

A Givens-weight representation for rank structured matrices

Steven Delvaux Marc Van Barel

Report TW453, March 2006



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

A Givens-weight representation for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW453, March 2006

Department of Computer Science, K.U.Leuven

Abstract

In this paper we introduce a Givens-weight representation for rank structured matrices, where the rank structure is defined by certain low rank submatrices starting from the bottom left matrix corner. This representation will be compared to the (block) quasiseparable representations occurring in the literature. We will then provide some basic algorithms for the Givens-weight representation, in particular showing how to obtain a Givens-weight representation for a full matrix, and how to reduce the order of the representation, whenever appropriate. We will also show how to update the representation under the action of Givens transformations, and how to compute matrix-vector products. As such, these results will be the basis for the algorithms on Givens-weight representations to be described in subsequent papers.

Keywords : rank structured matrices, representations, numerical approximation, Givens transformations, matrix-vector multiplication.

AMS(MOS) Classification : Primary : 65F, Secondary : 65F30, 15A03.

A GIVENS-WEIGHT REPRESENTATION FOR RANK STRUCTURED MATRICES

STEVEN DELVAUX*, MARC VAN BAREL*

Abstract. In this paper we introduce a Givens-weight representation for rank structured matrices, where the rank structure is defined by certain low rank submatrices starting from the bottom left matrix corner. This representation will be compared to the (block) quasiseparable representations occurring in the literature. We will then provide some basic algorithms for the Givens-weight representation, in particular showing how to obtain a Givens-weight representation for a full matrix, and how to reduce the order of the representation, whenever appropriate. We will also show how to update the representation under the action of Givens transformations, and how to compute matrix-vector products. As such, these results will be the basis for the algorithms on Givens-weight representations to be described in subsequent papers.

Keywords: rank structured matrices, representations, numerical approximation, Givens transformations, matrix-vector multiplication.

AMS subject classifications: 65F, 65F30, 15A03.

1. Introduction. In this paper we describe a way to obtain compact representations for rank structured matrices. More specifically, these will be the unitary-weight and Givens-weight representations, in increasing order of specification.

First we must define the class of matrices for which our representations will be appropriate.

DEFINITION 1. (See [3]:) 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 [3]. 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. An example of a rank structure is shown in Figure 1.1.

*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.

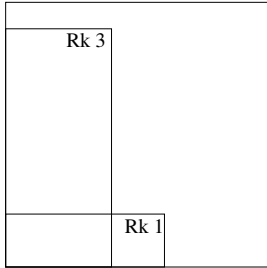


FIGURE 1.1. Example of a rank structure with two structure blocks \mathcal{B}_1 and \mathcal{B}_2 . The notation $\text{Rk } r$ denotes that the structure block is of rank at most r .

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.

It is easy to see that each structure block induces certain connections between the elements of the matrix. Hence it should be possible to represent a rank structured matrix using only a small number of parameters. Although one can devise several ways to obtain such a representation, there seem to be two classes of representations frequently used in the literature, which we call here *uv-representations* and *block quasiseparable representations*. Let us give a brief survey.

The class of *uv-representations* was historically the first, see e.g. [10]. We mention that this type of representation is only possible under certain conditions.

The different and more flexible class of block quasiseparable representations has been introduced in the book [6]. Starting from this book, many algorithms for these representations have been developed in the literature. They appear in the work of Dewilde and van der Veen: see e.g. [6, 7]. They were then used by Eidelman, Gohberg et al., who also introduced the name (block) ‘quasiseparable representation’: see e.g. [8, 9]. More recently, these matrices appear under the name of ‘sequentially semiseparable representations’ in the work of Chandrasekaran, Gu et al.: see e.g. [1]. Finally, we note that rank structured matrices appear also in a purely theoretical context in the work of Tyrtyshnikov [12], under the name of ‘weakly semiseparable matrices’.

We are not intending to give here a complete overview of all the existing algorithms for block quasiseparable representations: this would be a task which is even more complicated by the sometimes varying conditions under which these algorithms are derived, most notably the fact that the underlying structure blocks must be situated just below the main diagonal, are equidistant and so on, with the precise conditions depending sometimes from paper to paper. Instead we will compare our own algorithms with those in the literature at the appropriate places in the remaining of this and following papers. We refer also to Section 6 for a treatment of *uv-* and block quasiseparable representations.

In the special case of *semiseparable* matrices of semiseparability rank one, the latter being defined by a collection of structure blocks $\mathcal{B}_k = (k, k, 1)$, $k = 1, \dots, n$ on $\mathbb{C}^{n \times n}$, an alternative representation, the so-called Givens-vector representation has been introduced in [15]. A first step in the generalization of this representation has been taken in [13]. In the current paper we will carry this scheme one step further, by generalizing the idea of the Givens-vector representation to be able to represent *any*

rank structure. Moreover, in this and subsequent papers we will show how to develop algorithms for a variety of problems involving the Givens-weight representation.

This paper is organized as follows. Section 2 introduces the basic ideas of the Givens-weight representation. Section 3 considers the operations of approximating and reducing the Givens-weight representation. Section 4 describes an important advantage of Givens-weight representations, namely the fact that the representation can be easily updated under the action of Givens transformations. Section 5 describes how to perform matrix-vector products with the Givens-weight representation. Finally, Section 6 contains a brief comparison of the Givens-weight representation with block quasiseparable and uv -representations.

2. Givens-weight representations. In this section we will describe the basic ideas to obtain a compact representation for rank structured matrices. The representation will generalize the Givens-vector representation for semiseparable matrices of semiseparability rank one which was introduced in [15].

The Givens-weight representation will be an *internal* representation, which works strictly inside the area spanned by the structure blocks, and considers the ‘outside world’ to be inaccessible. In the next two subsections, we introduce the concepts of unitary-weight and Givens-weight representations, first on a special case, and later for general rank structures. In the last two subsections we will then further complete the description of the representation, by specifying two directions in which it can be extended: we describe representations that are based on column instead of row operations, and for the upper instead of the lower triangular matrix part. But let us start now with the basic ideas.

2.1. Example. In the present subsection we will try to indicate the underlying ideas of unitary-weight representations. To this end we will take the structure in Figure 2.1 as a didactical example. First, it may be noted that this figure does not show the surrounding matrix box anymore: this reflects a fact mentioned before, namely that only the area spanned by the structure blocks will be relevant for the representation, and that the ‘outside world’ will be inaccessible.

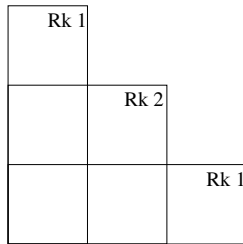


FIGURE 2.1. Example of a rank structure with three structure blocks $\mathcal{B}_1, \mathcal{B}_2$ and \mathcal{B}_3 . We will use this example to explain the mechanism of the unitary-weight and Givens-weight representation during the following paragraphs. From now on the surrounding matrix box, as in Figure 1.1, will not be shown anymore.

The unitary-weight representation is obtained by reducing the structure blocks into blocks of zeros, by the use of unitary row transformations. First we apply a unitary transformation to transform the bottom Rk 1 block into a block of zeros, with one row less: see Figure 2.2.

Having applied this operation, note that in columns 7, 8 and 9 we have already reached the ‘top’ of the structure. Therefore, this is now the right moment to consider the top elements of these columns, and to store them. These elements will be called *weights*, and they are visualized on a grey background in Figure 2.2.

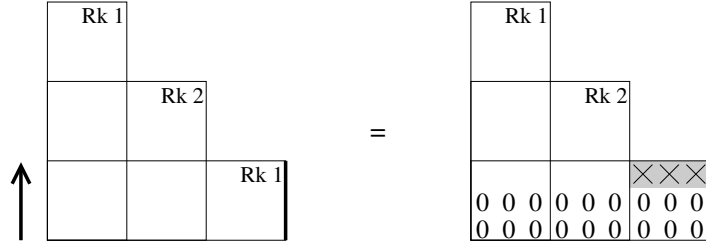


FIGURE 2.2. We apply a unitary transformation to transform the bottom two rows of the structure into zeros. This transformation acts only on the left of the vertical line which is indicated in boldface in the figure: we say that this line borders the action radius of the unitary transformation. Having performed this unitary transformation, the elements standing on a grey background are stored: they are called weights.

From now on we consider columns 7, 8 and 9 as finished, and we restrict our perspective to the previous columns. We can then apply a unitary transformation to transform the middle Rk 2 block into a block of zeros, with two rows less: see Figure 2.3.

Having applied this operation, note that also in columns 4, 5 and 6 we have reached the top of the structure. Therefore, this is now the right moment to consider the top elements of these columns, and to store them. This yields us a second block of weights, which is again visualized on a grey background in Figure 2.3.

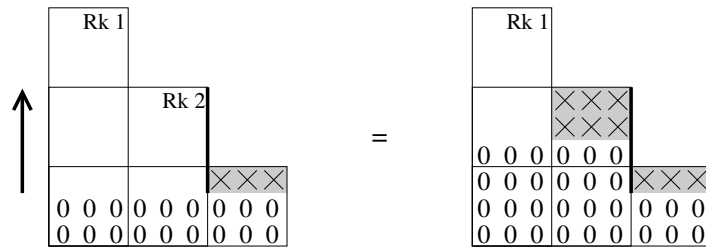


FIGURE 2.3. We apply the next unitary transformation, and store the new block of weights.

From now on we drop columns 4, 5 and 6 from our perspective. We can then apply a unitary transformation to transform the top Rk 1 block into a block of zeros, with one row less: see Figure 2.4. We conclude by storing the final block of weights.

The weights can now be collected into a single matrix, which we call the *weight matrix*. Together with the computed unitary transformations, this matrix yields us the complete *unitary-weight representation* of the given matrix: see Figure 2.5.

Of course, to be a good representation, the unitary-weight representation should allow the possibility to restore the original matrix which we started from. This can be done by ‘reversing’ the above steps.

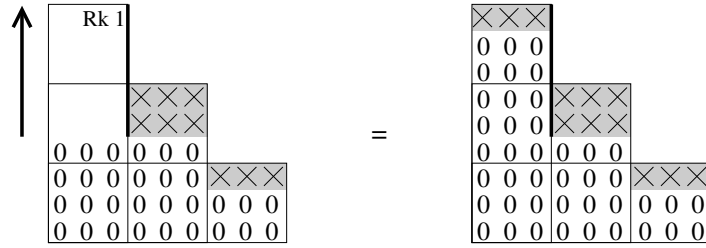


FIGURE 2.4. We apply the final unitary transformation, and store the new block of weights.

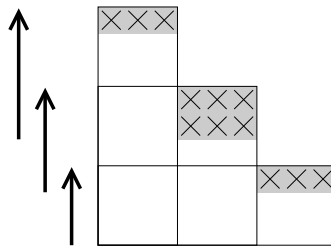


FIGURE 2.5. Schematic picture of the unitary-weight representation for the rank structure in Figure 2.1.

This reversal process will be called *spreading out* the unitary-weight representation, and it is shown in Figure 2.6. First, we start by spreading out the top block of weights, i.e. we multiply them with the inverse of the top unitary transformation of the unitary-weight representation: see Figure 2.6(a).

As a result of this spreading-out, the weight matrix will start to get filled-in. Moreover, since we are now at the point of entering the structure in columns 4, 5 and 6, this is now the right moment to bring also the middle block of weights into our perspective. Then we spread out all of these elements: see Figure 2.6(b).

As a result of this spreading-out, the weight matrix will get filled-in more and more. Moreover, since we are now at the point of entering the structure in columns 7, 8 and 9, this is now the right moment to bring also the last block of weights into our perspective. Then we spread out all of these elements: see Figure 2.6(c).

Finally we retrieve then the original, full matrix which we started from: see Figure 2.6(d).

The reader will have noticed that we used a grey/white colour code in Figure 2.6, as well as in the previous figures. Let us explain the meaning of this code. The grey elements are used to denote the *weights*, i.e. the elements that contain ‘condensed’ information about the full matrix. Thus in order to see the real meaning of these elements, they first have to be spread out by the next unitary transformations of the unitary-weight representation. On the other hand, the white elements denote the *real-size elements*, i.e. the elements that will not be influenced anymore by the next unitary transformations. These are actual elements of the full matrix.

