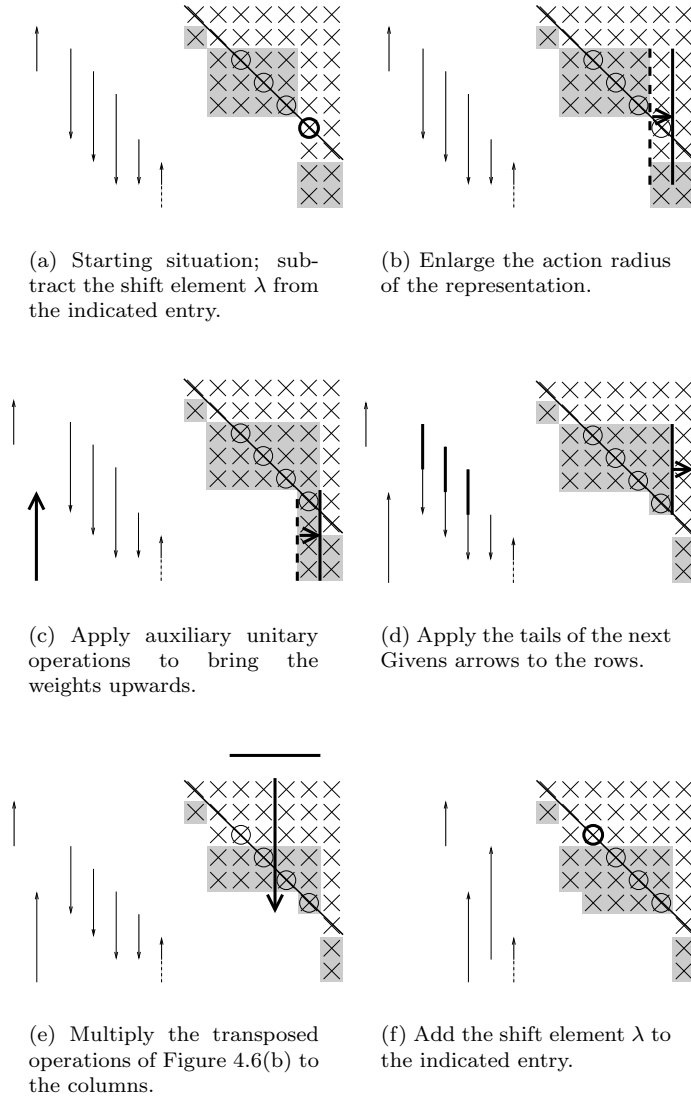


FIGURE 4.6. *The case of non-pure structure blocks.*

entry was ‘released’ in Figure 4.6(f), corresponding to the structure block propagating to the bottom right matrix corner.

We note that similar techniques for manipulating shift elements under the action of unitary similarity operations were used in [14], for the special case of reducing an arbitrary matrix into a diagonal plus semiseparable matrix of semiseparability rank one.

The connection with the paper [14] can be made even tighter. More precisely, we will describe how the Hessenberg reduction algorithm can be modified so as to transform a given rank structured matrix A into *lower semiseparable plus diagonal* form. Although this reduction could be achieved by first performing the Hessenberg

reduction, and subsequently using this as input for the algorithm in [14], we will describe here an intermingled version of the algorithm.

For definiteness, let us denote with $\lambda_k \in \mathbb{C}$, $k = 1, \dots, n$, the subsequent shift elements of the required, lower semiseparable plus diagonal structure. The corresponding structure blocks are defined as

$$\mathcal{B}_k : (i_k, j_k, r_k, \lambda_k) = (k, k, 1, \lambda_k).$$

The algorithm proceeds by introducing the shift elements in the *reverse* order $\lambda_n, \dots, \lambda_1$ on top of the matrix, and chasing them to the bottom right. We can start this process by adding the *last* shift element λ_n on top of the matrix, where it trivially satisfies a structure block $\mathcal{B} : (i, j, r, \lambda) = (1, 1, 1, \lambda_n)$: see Figure 4.7(a).

Let us note that in terms of the Givens-weight representation, this extra structure block \mathcal{B} of Figure 4.7(a) can be incorporated in the factorization $D_\lambda + A_{\text{pure}}$ by placing the value λ_n at the $(1, 1)$ element of D_λ , and subtracting it from the corresponding element of A_{pure} . It implies also an extra unitary transformation U_0 to be added to the Givens-weight representation of A_{pure} , in order to compress the *pure* variant of this extra structure block, which we denote as $\mathcal{B}_{\text{pure}}$, by bringing its weight completely to the top row of the matrix A_{pure} . Since the structure block $\mathcal{B}_{\text{pure}}$ is Rk 1, the Givens arrows of U_0 will be of width one.

