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

Efficient Algorithm for Simultaneous Reduction to the m-Hessenberg–Triangular–Triangular Form

Nela Bosner

the date of receipt and acceptance should be inserted later

Abstract This paper proposes an efficient algorithm for simultaneous reduction of three matrices. The algorithm is a blocked version of the algorithm described by Miminis and Page (1982) which reduces A to the m-Hessenberg form, and B and E to the triangular form. The m-Hessenberg–triangular–triangular form of matrices A, B and E is specially suitable for solving multiple shifted systems. Such shifted systems naturally occur in control theory when evaluating the transfer function of a descriptor system, or in interpolatory model reduction methods. They also arise as a result of discretization of the time-harmonic wave equation in heterogeneous media, or originate from structural dynamics engineering problems. The proposed blocked algorithm for the m-Hessenberg–triangular–triangular reduction is based on the aggregated Givens rotations, which are a generalization of the blocked algorithm for the Hessenberg–triangular reduction proposed by Kågström et al. (2008). Numerical tests confirmed that the blocked algorithm is up to 3.4 times faster than its non-blocked version based on regular Givens rotations only. As an illustration of its efficiency, two applications of the m-Hessenberg–triangular–triangular reduction coming from control theory are described: evaluation of the transfer function of a descriptor system at many complex values, and computation of the staircase form used to identify the controllable part of the system.

Keywords m-Hessenberg–triangular–triangular form · orthogonal transformations · level 3 BLAS · blocked algorithm · solving shifted system · transfer function evaluation · staircase form

Mathematics Subject Classification (2000) 15A21 · 15A06 · 65F05 · 65Y20 · 93B05 · 93B10 · 93B17 · 93B40

1 Introduction

Our goal is to implement an efficient algorithm for solving general shifted systems of the form $(\sigma E - A)X = B$, with $A, E \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$, where $m \ll n$, and for possibly large number of shifts $\sigma \in \mathbb{R}$ or $\sigma \in \mathbb{C}$. Transforming matrices A, B and E into a suitable form can increase efficiency of repeated system solving, as described in [3] and [4] for $E = I$, where I is the identity matrix.

N. Bosner
Department of Mathematics, University of Zagreb, Croatia
Tel.: +385-1-4605781
Fax: +385-1-4680335
E-mail: nela@math.hr

Shifted systems frequently occur in control theory, where in case of descriptor systems

$$\begin{aligned} E\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1.1}$$

with $C \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{p\times m}$, the corresponding transfer function has the following form

$$\mathscr{G}(\sigma) = C(\sigma E - A)^{-1}B + D.$$

Similar shifted systems can be found in interpolatory model reduction where a sequence of subspaces is computed. The subspaces are spanned by $(\sigma_i E - A)^{-1}B$ and $(\sigma_i E - A)^{-*}C^T$, $i = 1,\dots r$ in case when $m = p = 1$, and by $(\sigma_i E - A)^{-1}Bb_i$ and $(\sigma_i E - A)^{-*}C^T c_i$ for suitable vectors b_i and c_i in case when $m, p > 1$. In the iterative rational Krylov optimal $\mathscr{H}_2$ model reduction algorithm (IRKA) [10], [1], a shifted linear solver represents its important part. Shifted systems are naturally obtained as a result of numerical methods applied to mathematical models in applied mathematics. Such examples are coming from acoustic problems in many areas, e.g., from aeronautics, marine technology, geophysics, and optical problems, and they are the result of solving the time-harmonic wave equation in heterogeneous media. Finite Element discretization of a Helmholtz equation, where a model of wave propagation in the earth crust is observed, results with a system of the form $(K - z_1 M)p = b$, $z_1 = (2\pi f)^2$. K is a discretized Laplacian and M is a mass matrix, see for example [8]. Another example comes from the structural dynamics engineering problem, where direct frequency analysis leads to the solution of the algebraic linear system of the form $(\sigma^2 A + \sigma B + C)x = b$, for several values of the frequency-related parameter σ. Linearization yields the system

$$\left(\begin{bmatrix} B & C \\ C^\tau & 0 \end{bmatrix} + \sigma \begin{bmatrix} A & 0 \\ 0 & -C^\tau \end{bmatrix} \right) \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}.$$

For details see [20] and [21].