This grey/white code turns out to be quite handy, and therefore it will be frequently used in what follows.

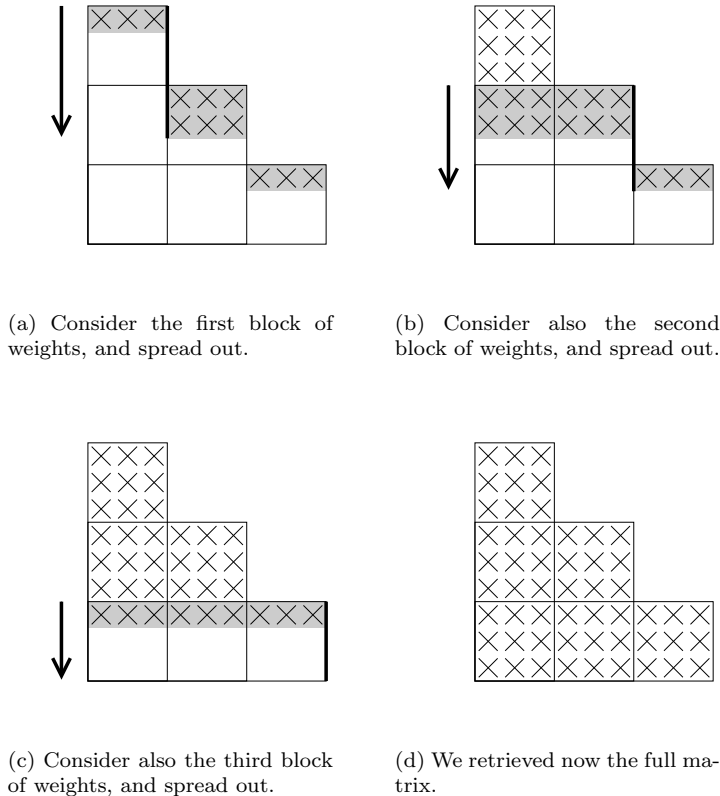


FIGURE 2.6. *Spreading out the unitary-weight representation.*

2.2. General definitions. Now we are ready to handle the unitary-weight and Givens-weight representations in the general case. We will do this for a matrix A satisfying some general rank structure $\mathcal{R} = \{\mathcal{B}_k\}$. Just as in the example of the previous subsection, we will have to assume that the structure blocks are ordered in a *sequential* way, i.e. that there is no pair of structure blocks for which the first one is completely contained in the second one. If this condition is not satisfied yet, then we first have to remove from the structure all such structure blocks which are completely contained in another structure block. (Actually, these ‘internal’ structure blocks are not completely useless, in the sense that they lead to an additional sparsity pattern in the Givens-weight representation. But we will not be concerned about this here).

Thus we can assume now that the rank structure does not contain any ‘internal’ structure blocks anymore. We can then order the remaining structure blocks in a sequential way, going from the top left to the bottom right corner of the matrix. Stated in another way, the structure blocks are ordered in such a way that both the row and column indices i_k and j_k of the structure blocks increase in a strictly monotonic way.

Then we can come to the general definition of unitary-weight representations.

DEFINITION 2. (*Index sets:*) Let $\mathcal{R} = \{\mathcal{B}_k\}_{k=1}^K$ be a rank structure, where the structure blocks are ordered from top left to bottom right. Then we define index sets

$I_k = \{i_k, \dots, i_{k+1} - 1\}$, $I_{k,\text{top}} = \{i_k, \dots, i_k + r_k - 1\}$, and $J_k = \{j_{k-1} + 1, \dots, j_k\}$, for $k = 1, \dots, K$. Here the extremal values of row and column indices are defined as $i_{K+1} := N + 1$ and $j_0 := 0$, and we define also $r_{K+1} := 0$.

DEFINITION 3. (*Unitary-weight representation:*) 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 $(\{U_k\}_{k=1}^K, W)$. Each U_k is a unitary transformation, acting on the rows and columns indexed by $I_k \cup I_{k+1,\text{top}}$ and $\bigcup_{l=1}^k J_l$, respectively, and intended to create zeros in all these rows, except those of $I_{k,\text{top}}$. These unitary transformations U_k should be applied subsequently for $k = K, \dots, 1$. On the other hand, the matrix $W \in \mathbb{C}^{m \times n}$ is called the weight matrix, and it contains all the blocks of elements obtained in the rows and columns indexed by $I_{k,\text{top}}$ and J_k , respectively, at the moment just after applying U_k . See Figure 2.7.

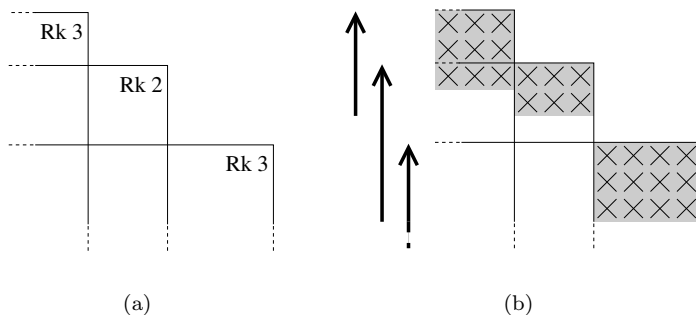


FIGURE 2.7. For the rank structure in the left picture, the right figure shows a schematic picture of the unitary-weight representation.

If a unitary-weight representation of a matrix is given, then we can restore the full matrix by *spreading out* the representation. We explained this already in the previous subsection.

Now we will imply some extra constraints to the above definition of unitary-weight representations, by additionally splitting each unitary transformation U_k into a product of Givens transformations. This will lead us to the description of *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 be sometimes denoted as $G_{i,i+1}$, and the index i will be called the *row index* of the Givens transformation.

First, 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 2.8.

Having introduced all these notions, we can now specify from unitary-weight to Givens-weight representations.

DEFINITION 4. (*Givens-weight representation:*) 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

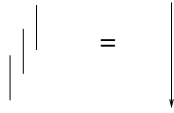


FIGURE 2.8. A Givens-arrow consisting of 3 Givens transformations. Concerning this figure, we recall the reader that we consider each Givens transformation as ‘acting’ on the rows of an (invisible) matrix standing on the right of it, and hence that the Givens transformations in the figure should be evaluated from right to left, hereby explaining the downward direction of the Givens arrow.

\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 2.9.

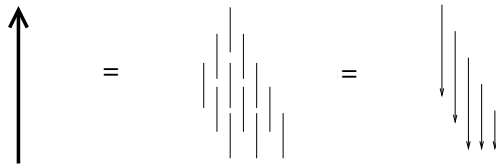


FIGURE 2.9. Suppose that the current structure block is $\text{Rk } 3$, and that the corresponding unitary transformation U_k spans over 6 rows. Then we assume for this unitary transformation a decomposition into a product of Givens arrows of width at most 3.

Of course we should explain why the assumption is made that each Givens arrow in the decomposition of U_k has width at most r_k . But this is a logical assumption: the reader should 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, and 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: see Section 3 for more details.

Note that by decomposing each unitary transformation U_k as specified in Definition 4, 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. This overlap may be especially severe when the structure is *dense*, i.e. when the vertical gaps between two subsequent structure blocks are small compared to their rank upper bounds: see Figure 2.11.

In practical situations we would like to avoid such a superfluous amount of Givens transformations. We will describe now some extreme cases where this is achieved.

DEFINITION 5. (*Canonical Givens-weight representation:*) Under the same conditions as in Definition 4, 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. If the same condition holds with the word ‘tails’ replaced by ‘tops’, then the representation is said

to be canonical of type 2*.

It turns out that *any* Givens-weight representation can be brought into each of the canonical forms of the above definition. This reduction can be achieved by means of so-called ‘pull-through techniques’. The following lemma is pivotal in this respect.

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 2.10.

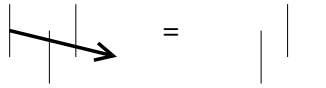


FIGURE 2.10. *Pull-through lemma applied in the downward direction. One could imagine that the leftmost Givens transformation is really ‘pulled through’ the two rightmost Givens transformations.*

The above version of the pull-through lemma was formulated in the downward direction. In a similar way there exists a pull-through lemma in the *upward* direction, but we will not be concerned about this here.

As an example, let us apply the pull-through lemma in order to reduce the representation in figure 2.11. The resulting situation is then shown in Figure 2.12. Note that the superfluous Givens transformations have been removed by the pull-through process, and that the representation has become canonical of type 1. Also the canonical representation of type 2 can be obtained in this way, namely by applying the pull-through lemma in the *upward* direction.

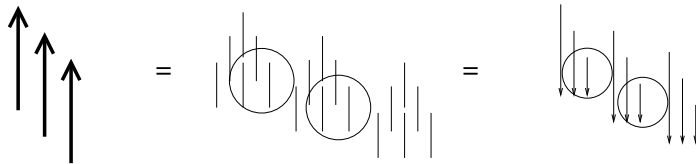


FIGURE 2.11. *Decomposition into a superfluous amount of Givens transformations, in case of a dense rank structure \mathcal{R} .*

Note that for efficiency reasons, the pull-through operations should be grouped in such a way that we move only one time from top to bottom of the matrix. Moreover, we should also *enlarge* each time the action radius of the Givens transformations which will be pulled-through, in order to give them the same action radius as the next unitary operation U_k . (See Section 4 for similar manipulations with the action radius). This implies that also the weight matrix of the Givens-weight representation will be involved in the pull-through process.

*We may point out that under some additional conditions, the canonical representations of type 1 and 2 have an interpretation in terms of ‘zero-creating’ and ‘rank-decreasing’ Givens patterns, respectively, to be discussed in a further paper. We should mention that the zero-creating Givens pattern has also been discussed in [5], although it was done there in a more theoretical context.

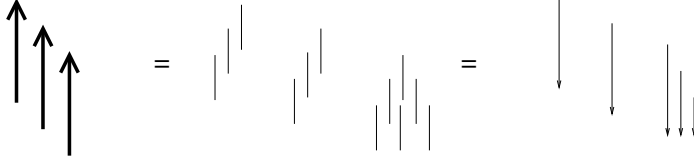


FIGURE 2.12. After removal of the superfluous Givens transformations in Figure 2.11, for example by the pull-through lemma, we end up with a canonical Givens-weight representation of type 1. Although this reduction process can always be applied, the reader should keep in mind that the occurrence and/or removal of a huge amount of superfluous Givens transformations should preferably be avoided due to numerical inefficiency.

In a certain sense, the canonical representations of Definition 5 represent two extreme cases having a small amount of Givens transformations. For practical computations we can afford to work with a more general class of Givens-weight representations which we call *efficient*.

DEFINITION 7. (*Efficient Givens-weight representations*): Under the same conditions as in Definition 4, let there be given a Givens-weight representation for $A \in \mathbb{C}^{m \times n}$ according to the rank structure \mathcal{R} . This representation is said to be *efficient* if its number of Givens transformations exceeds the number of Givens transformations occurring in the canonical Givens-weight representation of type 1 (say), by at most a certain, fixed percentage. For practical choices of the underlying rank structure \mathcal{R} , i.e. when $m \approx n$ and the unstructured lower triangular part is small compared to the structured lower triangular part, this means that the number of Givens transformations should be approximately equal to rn , where r is a measure for the average semiseparability rank.

This definition says that a Givens-weight representation can only be efficient if the number of superfluous Givens transformations is not too high w.r.t. the ‘optimal’ value. As an illustrative example, the reader could keep in mind the difference between Figures 2.11 and 2.12 above.

Summarizing, we have now completely described the Givens-weight representation induced by a rank structure \mathcal{R} and some of its practical variants. For the remaining part of this section, we will indicate how this entire terminology can be reformulated in two other situations.