Now we create zeros in the first column of A_{pure} by applying the tails of the subsequent Givens arrows $G_{n-1,n}, \dots, G_{1,2}$ to the rows, and their transposes $G_{n-1,n}^H, \dots, G_{1,2}^H$ to the columns. Because also the first column is involved in this process, the created zeros in the first column will be destroyed again. On the other hand, each of the existing structure blocks will be chased one position to the bottom right position, according to the implementation in Figure 4.6. In particular, it follows that the extra structure block \mathcal{B} will transform into a new structure block $\tilde{\mathcal{B}} : (i, j, r, \lambda) = (2, 2, 1, \lambda_n)$: see Figure 4.7(b).

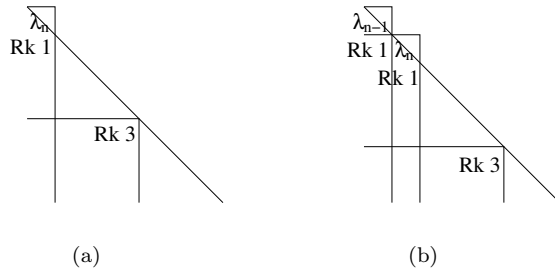


FIGURE 4.7. The left picture shows the starting situation for the lower semiseparable plus diagonal reduction algorithm. The first extra structure block \mathcal{B} has already been added to the top left corner. The right picture shows the situation after applying the first sweep of Givens transformations of the lower semiseparable plus diagonal reduction algorithm. Note that the structure blocks have moved to the bottom right corner, and that the next extra structure block has already been added to the top left corner.

Suppose then that we have already created $k-1$ lower semiseparable plus diagonal structure blocks in the top left corner of the matrix, and suppose that we are at the point of applying the k th upward sweep of Givens transformations, $2 \leq k \leq n-1$. Before doing this, we add the new shift element λ_{n+1-k} on top of the matrix, where it trivially satisfies a structure block $\mathcal{B} : (i, j, r, \lambda) = (1, 1, 1, \lambda_{n+1-k})$. We can then

correspondingly update the Givens-weight representation. Next, we can do the same as above, i.e., we apply similarity transformations based on the subsequent tails of the Givens arrows $G_{n-1,n}, \dots, G_{1,2}$ of the matrix A_{pure} in order to chase each of the existing structure blocks one position to the bottom right matrix corner.

Note that at the end of this process, the given rank structured matrix will be brought completely into lower semiseparable plus diagonal form, with the required shift elements λ_k .

Let us point out one more thing about the above reduction algorithm. In principle, one should be cautious about the validity of the structure propagation mechanism for the trivially satisfied structure block \mathcal{B} in the top left matrix corner, since this structure block involves the first row of the matrix, so that the result of Theorem 9 does *not* hold anymore. Still, the structure propagation argument *does* remain valid, by the fact that, by construction, the applied Givens transformations created zeros in the first column of the *pure* component A_{pure} , rather than A itself, and this was precisely the necessary condition needed in the first paragraph of the proof of Theorem 9.

REMARK 11.

1. *The presence of rank structure in the upper triangular part was left out from the above discussion, but can be handled similarly as before. Let us assume e.g. that the matrix is Hermitian. Then we will suffice with having access over the Givens-weight representation of the lower triangular part, having now a ‘bulge’ at each place where the structure goes beyond the main diagonal. Let us consider e.g. the situation in Figure 4.6(b). We need here to have access over the three top elements lying in between the two thick vertical lines. Each of these elements could then be computed by the Hermitian character, using the knowledge of the corresponding element in the lower triangular part, which can be obtained by means of some auxiliary spreading-out operations of the corresponding column of the lower triangular part (one should use an auxiliary variable for doing this). Let us note that these spreading-out operations become more expensive when the required element is situated further from the main diagonal.*
2. *The Hessenberg reduction algorithm can be seen as a special case of the lower semiseparable plus diagonal reduction algorithm, provided that one chooses $\lambda_k = \infty$ for each k , in the sense described in [5].*
3. *When the required diagonal plus semiseparable structure is close to Hessenberg form, i.e. $\lambda_k \approx \infty$, it can be expected that numerical problems may arise due to large values of the λ_k . To solve this problem, we may observe that the only place where information about the shift elements is needed, is for the determination of each top Givens transformation $G_{1,2}$. Once this has been done, the application of this and other Givens transformations can be applied in a ‘shift-free’ way, by working with the induced pure structure blocks, lying just below the main diagonal. Our numerical experiments indicate that this shift-free variant is indeed more stable than the variant described above, even for moderate sizes of the shift elements λ_k .*