On the other hand, when solving the generalized eigenvalue problem $Ax = \lambda Ex$ the first step in computing the generalized Schur decomposition of the pair (A,E) is to reduce A and E to the generalized upper Hessenberg form (Hessenberg–triangular form) using orthogonal transformations based on Givens rotations (see, [9, Section 7.7]). Efficient block algorithms for the Hessenberg–triangular reduction are proposed by Kågström et al. in [12]. The reduction results with

$$Q^\tau A Z = H, \qquad Q^\tau E Z = T_E,$$

where H is an upper Hessenberg matrix and T_E is an upper triangular matrix. In this case $\mathscr{G}(\sigma)$ would have the form

$$\mathscr{G}(\sigma) = (CZ)(\sigma T_E - H)^{-1}(Q^\tau B) + D.$$

where a Hessenberg system of linear equations $(\sigma T_E - H)X = Q^\tau B$ has to be solved, but $Q^\tau B$ does not have any convenient form. This is a standard approach for computing the transfer function of the descriptor system, see for example [24]. If we want to use a similar and more efficient approach as in [4], we have to include the matrix B into reduction. This approach can easily be generalized to the descriptor systems, as already described in the same paper.

Now we introduce the m-Hessenberg–triangular–triangular form for the matrices A, B and E. We will show that there exist orthogonal matrices Q, $Z \in \mathbb{R}^{n\times n}$, which simultaneously reduce matrices A, B and E to the desired form.

Definition 1.1 Reduction of the matrices A, B and E to the m-Hessenberg–triangular–triangular form is obtained as

$$\begin{aligned} &\hat{A} = Q^\tau A Z = H && \text{where } H \text{ is an upper } m\text{-Hessenberg matrix,} \\ &\hat{B} = Q^\tau B = \begin{bmatrix} T_B \\ 0 \end{bmatrix} && \text{where } T_B \text{ is an upper triangular matrix,} \\ &\hat{E} = Q^\tau E Z = T_E && \text{where } T_E \text{ is an upper triangular matrix,} \end{aligned}$$

and $Q, Z \in \mathbb{R}^{n\times n}$ are orthogonal.

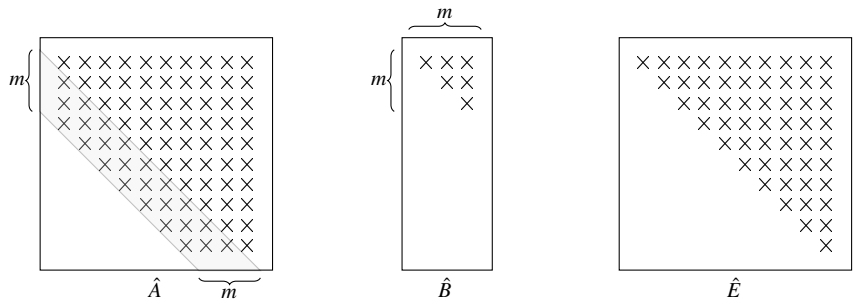


Fig. 1.1: The m-Hessenberg–triangular–triangular form of the matrices $\hat{A}$, $\hat{B}$, and $\hat{E}$.

Besides shifted systems, the m-Hessenberg–triangular–triangular form is used in [19] and [17] as a tool to determine controllability of the system in the single input case ($m = 1$), and in [17] as a preprocessing faze in an algorithm for pole assignment.

In this paper we describe in details the blocked algorithm for the m-Hessenberg–triangular–triangular reduction in Section 3, and in Section 4 we give a short insight in its backward stability analysis. In the following sections we illustrate the application of the m-Hessenberg–triangular–triangular reduction in two cases from control theory. In Section 5 the evaluation of the transfer function is elaborated, showing that this can be done efficiently when the system is in the m-Hessenberg–triangular–triangular form. In Section 6 we describe an efficient algorithm for reduction of the descriptor system to the staircase form, which is based on the m-Hessenberg–triangular–triangular reduction. The staircase form reveals the controllable part of the system. Only an introduction to the GPU algorithm of the m-Hessenberg–triangular–triangular reduction is given in Section 7, and finally in Section 8 we confirm the superiority of our algorithms by presenting the results of our numerical tests.

2 The non-blocked algorithm

The non–blocked algorithm is a generalization of the Hessenberg–triangular reduction described in Algorithm 7.7.1 in [9], and is actually introduced for $m = 1$ by Miminis and Paige in [17](see also [19] for $m > 1$).