2.3. Row versus column operations: swapping the representation. The unitary-weight and Givens-weight representations described above have the feature that the unitary operations are considered as *row* operations. In a completely analogous way, one could build a representation based on *column* operations.

We are not intending to reproduce all the above definitions and examples to the column case. Instead, we will focus here on an efficient *swapping algorithm* to change from a row to a column representation or vice versa, hereby generalizing the swapping algorithm for semiseparable matrices of semiseparability rank 1 introduced in [14].

The swapping algorithm will be fairly simple and, together with its generalization to ‘generalized swapping’ to be described further on, it will form a key ingredient for manipulating Givens-weight representations. Moreover, in Section 3 we will come to a possible application of the swapping algorithm, by showing that it can be used to perform an additional compressing of the representation, i.e. to approximate the matrix by one having smaller ranks in its underlying rank structure, whenever appropriate.

The swapping process is illustrated in Figure 2.13.

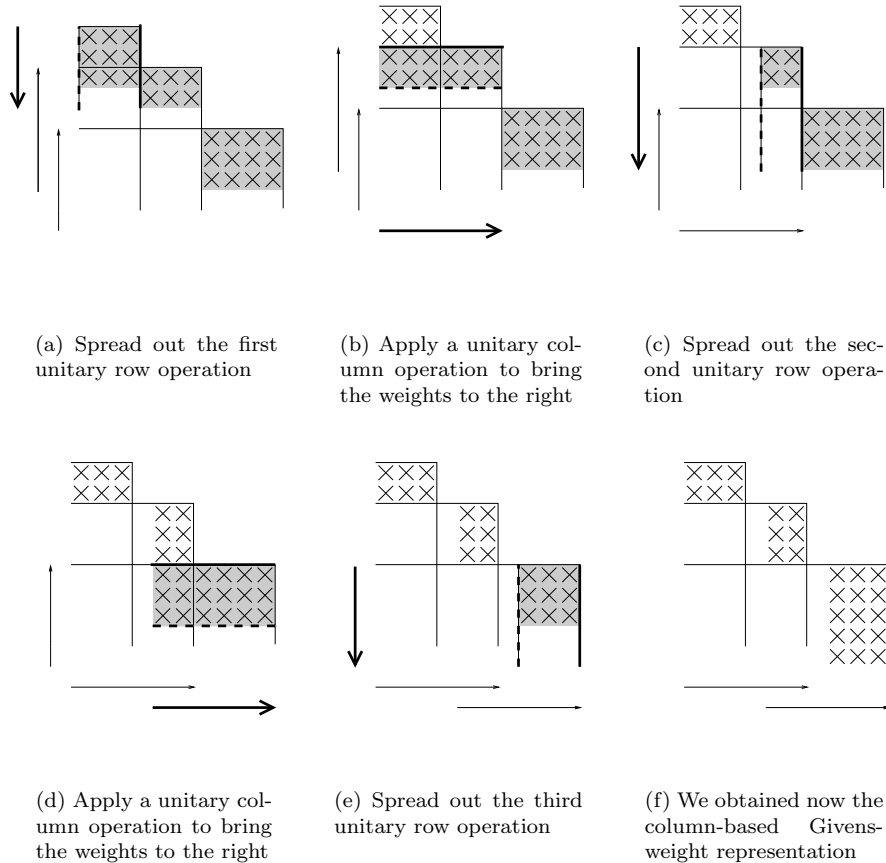


FIGURE 2.13. *Swapping the representation.*

Let us comment on this figure. Figure 2.13(a) shows the initial weight matrix, in which we have started to spread out the Givens transformations belonging to the first unitary transformation. We would then like to go on by spreading out also the next unitary operations, so that we finally obtain the full version of the given rank structured matrix.

Before spreading out further, however, we apply an auxiliary column operation in order to bring the weights as much as possible to the right: see Figure 2.13(b). This auxiliary column operation serves to prevent that the matrix would get completely filled-in by the spreading-out process. It is applied only between the thick horizontal lines shown in Figure 2.13(b).

Having done this, we can now go on to further spread out the rank structured matrix, and applying each time a unitary column operation to bring the weights as much as possible to the right. Figures 2.13(c), 2.13(d), 2.13(e) and 2.13(f) show the next steps in this swapping process. In particular, the final situation in Figure 2.13(f) shows the situation where we have completely spread out the weight matrix, by annihilating the action of the original row operations, and where at the same time the weight matrix has again been compressed by the use of auxiliary column operations. In other words, we have now completely switched from a row-based to a column-based

Givens-weight representation.

Note that also the weight blocks of the column representation can easily be read off during the swapping algorithm.

For this algorithm, as well as for many other algorithms to be described, the algorithm was illustrated for a unitary-weight rather than a Givens-weight representation. But this is only for clarity reasons; in reality the above algorithm is capable of both exploiting and preserving the sparsity pattern satisfied by the Givens transformations. The main observation for this purpose is that by definition, the Givens transformations are organized in *Givens arrows*, each of them pointing in the opposite direction w.r.t. the algorithm flow: see Definition 4. This will guarantee that there is no superfluous fill-in during the algorithm, in the sense that, loosely speaking, each Givens transformation will cause only *one* weight element to get filled in. The latter fact will then guarantee that the number of swapped Givens transformations is of the same order as the number of original Givens transformations: see Figure 2.14.

Let us consider the numerical complexity of the swapping algorithm. To this end we can first note that each Givens transformation on both rows and columns acts on a number of approximately r weight elements, where r is a measure for the average semiseparability rank of the rank structure \mathcal{R} . In particular, if the Givens-weight representation is efficient in the sense of Definition 7, and assuming that we work with a practical choice of rank structure, it follows that the complexity reduces to $O(r^2n)$.

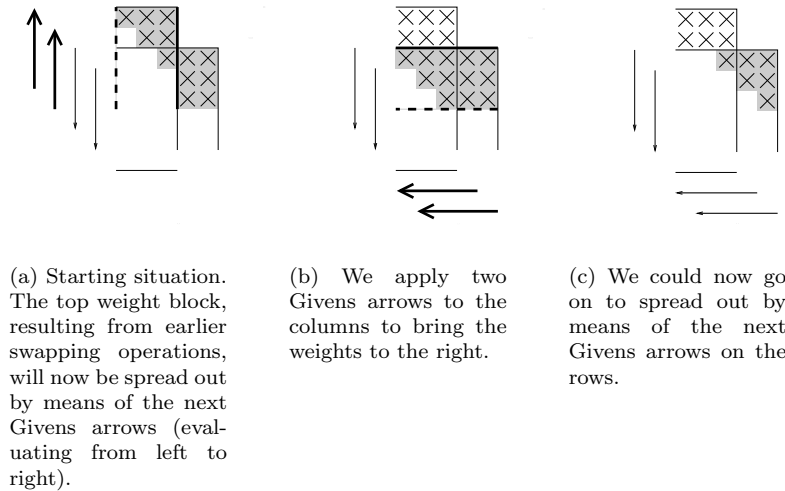


FIGURE 2.14. The figure shows a more detailed illustration of the swapping operation, in terms of individual Givens arrows. Note that the sparsity of the weight block in Figure 2.14(a) has been preserved during the algorithm, in the sense that the weight block in Figure 2.14(c) satisfies the same sparsity. Moreover, this sparsity allows the number of swapped Givens transformations determined in Figure 2.14(b), to be of the same order as the number of original Givens transformations.

2.4. Representation for the upper triangular part. We will now focus on a second way to extend the definition of Givens-weight representation. Until now we have only considered rank structures in the *lower* triangular part. But it is not difficult to generalize these notions to a Givens-weight representation for the structured *upper* triangular part. Formally speaking, we define a rank structure in the block upper

triangular part[†] as a collection \mathcal{R}^T of structure blocks satisfied by the transpose matrix A^T . It is then straightforward to generalize the notion of Givens-weight representation to this case.

Note that if the matrix satisfies rank structure in both its block lower and block upper triangular part, we will obtain in this way a representation for the *full* matrix, as we describe now.

DEFINITION 8. *Let $A \in \mathbb{C}^{m \times n}$ be a matrix having rank structure both in its block lower and in its block upper triangular part, and suppose that the structure blocks of the lower and upper triangular parts do not overlap with each other. Let there be given*

- *a Givens-weight representation for the structured lower triangular part of A ,*
- *a Givens-weight representation for the structured upper triangular part of A ,*
and
- *a few real-size elements to represent the unstructured matrix part.*

This collection will be briefly denoted as a Givens-weight representation for the matrix A . In the same way, let us consider the matrix obtained by gluing together the weight matrices of the representations for the lower and the upper triangular part, and the sparse matrix containing the unstructured matrix elements around the main diagonal. This matrix will be denoted as the weight matrix of the representation: see Figure 2.15.

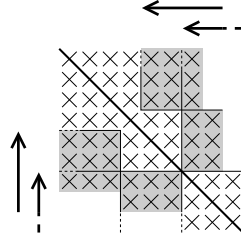


FIGURE 2.15. *Schematic picture of a Givens-weight representation for a matrix A having rank structure both in its structured lower and in its structured upper triangular part. The figure shows the weight matrix as well as the unitary transformations of the representation. Note that the matrix is assumed here to be symmetric, and that the lower and upper triangular representations are based on row and column operations, respectively.*

Note that both the upper and the lower triangular part could be represented by either a row-based or a column-based Givens-weight representation, or even ‘hybrid’ versions of this, leading to a total of at least $2 \times 2 = 4$ different ways of representing the matrix A . The example in Figure 2.15 shows just one example of such an assignment.

Summarizing, we have now described the basic concepts of Givens-weight representations for rank structured matrices. In the remainder of this paper, as well as in the next papers, we will move to the development of practical algorithms for this representation.

3. Approximating and building the representation. In this section we show how a Givens-weight representation can be built to approximate a full matrix, and how a Givens-weight representation can be approximated to have lower ranks in

[†]The precise meaning of the term ‘block upper triangular part’, or its synonym ‘structured upper triangular part’ depends of course on the underlying distribution of the structure blocks. In particular, the structure blocks are allowed to go beyond the main diagonal. The same remarks hold of course for the block *lower* triangular part.

its underlying rank structure. It will suffice to describe these operations for a Givens-weight representation of the block lower triangular part; the block upper triangular part can then be handled in a similar way.

3.1. Approximating a full matrix. We start with building a Givens-weight representation to approximate the block lower triangular part of a full matrix. The basic ideas of this process have already been given in Section 2, where it was explained how the representation requires at the k th step the determination of a unitary transformation U_k that compresses a certain $\text{Rk}(r_k)$ matrix, except for its top r_k rows. Moreover, it was explained there how the concept of Givens-weight representation requires this unitary operation U_k to be decomposed in a certain way into a product of Givens arrows of width at most r_k : see Figure 2.9.

To achieve this in a practical way, we will first consider the case where the required $\text{Rk}(r_k)$ matrix has been decomposed in the form of a truncated singular value decomposition, or more generally any rank-revealing factorization GH^H with G and H both having precisely r_k columns. The required unitary transformation U_k can then be determined by simply applying Givens arrows to compress the generator G ; in this way the decomposition into Givens arrows will have precisely the form required in Figure 2.9.

We may mention that this scheme can be improved to obtain a more compact form for the Givens arrows. To this end we first apply an auxiliary unitary *column* operation to the generator G in order to bring its bottom square submatrix in upper triangular form. Having done this, the number of Givens arrows required to compress the generator G except for its top r_k rows can be considerably lower. Having computed these Givens arrows, we can again remove the influence of the auxiliary column operations by multiplying with their inverses to the columns, and we can then go on with the next structure block.

It can be shown that with a proper implementation of this scheme, each of the applied Givens arrows will *definitively* eliminate a row, which will then not be touched anymore by any of the following operations. In other words, this means that the tops of the subsequent Givens arrows will be strictly monotonically proceeding upwards, and hence that we will have obtained a canonical Givens-weight representation of type 2.