5. 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

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

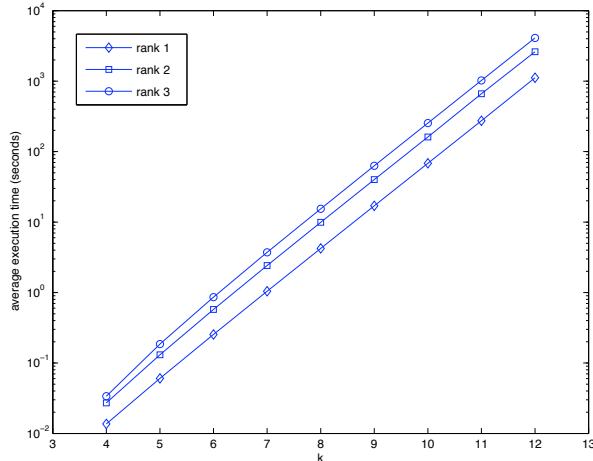


FIGURE 5.1. Average execution time for 5 random samples of size $n = 2^k$ and rank $r = 1, 2, 3$.

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 $k = 4, \dots, 12$. 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.

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 5.1 shows for each size $n = 2^k$ and each rank index r the execution time $T_{k,r}$ averaged over the 5 samples of performing the reduction to Hessenberg form.

To check that the computational complexity is quadratic in the size of the matrix, Figure 5.2 shows the fraction $T_{k+1,r}/T_{k,r}$ averaged over the 5 samples and over the ranks $r = 1, 2, 3$.

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 5.3 shows the relative 2-norm of the error averaged over the 5 samples and the rank indices $r = 1, 2, 3$ [†].

The computation of Givens transformations during the algorithm was performed according to the implementation described in [8]. We note that the accuracy of the

[†]The eigenvalues of the resulting tridiagonal matrix at the end of the Hessenberg reduction process were computed using the matlab routine `trideig` due to P.-O. Persson. This routine is available at <http://www.mit.edu/~persson/mltrid/index.html>.

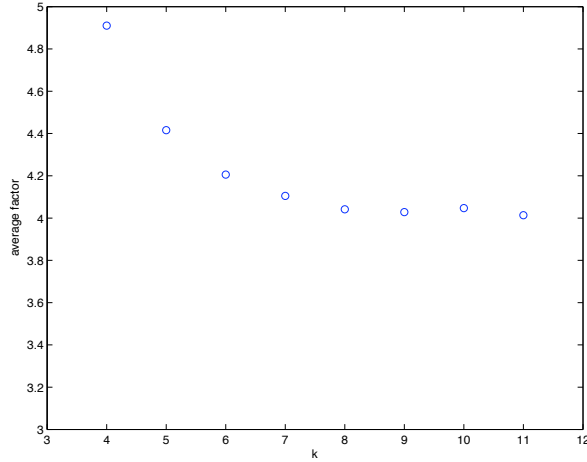


FIGURE 5.2. Fraction $T_{k+1,r}/T_{k,r}$ averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of the size $n = 2^k$.

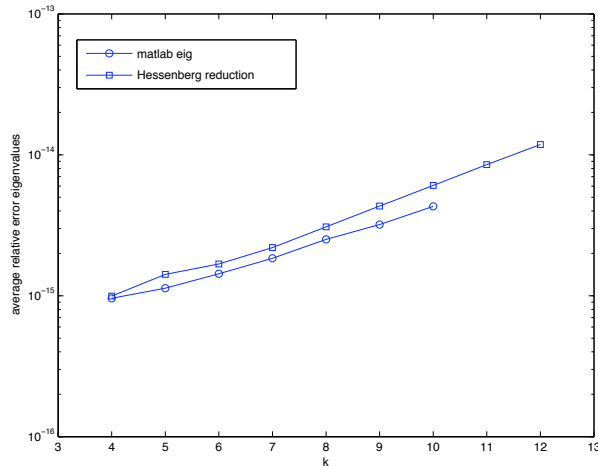


FIGURE 5.3. Relative 2-norm of the error of the eigenvalues averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of the size $n = 2^k$.

algorithm turns out to be slightly sensible to the choice of the used Givens routine, but this effect is rather modest.