1. First, an orthogonal Q_E is determined such that $E = Q_E^\tau E$ is upper triangular, and $B = Q_E^\tau B$ and $A = Q_E^\tau A$ are updated accordingly.
2. The next step is reduction of the matrix B. Since this matrix is affected only by orthogonal transformations from the left, it can be reduced to the upper triangular form while simultaneously preserving upper triangular form of E. In each step of the algorithm a Givens rotation from the left is applied to annihilate one element below the main diagonal of B. The same transformation is then applied to E introducing one nontrivial subdiagonal element. Then, another Givens rotation is applied to E from the right in order to restore its triangular form. Both transformations have to be applied to the matrix A, too.
3. When the reduction of B is finished, the algorithm switches to A. In order to simultaneously reduce the matrix A and preserve the upper triangular form of the matrix E, the optimal reduced form of A is Hessenberg. But, if we take the upper triangular form of B into account, than the desired reduced form of A is m-Hessenberg. Orthogonal transformations applied to A from the

left determined to annihilate elements bellow its m-th subdiagonal, when applied to B will not affect its reduced form.

4. The role of the matrix C is rather passive. It is updated by all orthogonal transformations from the right that were produced in the reduction.

To sum up, this algorithm reduces the first n columns of $[\,B\ A\,]$ to the upper triangular form in $n-1$ outer-loop steps, restoring the triangular form of E at each inner-loop step.

3 The blocked algorithm

First, we should notice that in case of descriptor systems, matrices E that occur in practice are sparse and structured. Thus, the triangular reduction of E can be performed by a structure specific transformation, which is much more efficient than the standard QR factorization. On the other hand, our algorithm is developed for a general case, where we do not imply any structure or non-singularity of E, and can be applied for solving general shifted systems.

The blocked algorithm for the m-Hessenberg–triangular–triangular reduction is a generalization of the Level 3 BLAS [7] variant of the algorithm for the Hessenberg–triangular reduction based on Givens rotations, and introduced by Kågström et al. in [12] (see also [14]). Since the application of Givens rotations in the non–blocked algorithm is performed only by Level 1 BLAS operations both with unit and non-unit stride, this algorithm does not use the fast cache memory optimally. To increase its efficiency we must optimize the usage of cache memory by reordering operations in the algorithm, which will allow us to utilize efficient Level 3 BLAS operations based on the matrix–matrix product.

We will start with an observation. In our algorithm we are annihilating consecutive elements of $[\,B\ A\,]$ columnwise, so we will use the same notation as in [12]: $G^{(j)}_{i-1,i}$ denotes the Givens rotation used to annihilate $B(i,j)$ or $A(i,j-m)$ if $j>m$. This is the Givens rotation generated in the j-th outer loop step and the i-th inner loop step of the non–blocked algorithm. In order to determine this Givens rotation, all rotations from previous steps have to be applied to B or A in a given order. On the other hand, if we want to apply a sequence of Givens rotations defined in the non–blocked algorithm to a vector $x\in\mathbb{R}^n$ which does not determine any of the rotations from the sequence, then we have some freedom to reorder the rotations. Still, there are some dependencies between Givens rotations from different outer loop steps that we cannot avoid.

$$
\begin{array}{l}
G^{(1)}_{12,13}\to G^{(1)}_{11,12}\to G^{(1)}_{10,11}\to G^{(1)}_{9,10}\to G^{(1)}_{8,9}\to G^{(1)}_{7,8}\to G^{(1)}_{6,7}\to G^{(1)}_{5,6}\to G^{(1)}_{4,5}\to G^{(1)}_{3,4}\to G^{(1)}_{2,3}\to G^{(1)}_{1,2}\\
G^{(2)}_{12,13}\to G^{(2)}_{11,12}\to G^{(2)}_{10,11}\to G^{(2)}_{9,10}\to G^{(2)}_{8,9}\to G^{(2)}_{7,8}\to G^{(2)}_{6,7}\to G^{(2)}_{5,6}\to G^{(2)}_{4,5}\to G^{(2)}_{3,4}\to G^{(2)}_{2,3}\\
G^{(3)}_{12,13}\to G^{(3)}_{11,12}\to G^{(3)}_{10,11}\to G^{(3)}_{9,10}\to G^{(3)}_{8,9}\to G^{(3)}_{7,8}\to G^{(3)}_{6,7}\to G^{(3)}_{5,6}\to G^{(3)}_{4,5}\to G^{(3)}_{3,4}\\
G^{(4)}_{12,13}\to G^{(4)}_{11,12}\to G^{(4)}_{10,11}\to G^{(4)}_{9,10}\to G^{(4)}_{8,9}\to G^{(4)}_{7,8}\to G^{(4)}_{6,7}\to G^{(4)}_{5,6}\to G^{(4)}_{4,5}\\
\qquad\qquad\qquad\qquad\cdots
\end{array}
$$

Fig. 3.1: The sequence of Givens rotations in the non-blocked algorithm for $n=13$.