Consider now the case where the given $\text{Rk}(r_k)$ matrix is compressed by means of a rank-revealing QR-factorization [11]. We recall that this procedure consists in searching for the column with largest norm and bringing it in upper triangular form. We can then remove the top row from our perspective, and apply the same procedure to the remaining matrix. This procedure is then repeated until the remaining matrix is numerically zero, which will be the case after at most r_k steps. The remaining, numerically zero elements are simply truncated to zero.

Now it is straightforward to see that the truncated rank-revealing QR-factorization will lead to a compressing of the given $\text{Rk}(r_k)$ matrix except for its top r_k rows. The required unitary transformation U_k is then obtained by means of the applied Givens transformations of the rank-revealing QR-factorization, more precisely by rearranging them into Givens arrows of width r_k .

Just as in the previous paragraphs, it is possible to improve this scheme in order to obtain a more compact representation. This follows by remarking that the rank-revealing QR-factorization induces in a natural way a factorization GH^H , where H^H contains the top r_k rows of the compressed $\text{Rk}(r_k)$ matrix, and the matrix G contains the first r_k columns of the unitary matrix formed by the successive Givens transfor-

mations. We can then proceed as described in the previous paragraphs to obtain a canonical Givens-weight representation of type 2.

REMARK 9. *The above discussion assumed that the exact ranks of the full matrix were known. Nevertheless, all the tools that we described allow to determine these ranks in a dynamical way as the algorithm proceeds, depending on some numerical error threshold ϵ . Hence we can really approximate the given matrix by a rank structured one.*

3.2. Approximating a compressed matrix. We will now show how to approximate a matrix with Givens-weight representation, by one with lower ranks in its underlying rank structure. Thus let us assume that the Givens-weight representation has led to an *overestimation* of the numerical ranks of the underlying structure blocks, at least up to some numerical error threshold ϵ . This fact will then be revealed during the swapping process, where now an additional compressing of the weight matrix can be performed: see Figure 3.1.

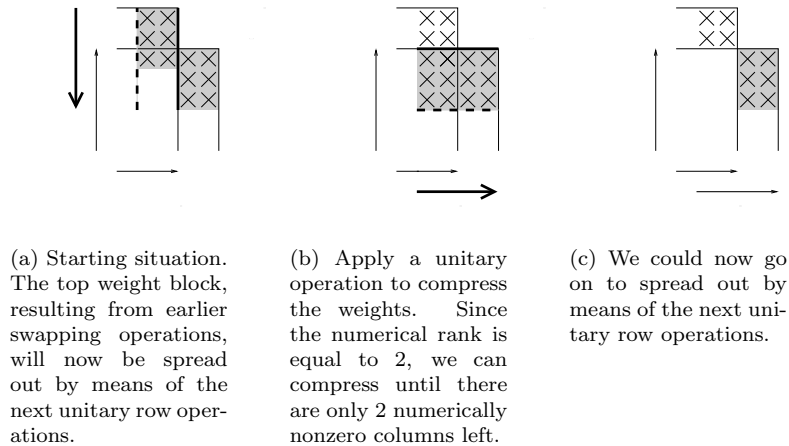


FIGURE 3.1. *Specification of Figure 2.14 in case the numerical ranks of the structure blocks are equal to two. We will then be able to additionally compress the representation.*

As can be seen, the above reduction algorithm follows by an easy application of the swapping procedure.

Let us describe here yet another closely related application of the swapping procedure, namely the process of going from a ‘coarse’ to a ‘fine’ rank structure, in the sense of Figure 3.2. The corresponding algorithm is shown in Figure 3.3.

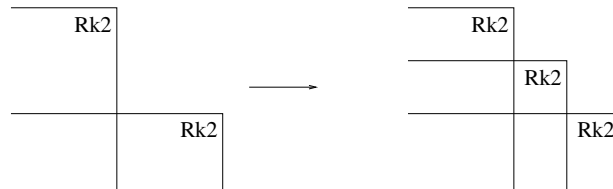


FIGURE 3.2. *Going from a coarse to a fine rank structure. It is assumed here that the ‘intermediate’ structure block is known to be numerically of rank at most two.*

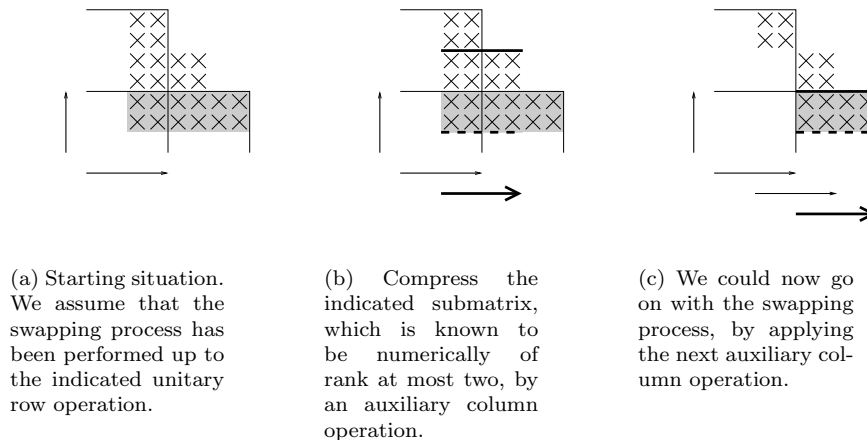


FIGURE 3.3. Going from a coarse to a fine rank structure in terms of the Givens-weight representation, for the example in figure 3.2. The algorithm performs a swapping procedure, during which the intermediate structure blocks are absorbed into the structure. Although the figure shows the case of a single intermediate structure block, the situation is similar in the case of several intermediate structure blocks.

It is easy to see that the process in Figure 3.3 can be extended to deal with *several* intermediate structure blocks, at least when these intermediate structure blocks are treated from top left to bottom right, corresponding to the flow direction of the swapping process.

3.3. Complexity issues. We will describe now some complexity issues for the approximation algorithms described in the previous subsections. First, note that the illustrations in Figures 3.1 and 3.3 have been expressed in terms of a unitary-weight instead of a Givens-weight representation. We already mentioned that this is done mainly for clarity reasons, but in the present case there is also a more fundamental reason, as we explain now.

The reader should recall that for the swapping algorithm of Section 2, the particular sparsity pattern of the Givens-weight representation allowed for an additional benefit, in the sense that the swapped Givens arrows could be immediately read off in terms of the original Givens transformations. This fact was reflected by the Hessenberg-like shapes of the grey weight matrices in Figure 2.14.

In contrast, these observations are *not* true anymore for the algorithms in the previous subsections. The reason is that we are dealing here with *approximation* algorithms, and that the additional compression of the weight blocks should be performed by means of a numerically stable method such as a truncated singular value decomposition, a pivoted QR-factorization or a similar routine. But to our knowledge, such routines are unable to exploit the given shape of the matrix, in the sense that, denoting with r a number such that the number of rows, the number of columns, the original ranks and the numerical ranks are all of order r , then the complexity is $O(r^3)$, even if the original matrix was given in a Hessenberg-like form.

Of course, one can always use the techniques described in Subsection 3.1 (but now with the role of rows and columns reversed) in order to obtain a reduced representation in *canonical* form of type 2. Alternatively, the superfluous Givens transformations could be removed afterwards by means of the pull-through lemma. But the point

that we want to make here is that these techniques can not improve on the $O(r^3)$ complexity per step. Thus in the case of a *dense* rank structure, i.e. when the gaps between the subsequent structure blocks are very small, the reduction process will be of total complexity $O(r^3n)$, which is rather inefficient.

A solution may be to work with only $O(\frac{n}{r})$ structure blocks, i.e. to choose the gaps between the structure blocks to be of order r . At the rate of an $O(r^3)$ complexity per step, the total complexity reduces then to $O(r^2n)$. This type of solution was sometimes also observed in the literature: see e.g. [2, page 13].

Finally, we recall once more that the complexity problems described above are really inherent to the *approximation* problems occurring in the current section. In contrast, most other algorithms to be described in this and further papers will be *exact* and will therefore be able to both exploit and preserve the sparsity of the Givens-weight representation, disregarding the underlying distribution of structure blocks.

We close this section with some references to similar algorithms in the literature. Approximation algorithms were already given in [6, Chapters 3 and 5] in a more general operator-theoretical context. In particular, algorithms were provided there to obtain a block quasiseparable representation in input or output normal form, which are the theoretical equivalents of unitary-weight (but not Givens-weight) representations: see Section 5. A more recent reference about approximation algorithms can be found in [2, Section 11].

4. Updating the representation under the action of Givens transformations. In this section we will show how the Givens-weight representation can be updated under the influence of *a sequence of Givens transformations*. This problem is particularly important since sequences of Givens transformations form a key ingredient in several matrix algorithms, such as QR-factorization, reduction to Hessenberg form, the QR-algorithm, and so on.

In the following, we restrict ourselves to the structured lower triangular part, and we assume that the Givens-weight representation is row-based. We will then show how the representation can be updated under the influence of a sequence of Givens transformations, which we assume to be of one of the following forms: a leftgoing or rightgoing sequence of Givens transformations acting on the columns, or an upgoing or downgoing sequence of Givens transformations acting on the rows. Each of these four cases will be handled with an appropriate technique for updating the representation.

The first case that we describe is when the sequence of Givens transformations is acting on the columns and going leftwards. Obviously, these operations will lead to an increase of the ranks of the structure blocks, as explained in Figure 4.1.

The corresponding operations on the Givens-weight representation are shown in Figure 4.2.

First, Figure 4.2(a) shows the initial Givens-weight representation. Here the rightmost column is assumed to be a result of the action of earlier column operations.

We want now to apply the next Givens transformations to the columns: see Figure 4.2(b). These operations are first applied to the unstructured matrix part in rows 1, ..., 5. On the other hand, the rank structured matrix part in rows 6, 7 and 8 can be updated by just applying the column operations to the weights. The combination of these two operations is indicated by the downpointing arrow in Figure 4.2(b), indicated in boldface.

We should still explain why it is valid to apply the column operations directly to the weights. To this end, we recall that the weights contain compressed information

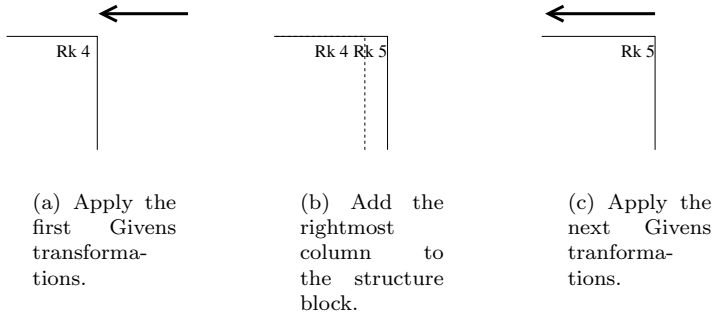


FIGURE 4.1. By applying a leftgoing sequence of Givens transformations, the ranks of the structure blocks increase w.r.t. their original value.

about all the elements below of them, and that in order to obtain these elements in full form, the weights should first be spread out by the unitary row operations of the Givens-weight representation. But clearly, by the associativity of matrix multiplication, it does not matter whether we first spread out the weights by the use of these row operations, or instead first apply the disturbing column operations. This shows that indeed, it is allowed to apply the column operations directly to the weights.

We would then like to go on by applying the next column operations. But doing this would lead to a mixture of real-size elements and weights, which is definitely not allowed. Therefore, we first have to ‘enlarge’ the action radius of the representation by ‘absorbing’ the disturbing column into it: see Figure 4.2(c).

Before applying the next column operations, we may note that applying all these next column operations will ultimately lead to a complete fill-in of the lower triangular part. In order to avoid this, it is now the right moment to apply an auxiliary row operation, to bring the weights as much as possible to the top: see Figure 4.2(d). This row operation is then added to the representation. Note that the fact that we have auxiliary operations being added to the representation, as well as the fact that the weight blocks have ‘grown’ w.r.t. their initial size, both reflect the fact that the ranks of the structure blocks should increase by the application of the Givens transformations, as it was explained earlier in Figure 4.1.