The pull-through lemma was implemented in the trivial way. This means that, using the notations of Lemma 6, we explicitly computed the 3 by 3 matrix $G'_{1,2}G_{2,3}G_{1,2}$ in its full form. Subsequently we computed Givens transformations such that

$$(\tilde{G}_{2,3})^H (\tilde{G}_{1,2})^H (\tilde{G}'_{2,3})^H G'_{1,2} G_{2,3} G_{1,2} \quad (5.1)$$

is upper triangular, with positive diagonal elements. Hence (5.1) is a unitary, upper triangular matrix with positive diagonal elements, so that this must be the identity matrix. Rewriting this fact leads to the desired refactorization of Lemma 6. We note that our future work will describe in more detail the implementation of the pull-through lemma, including a non-trivial speed-up which was already implicit in

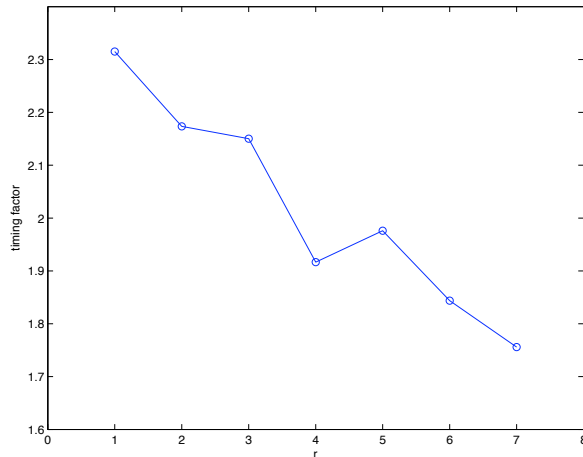


FIGURE 5.4. Fraction T_{2^r}/T_r for size $n = 2^9$ in function of the rank index $r = 2^l$ with $l = 0, 1, \dots, 6$.

[9].

To check that the computational complexity is linear as a function of the rank index r , we considered the execution time T_r for matrices of fixed size $n = 2^9 = 512$ and varying rank index $r = 2^l$ with $l = 0, 1, \dots, 7$. Figure 5.4 gives the fraction T_{2^r}/T_r for subsequent rank indices. Note that the fraction tends to approximate 2 for large rank indices r . In fact, the fraction tends to go even below this value of 2; but we note that this is artificial since for high ranks, the ranks become relatively large w.r.t. the size $n = 2^9$, 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.

6. Conclusion. In this paper we described an algorithm for performing the Hessenberg reduction of rank structured matrices. Numerical experiments indicated that this approach leads to a stable computation of the eigenvalues of the given matrix. We showed that the algorithm has complexity $O((r + s)n^2)$ in case of Hermitian plus low rank matrices. We explained that this complexity holds also for unitary plus low rank matrices, provided that one agrees to perform numerical approximations during the algorithm. Our future work includes an alternative algorithm for the Hessenberg reduction of unitary and related matrices, using an appropriate representation.

REFERENCES

- [1] S. Chandrasekaran and M. Gu. Fast and stable eigendecomposition of symmetric banded plus semi-separable matrices. *Linear Algebra Appl.*, 313:107–114, 2000.
- [2] S. Delvaux and M. Van Barel. Structures preserved by the QR-algorithm. *J. Comput. Appl. Math.*, 187(1):29–40, 2005. DOI 10.1016/j.cam.2005.03.028.
- [3] 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.
- [4] S. Delvaux and M. Van Barel. Rank structures preserved by the QR-algorithm: the singular case. *J. Comput. Appl. Math.*, 189:157–178, 2006.

- [5] S. Delvaux and M. Van Barel. Structures preserved by matrix inversion. *SIAM J. Matrix Anal. Appl.*, 28(1):213–228, 2006.
- [6] D. Fasino, N. Mastronardi, and M. Van Barel. Fast and stable algorithms for reducing diagonal plus semiseparable matrices to tridiagonal and bidiagonal form. *Contemp. Math.*, 323:105–118, 2003.
- [7] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal.*, 2:205–224, 1965.
- [8] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.
- [9] W. B. Gragg. The QR algorithm for unitary Hessenberg matrices. *J. Comput. Appl. Math.*, 16:1–8, 1986.
- [10] N. Mastronardi, S. Chandrasekaran, and S. Van Huffel. Fast and stable algorithms for reducing diagonal plus semiseparable matrices to tridiagonal and bidiagonal form. *BIT*, 41(1):149–157, 2003.
- [11] B. N. Parlett. *The Symmetric Eigenvalue Problem*, volume 20 of *Classics in Applied Mathematics*. SIAM, Philadelphia, 1998.
- [12] H. Rutishauser. On Jacobi rotation patterns. In *Proceedings of Symposia in Applied Mathematics, vol. 15, Experimental Arithmetic, High Speed Computing and Mathematics*, pages 219–239, Providence, 1963. Amer. Math. Society.
- [13] H. R. Schwartz. Tridiagonalization of a symmetric band matrix. *Numer. Math.*, 12:231–241, 1968.
- [14] R. Vandebril, E. Van Camp, M. Van Barel, and N. Mastronardi. Orthogonal similarity transformation of a symmetric matrix into a diagonal-plus-semiseparable one with free choice of the diagonal. *Numer. Math.*, 102:709–726, 2006.
- [15] J. H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, 1965.