The updated Givens-weight representation is shown in Figure 4.2(e). Because this situation is similar to the one we started from in Figure 4.2(a), the application of the next column operations is not shown anymore.

The overall procedure which we just described will be called a *generalized swapping procedure*, since the original sequence of Givens transformations acting on the *columns*, was ‘swapped’ by the algorithm into an auxiliary sequence of Givens transformations acting on the *rows*, which was then added to the representation. We may note that the number of swapped Givens transformations is again of the same order as the number of original Givens transformations.

The complexity of this algorithm arises from two sources. The first source is due to the fact that during the algorithm, each Givens arrow of the Givens-weight representation will be involved *exactly one time* in the process of enlarging its action radius with one extra column. The second source is due to the generalized swapping itself, for which we can make the same comments as before for the usual swapping algorithm. Thus if the Givens-weight representation is efficient in the sense of Definition

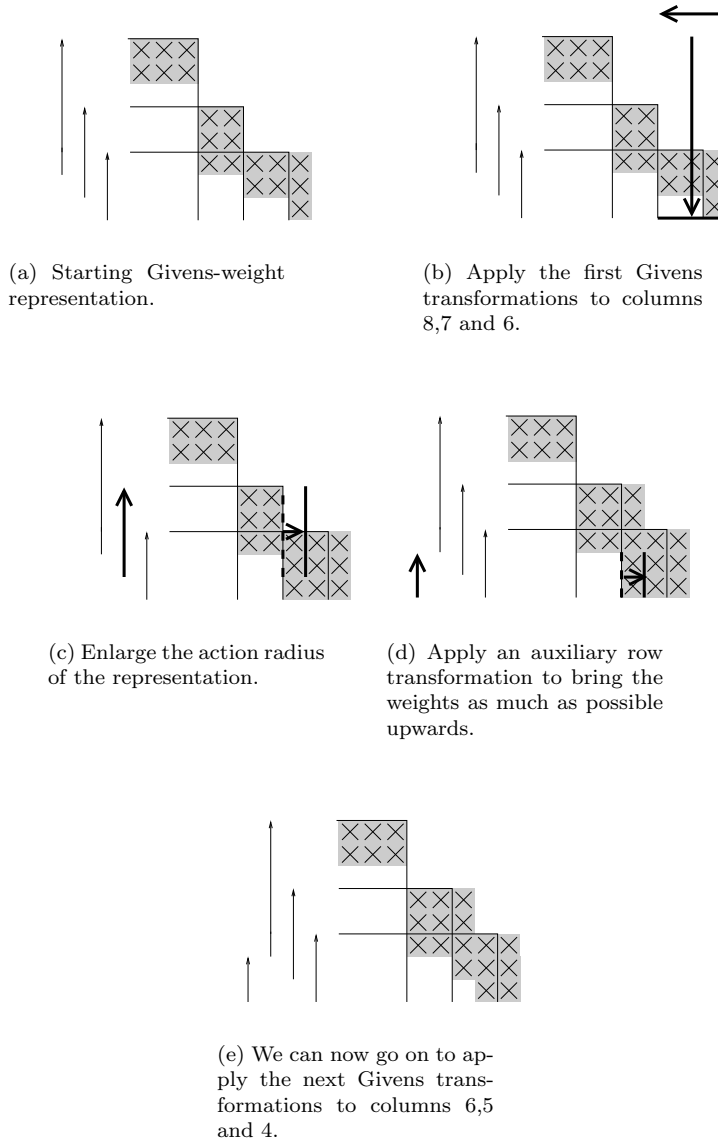


FIGURE 4.2. Updating the Givens-weight representation under the action of a leftgoing sequence of Givens transformations. We refer to these operations as a generalized swapping process.

7, it follows that the complexity of both parts reduces to $O(rn)$.

REMARK 10. Some theoretical equivalents to the operations of ‘enlarging’ and, further on, ‘regressing’ the action radius, appear in the work of Chandrasekaran, Gu et al. under the names of merging and splitting the representation: see e.g. [1].

REMARK 11. Instead of a sequence of individual Givens transformations, one could also update the representation under the action of a sequence of Givens arrows, each of them pointing in a direction opposite to the global algorithm flow, as usual. The latter situation will occur in several concrete places where the generalized swapping

algorithm is used, see e.g. [4]. Denoting with s the average width of the Givens arrows, then the complexity bound has to be updated to the form $O(rsn)$. Note that if $s = r$, we retrieve the $O(r^2n)$ cost of the usual swapping process.

The second case that we distinguish is when the sequence of Givens transformations is acting on the columns and going rightwards. By these operations, the ranks of the structure blocks will remain intact, but every structure block will lose one column: see Figure 4.3.

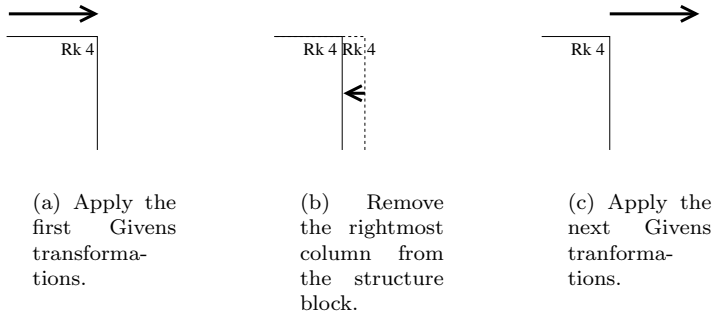


FIGURE 4.3. By applying a rightgoing sequence of Givens transformations, the structure blocks lose one column.

The corresponding operations on the Givens-weight representation are shown in Figure 4.4.

Let us comment on this figure. Figure 4.4(a) shows the initial Givens-weight representation. Figure 4.4(b) shows the application of the first Givens transformations to the columns. Just as in the situation of Figure 4.2 given earlier, we can apply these operations directly to the weights.

We would then like to go on by applying the next column operations. However, note that doing this would lead to a mixture of real-size elements and weights, which is not allowed. We avoid this problem by first ‘regressing’ the action radius of the representation: see Figure 4.4(c).

The updated Givens-weight representation is then shown in Figure 4.4(d). Because this situation is similar to the one we started from in Figure 4.4(a), the application of the next column operations is not shown anymore. The overall algorithm as just described will be called a *generalized regression process*.

Similarly to the generalized swapping process, the complexity of the generalized regression process also arises from two different sources. These two sources are given by the regression of the action radii of the representation, as well as the actual application of the column Givens transformations to the weights. If we assume that the representation is efficient, then for the same reasons as before, the complexity of both procedures reduces to $O(rn)$.

The third case that we distinguish is when the sequence of Givens transformations is acting on the *rows* and going downwards. By these operations, the ranks of the structure blocks will increase: see Figure 4.5.

In terms of the Givens-weight representation, we can then just *concatenate* the disturbing Givens chain to the representation: see Figure 4.6.

Let us comment on this figure. Figure 4.6(a) shows the initial Givens-weight representation. Note that we used here a different notation w.r.t. the previous figures,

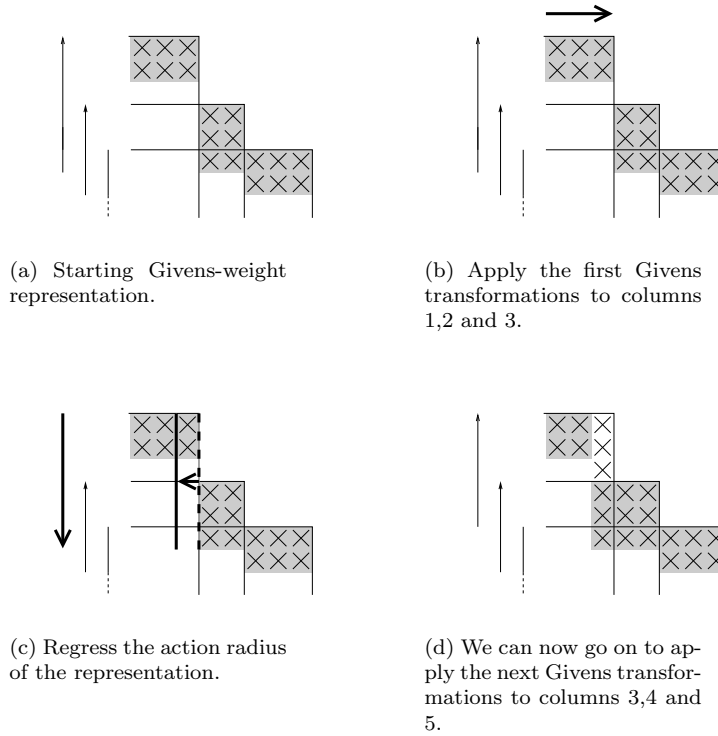


FIGURE 4.4. *Updating the Givens-weight representation under the action of a rightgoing sequence of Givens transformations. We refer to these operations as a generalized regression process.*

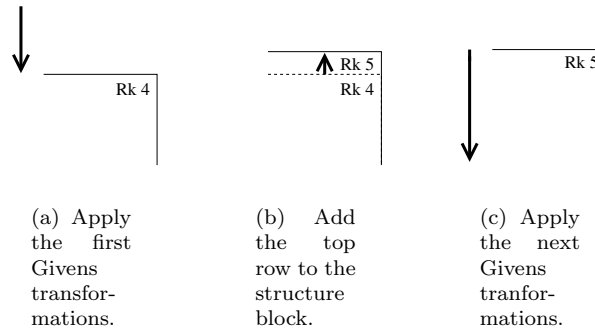


FIGURE 4.5. *By applying a downgoing sequence of Givens transformations, the ranks of the structure blocks increase w.r.t. their original value.*

in the sense that the arrows on the left denote now the unitary transformations used to *spread out* the structure, instead of those used to compress the structure. Thus in order to obtain the full matrix, we should spread out the weight matrix by applying the subsequent unitary operations denoted in the figure, hereby applying them now from right to left.

Figure 4.6(b) shows the application of the first Givens transformations. We ap-

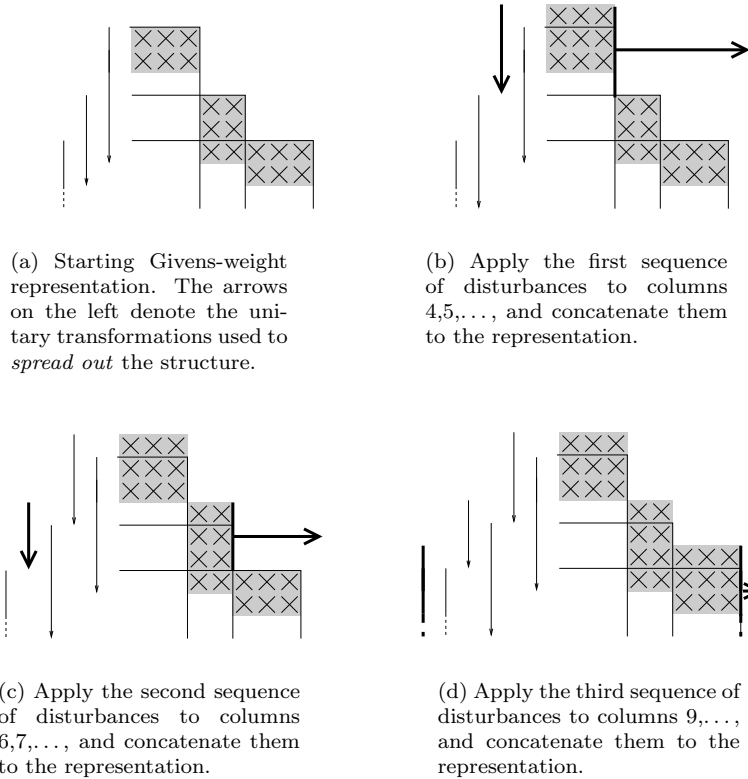


FIGURE 4.6. Updating the Givens-weight representation under the action of a downgoing sequence of Givens transformations. We refer to these operations as a concatenation process.

ply them first to the unstructured matrix part in rows 4,5,..., as indicated by the rightpointing arrow in the figure. On the other hand, for the application to the rank structured matrix part in rows 1,2 and 3, we do not actually apply the Givens transformations but instead just *concatenate* them to the structure.

To see why this concatenation process is allowed, we recall that in order to retrieve the full matrix, the weight matrix should first be spread out by means of all the unitary row operations indicated in the figure, evaluating them from right to left, and hence applying *at the very end* the newly concatenated Givens transformations. Thus indeed, concatenating to the representation is equivalent to applying these Givens transformations to the full matrix.

Still concerning Figure 4.6(b), note that the (1,1), (1,2) and (1,3) elements have turned from white into grey. This reflects the fact that these elements do not contain real-size elements of the matrix anymore, but instead should first be spread out by the newly concatenated Givens transformations in order to see their real meaning. Note that the fact that we have new Givens transformations being concatenated to the representation, as well as the fact that the weight blocks have grown w.r.t. their initial size, both reflect the fact that the ranks of the structure blocks increase by one, as it was explained earlier in Figure 4.5.

Figures 4.6(c) and 4.6(d) show then in an analogous way the application of the

next Givens transformations.

The fourth and last case that we distinguish is when the sequence of Givens transformations is acting on the rows and going upwards. By these operations, the ranks of the structure blocks will remain intact, but every structure block will lose one row: see Figure 4.7.

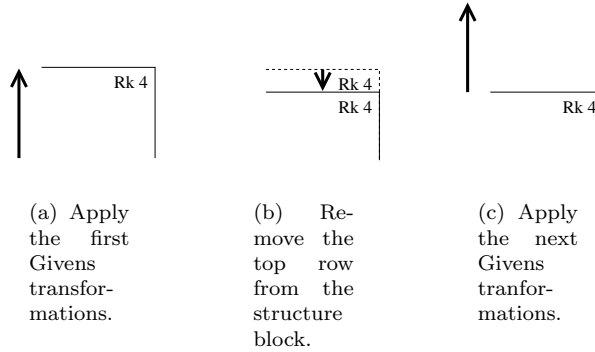


FIGURE 4.7. By applying an upgoing sequence of Givens transformations, the structure blocks lose one row.

The corresponding operations on the Givens-weight representation are shown in Figure 4.8.

Let us comment on this figure. Figure 4.8(a) shows the initial Givens-weight representation, where again the arrows on the left denote the unitary operations used for *spreading out* the structure. We assume here that the bottom right structure block has already ‘shrunk’ under the action of earlier operations.

Figure 4.8(b) shows the application of the next Givens transformations. We apply them first to the unstructured matrix part in columns 6, 7, ..., as indicated by the rightpointing arrow in the figure. On the other hand, for the application to the rank structured matrix part in columns 1, ..., 5, we do not actually apply the Givens transformations, but instead we first concatenate them to the structure, and subsequently pull them through the other unitary transformations, as indicated by the right-downpointing arrow in Figure 4.8(b), indicated in boldface.

The latter process is shown in detail in Figure 4.9.

Now the pulled-through Givens transformations (in fact *transformation*, since the other Givens transformation does not survive the pull-through process: see Figure 4.9) can then just be *removed* from the representation. To see why this is valid, we recall that in order to obtain the full matrix, the weight matrix should be spread out by the subsequent unitary transformations, hereby applying them from right to left, and hence applying *first* the pulled-through versions of the disturbing Givens transformations. But by the pull-through process, the latter operations have moved downwards so much that they do not act on any weight in columns 1, ..., 5 anymore. Thus indeed, it is allowed to get rid of the pulled-through Givens transformation.

Note that the fact that we could get rid of the pulled-through Givens transformations, reflects the fact that the ranks should remain intact under this process, as explained in Figure 4.7 above, and hence that there should not be any additional weight or Givens transformation being added to the representation.

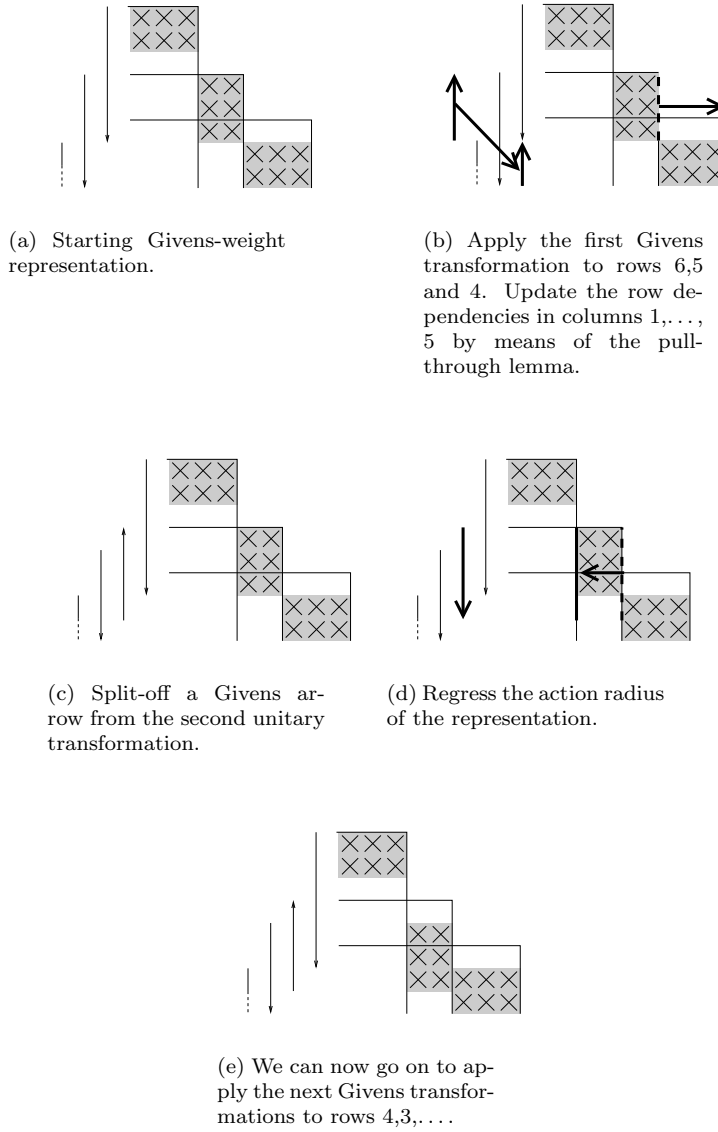


FIGURE 4.8. Updating the Givens-weight representation under the action of an upgoing sequence of Givens transformations. We refer to these operations as a pull-through process.

We would now like to go on by applying the next row operations. However, note that doing this would lead to a mixture of real-size elements and weights, which is not allowed. We avoid this problem by first regressing the action radius of the representation: see Figure 4.8(d).

Concerning this regression process, note that we did not regress the action radius of the complete unitary operation, as shown in Figures 4.8(a) and 4.8(b), but only that of its top Givens arrow, as shown in Figure 4.8(c). Here the reader should keep in mind that, *by definition*, each unitary transformation consists of a number of Givens

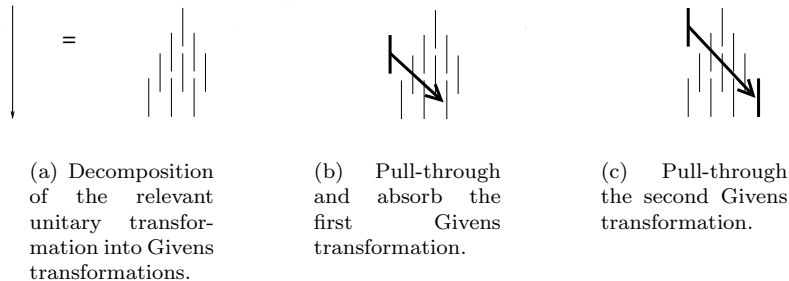


FIGURE 4.9. Detailed description of the pull-through process involved in Figure 4.8(b). The Givens transformations will be pulled through a unitary transformation, which is shown as a full Givens decomposition in Figure 4.9(a). During this process, the first applied Givens transformation in Figure 4.9(b) does not ‘survive’ the pull-through process, in the sense that it can be absorbed into an existing Givens transformation. On the other hand, the second Givens transformation in Figure 4.9(c) does make it to the other side, but as explained above, it can still be removed from the representation.

arrows, each of them pointing in the direction opposite to the global decompression flow. The reason why we usually show the unitary transformations instead of their individual Givens arrows, is just for notational convenience.

The updated Givens-weight representation is shown in Figure 4.8(e). Because this situation is similar to the one we started from in Figure 4.8(a), the next operations are not shown anymore. The complete process as described above will be called a *pull-through process*.

We should still clarify one additional thing about the pull-through process. In order to perform the pull-through operations in Figure 4.9, it was required that the unitary operation U_k had the form imposed by Figure 2.9 (although now in a mirrored form, since we are currently showing the *decompressing* rather than the compressing Givens arrows). But this condition might have been violated by the regression operations performed during the pull-through process, since we regressed each time the action radius of a certain Givens arrow, causing it in fact to be transferred from the unitary operation U_{k+1} to U_k : see Figure 4.8(d). Therefore, the unitary operation U_k should first be restored into its original form by means of the pull-through lemma. Two cases may occur in this process. If the ‘transferred’ Givens arrow was shorter than those of U_k , then some of the bottom Givens transformations in Figure 2.9 will equal the identity matrix. On the other hand, if the ‘transferred’ Givens arrow was longer than those of U_k , then its superfluous Givens transformations could just be thrown away in a way much similar to Figure 4.9. It is clear that none of these two cases breaks the generality of the pull-through process.

Summarizing, we have now completely described how to update the Givens-weight representation under the action of a sequence of Givens transformations, which we assumed to be of one of four possible types. These updating operations are fundamental in manipulating the Givens-weight representation.

Concerning the complexity, it can be noted that each of the described algorithms is able to exploit and preserve the sparsity of the Givens-weight representation. If the original representation was efficient in the sense of Definition 7, then we obtain each time a complexity of $O(rn)$. (Except for the concatenation process, which is in principle costless, at least for its application to the structured lower triangular part).

Finally, we recall that in practical situations, the representation will often have to be updated under the action of a sequence of Givens *arrows* rather than individual Givens transformations: see Remark 11. Denoting the average width of these Givens arrows with s , then the complexity has to be updated each time as $O(rsn)$. Thus in the important case where $r = s$, the complexity takes the form $O(r^2n)$.

REMARK 12. (*Gaussian-weight representation:*) *Another direction in which to extend these results is to update the representation under the action of a chain of elementary, but not necessarily Givens transformations. We use here the term elementary operation to denote an operation which differs from the identity matrix only in two subsequent rows or columns. An example of this are the elementary Gaussian operations, whose non-trivial part is of the form $\begin{bmatrix} 1 & 0 \\ a & 1 \end{bmatrix} P$ for certain $a \in \mathbb{C}$ and 2 by 2 permutation matrix P (at least assuming that we work with row operations). We note that the generalized swapping and regression processes described in this section can be extended to this case without difficulty, while the two other cases are somewhat more problematic.*

More fundamentally, one could also consider the representation itself to be built up by means of elementary Gaussian operations: a Gaussian-weight representation. In fact almost all operations which we described in this paper can be reformulated for this case. (A notable exception are the pull-through operations, for which no extension to the Gaussian case exists.) Most important in this respect is the ‘Givens-like’ property that for any vector in \mathbb{C}^2 , one can find an elementary Gaussian operation to annihilate its bottom element. Moreover, by appropriate choice of the pivoting matrix P one can even guarantee that $|a| \leq 1$. Amongst others, this property allows e.g. to swap from a Givens-weight to a Gaussian-weight representation, and vice versa.

5. Matrix-vector multiplication. In the present section we describe how a matrix $A \in \mathbb{C}^{m \times n}$ represented by a Givens-weight representation can be multiplied with a vector $\mathbf{x} \in \mathbb{C}^n$.

A first note is that such matrix-vector multiplications are obviously *additive*, in the sense that we can freely split the matrix A into a sum $A = A_1 + A_2 + A_3$, hereby transforming the matrix-vector product into a sum $A_1\mathbf{x} + A_2\mathbf{x} + A_3\mathbf{x}$.

This additivity allows us to immediately split the problem: let there be given a Givens-weight representation for the structured lower triangular part and one for the structured upper triangular part of A , which do not overlap, together with possibly a few real-size elements around the diagonal. The multiplication with the unstructured matrix part can be achieved by usual sparse matrix techniques. Hence it will suffice to describe the matrix-vector multiplication with the structured lower and upper triangular parts.

First we consider the structured lower triangular part. We will assume that its Givens-weight representation is based on *row* operations (say): see Figure 5.1.

Let us comment on this figure. Figure 5.1(a) shows the starting situation, where we have already computed the matrix-vector product of the vector of unknowns with the top left block of weights. Note that in these figures, the vector of unknowns is represented by a vetical sequence of crosses, just as the components of the matrix-vector product which have already been computed.

To proceed further, we should recall the precise meaning of the Givens-weight representation: we recall that the weights contain compressed information about the full matrix, which can be obtained in its real form by spreading out, i.e. decompressing the weight blocks by the subsequent transposed unitary row operations of the Givens-

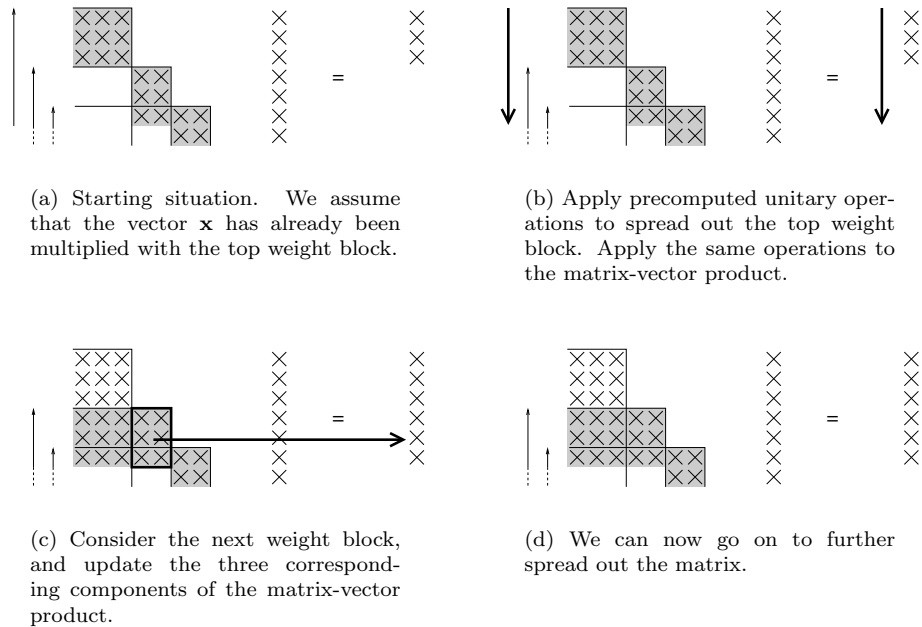


FIGURE 5.1. Example of a matrix-vector multiplication with the structured lower triangular part. The Givens-weight representation is assumed to be based on row operations.

weight representation. We imagine now that we start performing this spreading-out process. These spreading-out operations are not actually performed; instead we only perform them to the already computed components of the matrix-vector product: see Figure 5.1(b).

We are then at the moment of entering a new structure block. This is the right moment to consider also the next block of weights, and to multiply it with the vector of unknowns: see Figure 5.1(c). In this way, the corresponding components of the matrix-vector product are updated. Figure 5.1(d) shows the updated situation; we could now go on with the spreading-out process, and at the end of this process we will have obtained the matrix-vector product in its full form.

Let us now consider the matrix-vector multiplication with the structured upper triangular part. We will assume that its Givens-weight representation is based on *column* operations (say): see Figure 5.2.

Let us comment on this figure. The basic flow of the algorithm is determined by computing the subsequent components of the matrix-vector product, hereby proceeding from bottom to top of the matrix. Such an operation is shown in Figure 5.2(b).

We are then at the point of entering a new structure block in the upper triangular part. This is the right moment to multiply the matrix by the precomputed unitary column operations associated to this structure block, hereby compressing the matrix. Although this operation may sound rather expensive, from the computational point of view, nothing has to be done since the effect of this unitary column operation has already been *precomputed*, by the concept of Givens-weight representation. (In principle we should still apply this operation to the rows below the current action radius,

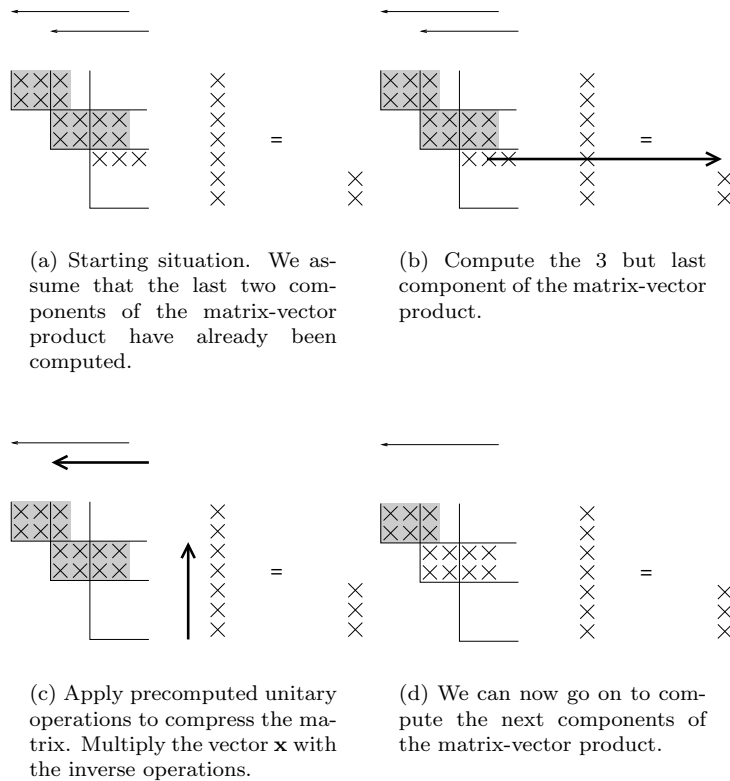


FIGURE 5.2. Example of a matrix-vector multiplication with the structured upper triangular part. The Givens-weight representation is assumed to be based on column operations.

but these rows have already been treated and thrown away). The only operation that actually *has* to be performed, is that we must multiply the vector of unknowns \mathbf{x} with the transpose of this unitary operation: see Figure 5.2(c).

Let us point out that, since these vertical arrows contain the inverse operations of the horizontal arrows in Figure 5.2(c), one could expect them to point in the direction *opposite* to the one indicated in Figure 5.2(c); but this is not correct since the former arrows act on columns while the latter act on the *rows*: the reader should think about this.

Figure 5.2(d) shows the updated matrix. Since this situation is similar to the one we started from, the computation of the next components of the matrix-vector product is not shown anymore.

Let us recall that in the above discussion, it was assumed that the block lower triangular part is represented by a row-based representation, while the block upper triangular part is represented by a column-based representation. But these assumptions could be easily removed.

The complexity of the above described algorithms arises from two different sources. The first source is due to the multiplication with the weights, while the second source arises from the application of Givens transformations to a vector. If the Givens-weight representation was efficient in the sense of Definition 7, it is easy to see that both op-

erations have complexity $O(rn)$. In fact, this bound should not come as a surprise since during the matrix-vector multiplication, every ingredient of the Givens-weight representation is involved *exactly one time*.

REMARK 13. *The complexity of the matrix-vector multiplication can be further speeded up by working with a Gaussian-weight representation (Remark 12). Indeed: while multiplying a Givens transformation with a vector involves 4 multiplications and 2 additions, in the Gaussian case this operation involves essentially only 1 multiplication and 1 addition. However, we should warn that our numerical experiments indicate that the Gaussian-weight representation may not be numerically stable, especially in case of high semiseparability ranks.*

REMARK 14. *A probably numerically more robust alternative compared to the previous remark is to use a full unitary-weight representation. We already emphasized that for this approach to be efficient, the distance between two subsequent structure blocks should be chosen to be approximately equal to the corresponding rank index. It can be shown that if one takes care of this condition, then the full unitary-weight representation may beat the Givens-weight representation.*

Yet another possibility is to use a Householder-weight representation. We define this as a unitary-weight representation in which each unitary component U_k consists of a single downpointing ‘Householder-arrow’ of width r_k , the latter being defined as a sequence of r_k Householder operations [11] acting on some localized rows, in an obvious way. For such representations to be efficient, one should also take care that the distance between two subsequent structure blocks approximately equals the corresponding rank index.

Also in the literature, algorithms have been developed to perform the matrix-vector multiplication with a rank structured matrix: see the book [6], where the matrix-vector multiplication has an interpretation in terms of the action of a time-varying system on a vector of input data, using a block quasiseparable representation. See also [8].

6. Comparison with other representations. In this section the Givens-weight representation will be compared with two other representations frequently encountered in the literature: block quasiseparable representations and uv -representations. For both these representations we describe an easy algorithm to transform the representation into the Givens-weight form. Moreover, it will be shown that especially the class of *block quasiseparable* representations is tightly connected to the Givens-weight format, and the similarities and differences between these two types of representation will be emphasized.

6.1. Quasiseparable representations. First we will focus on block quasiseparable representations. The idea for these representations has essentially been introduced in the book [6]. In the current paper, we will follow the terminology of Eidelman and Gohberg by calling these matrices block quasiseparable [8], but the reader should be aware that they appear also under other names in the literature, such as sequentially semiseparable matrices, matrices with low Hankel rank, and so on.

We will start with the most general definition of block quasiseparable matrices. A matrix is called *block quasiseparable* w.r.t. a given block partition, if it can be divided

as a block matrix w.r.t. this partition, with (i, j) th block element given by

$$\begin{cases} A_{i,j} = P_i T_{i-1} T_{i-2} \dots T_{j+1} Q_j, & i \geq j + 1, \\ A_{i,j} = D_i, & i = j, \\ A_{i,j} = G_i S_{i+1} S_{i+2} \dots S_{j-1} H_j, & i \leq j - 1 \end{cases}$$

for $i, j = 0, \dots, K$. Here an empty product denotes the identity matrix. The sizes of all auxiliary matrices occurring in these formulas must be chosen to be compatible with each other. In the literature, these matrices must sometimes satisfy some additional size restrictions, such as the fact that the D_i are scalar (these lead then to the usual, i.e. scalar *quasiseparable* matrices, occurring in many papers by Eidelman and Gohberg), the fact that the D_i are square, and so on. These conditions vary sometimes from paper to paper.

We will refer to the matrices T_k and S_k as *transition matrices*, although this terminology is somewhat nonstandard. Moreover, we will refer to the P_i , G_i and Q_j , H_j as *row and column shaft generators*, respectively.

The block partition corresponding to a given block quasiseparable representation, immediately reveals the shape of the underlying structure blocks \mathcal{B}_k . In the sequel we will assume that these structure blocks \mathcal{B}_k , $k = 1, \dots, K$ are labeled from top left to bottom right. Note also that the block quasiseparable formulae imply a certain limitation about the relative positions of the structure blocks in the block lower versus the block upper triangular part, but this restriction is not essential.

Figure 6.1 shows the block quasiseparable representation in a schematic way.

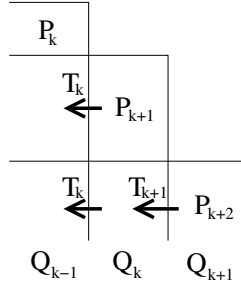


FIGURE 6.1. Schematic picture of a block quasiseparable representation. In order to obtain the (i, j) th block element, we should multiply the corresponding row shaft generator P_i and the corresponding column shaft generator Q_j , with in between the product of all transition matrices T_k that are needed to go from P_i to Q_j in the picture.

Now we consider the problem of transforming a given block quasiseparable representation into the Givens-weight format. We will restrict ourselves to the structured lower triangular part: the Givens transformations constituting the first unitary transformation U_K can be derived from the QR-factorization of P_K . Then denoting by X_K the square top part of the resulting R-factor, the Givens transformations constituting the next unitary operations U_k can be derived from the QR-factorization of $\begin{bmatrix} P_k \\ X_{k+1} T_k \end{bmatrix}$, for $k = K - 1, \dots, 1$. Here we defined each time the new X_k to be the square top part of the resulting R-factor. During this process, also the corresponding weight blocks can be computed each time as $X_k Q_{k-1}$: see Figure 6.2.

Concerning the complexity of this algorithm, it can be argued that the algorithm can lead to an $O(r^3 n)$ complexity in case of ‘dense’ rank structures, but only $O(r^2 n)$

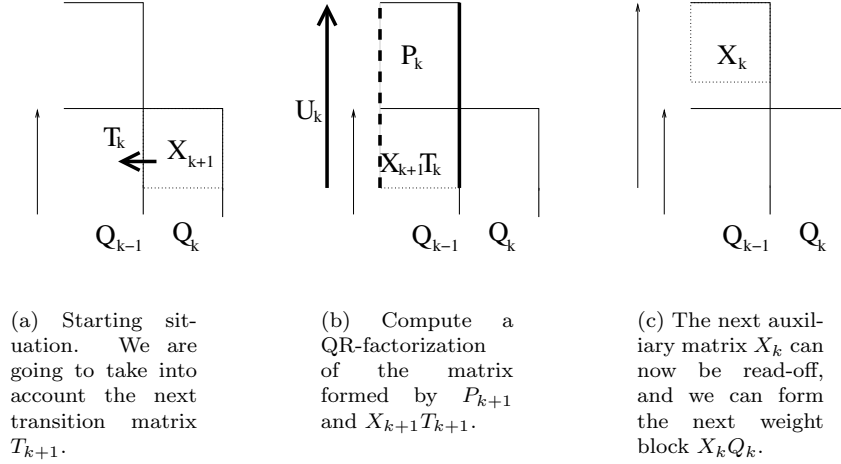


FIGURE 6.2. Transition from a block quasiseparable to a Givens-weight representation.

when the gaps between the structure blocks are of the same order as the corresponding rank indices. In fact, this should not come as a surprise, in the sense that the algorithm in Figure 6.2 has many resemblances with the process of swapping a *full* unitary-weight representation as described in Section 2. (Or in a more appropriate way, quasiseparable representations could be compared with *nonsingular*-weight rather than *unitary*-weight representations: see further.) Still pursuing this similarity, note that one can use the same techniques as described in Subsection 3.1 to obtain a Givens-weight representation in *canonical* form.

Conversely, suppose now that we have given a Givens-weight representation, or more generally a unitary-weight representation, and that we want to find a quasiseparable representation for it. The reader should then reacquire familiarity with Definition 3: denoting with $W_k \in \mathbb{C}^{r_k \times |J_k|}$ the k th weight block and with U_k the k th unitary transformation of the Givens-weight representation, we recall that the spreading-out process starts by forming

$$U_k^H \begin{bmatrix} W_k \\ 0 \end{bmatrix}.$$

Subsequently, the bottommost $r_{k+1} = |I_{k+1, \text{top}}|$ rows are further spread out by the next unitary transformations U_{k+1}^H, \dots . This suggests that we may obtain the quasiseparable parameters by subdividing

$$U_k^H = \left[\begin{array}{c|c} P_k & X \\ \hline T_k & X \end{array} \right],$$

where the blocks P_k and T_k have $|I_k|$ and $|I_{k+1, \text{top}}| = r_{k+1}$ rows, respectively, and r_k columns. The quasiseparable generators Q_k are chosen each time as $Q_k := W_{k+1}$, $k = 0, \dots, K$.

The dynamics of this algorithm are illustrated in Figure 6.3.

Still concerning the algorithm to go from a Givens-weight to a block quasiseparable representation, note that the above discussion revealed that unitary-weight representations are theoretically equivalent with the matrices $\begin{bmatrix} P_k \\ T_k \end{bmatrix}$ of the quasiseparable

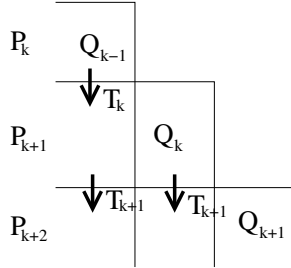


FIGURE 6.3. Schematic picture of a block quasiseparable representation, constructed in a ‘dual’ way w.r.t. Figure 6.1.

representation having orthonormal columns, for each k . Thus we see that unitary-weight representations theoretically correspond to the quasiseparable representations in *input normal form*, following the terminology of [6].

In fact there exists also the notion of *output normal form* in [6], meaning that all matrices $\begin{bmatrix} T_k & Q_k \end{bmatrix}$ must have orthonormal rows, for each k . It can then be argued by a similar argument as above that this notion corresponds to unitary-weight representations that are based on *column*, rather than row operations.

Thus we see that in a certain sense, the Givens-weight representation ‘breaks the symmetry’ of block quasiseparable representations by assigning a preference to either row or column operations. This may seem awkward, but as we already observed, this will not prohibit the Givens-weight format from being very *convenient* for the development of algorithms. Moreover, by the fact that the representation is based on *unitary* operations, which are well-known to be optimally conditioned w.r.t. the matrix 2-norm, it can be expected that an appropriate use of Givens-weight representations should lead to numerically stable algorithms.

Concerning the differences between block quasiseparable and Givens-weight representations, we should stress that:

- The above discussion showed a comparison between block quasiseparable and *unitary-weight* representations. However, the reader should not forget that we are primarily concerned with *Givens-weight* representations, where each unitary transformation U_k has an additional factorization as a sparse product of Givens transformations[‡]. Although it was observed that the Givens-weight representation may deviate by a factor of at most 4 from optimality (Remark 13), it has the advantage that it can *always* be used in an efficient way. Notable exceptions where the performance of the Givens-weight representation is also sensible to the distribution of the structure blocks are the *approximation* algorithms described in Section 3.

6.2. uv -representations. In this subsection we briefly consider uv -representations for rank structured matrices[§]. Such representations sometimes occur in solution

[‡]This sparsity property constitutes an essential difference between Givens-weight and block quasiseparable representations. However, for completeness, we note that some sparse factorizations in terms of individual Givens transformations were also described in [6, Chapter 14]. But the latter discussion concerns *unitary* rank structured matrices, and we were not able to find there any indication in terms of non-unitary rank structured matrices or algorithmic exploitation.

[§]The matrices allowing such a representation are sometimes called ‘generator representable matrices’ instead of uv -representable matrices [15], but we will not use this terminology here since the word ‘generator’ could be confused with e.g. the quasiseparable generators.

methods for differential and integral equations. Historically, the term ‘semiseparable matrix’ was even introduced in the context of uv -representations: see e.g. [10].

First of all, we must stress that in contrast to Givens-weight and block quasiseparable representations, uv -representations do not exist for every rank structured matrix, but instead only for a subclass which we call the class of uv -representable matrices. When implementing the QR-algorithm for semiseparable matrices of semiseparability rank 1 using the uv -representation for such matrices, for example, this can be considered as a severe weakness since the subsequent QR-iterates converge then to a limiting matrix in block upper triangular form, which in general is *not* uv -representable of rank one anymore: see [15].

Let us now give a formal definition of uv -representability.

DEFINITION 15. *Let \mathcal{R} be a rank structure with a global rank upper bound $r_k =: r$ for each k . We say $A \in \mathbb{C}^{m \times n}$ to be uv -representable w.r.t. \mathcal{R} if there exists a factorization*

$$A = uv + A_{\text{completion}}, \quad (6.1)$$

where $u \in \mathbb{C}^{m \times r}$, $v \in \mathbb{C}^{r \times n}$ and where $A_{\text{completion}}$ is a ‘completion’ matrix having the property that its restriction to each of the structure blocks of \mathcal{R} is zero. The factorization (6.1) is called a uv -representation of A .

The above definition states that for a matrix to be uv -representable, the low rank generators of the different structure blocks of \mathcal{R} should be ‘compatible’ in the sense that they can be completed to a global matrix uv of rank at most r .

Now let us give an algorithm to transform a uv -representation into a Givens-weight representation. Such an algorithm is trivial and based on the QR-factorization of the matrix u in (6.1). Let us partition u and v as block matrices according to the given distribution of the structure blocks, with block elements u_k and v_k , for $k = 0, \dots, K$. Moreover, let us assume that in the process of forming the QR-factorization

of u , the bottom submatrix $\begin{bmatrix} u_k \\ \vdots \\ u_K \end{bmatrix}$ has just been made upper triangular. Then

denoting with X_k the square top block of this upper triangular matrix, the k th weight block will be simply $X_k v_{k-1}$. Repeating this process for $k = K, \dots, 1$, at the end we will have obtained a Givens-weight representation for the structured lower triangular part of A . Moreover, assuming that the given matrix is square of size n , then it is easy to check that (i) the algorithm always leads to an *efficient* Givens-weight representation consisting of not more than $O(rn)$ Givens transformations, and (ii) the algorithm has complexity $O(r^2n)$, irrespective of the precise distribution of the structure blocks. These properties should be contrasted with the reduction process for quasiseparable representations in Subsection 6.1.

7. Conclusion. In this paper we have introduced the notions of unitary-weight and Givens-weight representations for rank structured matrices. It was described e.g. how the representation can be swapped, how it can be reduced to a lower complexity representation, how it can be updated under the action of a sequence of Givens transformations and how to use the representation in performing a matrix-vector multiplication. These results provide a basis for several algorithms using Givens-weight representations such as QR-factorization, solution of linear systems [4] and matrix inversion, explicit and implicit QR-iteration, Hessenberg reduction, and so on, to be described in our future work.

REFERENCES

- [1] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen. Fast stable solver for sequentially semi-separable linear systems of equations. *Lecture Notes in Computer Science*, 2552:545–554, 2002.
- [2] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen. Fast stable solvers for sequentially semi-separable linear systems of equations. Technical report, Department of mathematics, UC, Berkeley, 2003.
- [3] 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.
- [4] S. Delvaux and M. Van Barel. A QR-based solver for rank structured matrices. Report TW 454, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 2006.
- [5] 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.
- [6] P. Dewilde and A.-J. van der Veen. *Time-varying systems and computations*. Kluwer Academic Publishers, Boston, June 1998.
- [7] P. Dewilde and A.-J. van der Veen. Inner-outer factorization and the inversion of locally finite systems of equations. *Linear Algebra and its Applications*, 313:53–100, February 2000.
- [8] Y. Eidelman and I. C. Gohberg. On a new class of structured matrices. *Integral Equations and Operator Theory*, 34:293–324, 1999.
- [9] Y. Eidelman and I. C. Gohberg. A modification of the Dewilde-van der Veen method for inversion of finite structured matrices. *Linear Algebra and its Applications*, 343-344:419–450, April 2002.
- [10] I. C. Gohberg, T. Kailath, and I. Koltracht. Linear complexity algorithms for semiseparable matrices. *Integral Equations and Operator Theory*, 8(6):780–804, 1985.
- [11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.
- [12] E. E. Tyrtyshnikov. Mosaic ranks for weakly semiseparable matrices. In 85 2572, 2573, editor, *Large-Scale Scientific Computations of Engineering and Environmental Problems II*, volume 73 of *Notes on numerical fluid mechanics*, pages 36–41. Vieweg, 2000.
- [13] M. Van Barel, E. Van Camp, and N. Mastronardi. Orthogonal similarity transformation into block-semiseparable matrices of semiseparability rank k . *Numerical Linear Algebra with Applications*, 12:981–1000, 2005.
- [14] 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.
- [15] R. Vandebril, M. Van Barel, and N. Mastronardi. A note on the representation and definition of semiseparable matrices. *Numerical Linear Algebra with Applications*, 12(8):839–858, October 2005.