Let us define $x^{(0)}=x$, and $x^{(j)}=G^{(j)}_{j,j+1}G^{(j)}_{j+1,j+2}\cdots G^{(j)}_{n-1,n}x^{(j-1)}$ is the updated vector x after j outer loop steps. Then, in order to determine its i-th element $x^{(j)}(i)$, under assumption that $G^{(j)}_{n-1,n},\ldots,G^{(j)}_{i,i+1}$ are already applied to $x^{(j-1)}(i:n)$, we only need to apply $G^{(j)}_{i-1,i}$ to $x^{(j-1)}(i-1:i)$. Further, to determine $x^{(j-1)}(i-1:i)$, under assumption that $G^{(j-1)}_{n-1,n},\ldots,G^{(j-1)}_{i,i+1}$ are already applied

to $x^{(j-2)}(i:n)$ we need to apply $G_{i-1,i}^{(j-1)}$ and $G_{i-2,i-1}^{(j-1)}$ to $x^{(j-2)}(i-2:i)$. To determine $x^{(j-2)}(i-2:i)$, under assumption that $G_{n-1,n}^{(j-2)},\ldots,G_{i,i+1}^{(j-2)}$ are already applied to $x^{(j-3)}(i:n)$ we need to apply $G_{i-1,i}^{(j-2)}$, $G_{i-2,i-1}^{(j-2)}$ and $G_{i-3,i-2}^{(j-2)}$ to $x^{(j-3)}(i-3:i)$, and so on.

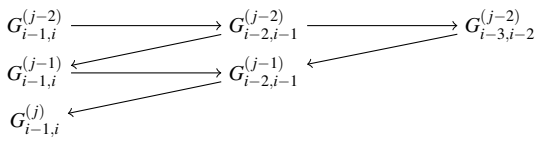


Fig. 3.2: The dependencies of the rotations affecting $x^{(j)}(i)$. An arrow $S \to T$ means that T can be applied after S.

Further, since the rotations $G_{i_1-1,i_1}^{(j_1)}$ and $G_{i_2-1,i_2}^{(j_2)}$ commute when $i_2 > i_1+1$ we can conclude that

- $G_{i-1,i}^{(j-1)}$, $G_{i-2,i-1}^{(j-1)}$ and $G_{i-1,i}^{(j)}$ commute with the sequence of rotations $G_{i-4,i-3}^{(j-2)},\ldots,G_{j-2,j-1}^{(j-2)}$, and
- $G_{i-1,i}^{(j)}$ commute with the sequence of rotations $G_{i-3,i-2}^{(j-1)},\ldots,G_{j-1,j}^{(j-1)}$, hence
- we can apply rotations presented in Figure 3.2 one after another, in a sequence

$$G_{i-1,i}^{(j-2)} \longrightarrow G_{i-2,i-1}^{(j-2)} \longrightarrow G_{i-3,i-2}^{(j-2)} \longrightarrow G_{i-1,i}^{(j-1)} \longrightarrow G_{i-2,i-1}^{(j-1)} \longrightarrow G_{i-1,i}^{(j)}$$

This sequence is more localized, in sense that when applied to the vector x only elements $x(i-3)$, $x(i-2)$, $x(i-1)$ and $x(i)$ are involved. Consequently, this approach will increase efficiency of cache line usage, since we can reorganize the sequence of Givens rotations in groups with local effect on a small block of vector elements, as in [12]. The groups of rotations are determined by two parameters:

- Columns of the extended matrix $[\,B\;A\,]$ are grouped in blocks of n_b, which is the first parameter.
- Givens rotations $G_{i-1,i}^{(j)}$ determined by elements of columns j belonging to a block, are further organized in local groups similar to the group shown in Figure 3.2. They form slanted stripes in Figure 3.3. The second parameter is the width of the stripes, which is for practical reason chosen to be n_b.

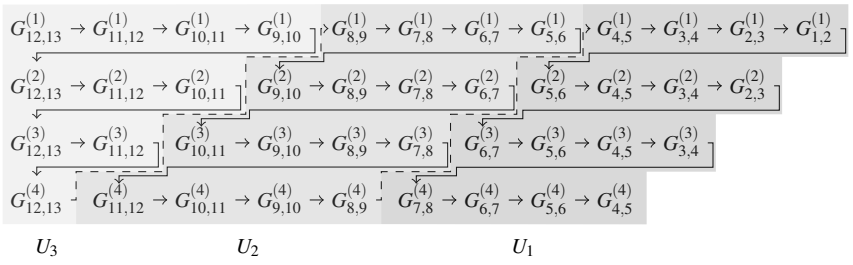


Fig. 3.3: Regrouping of Givens rotations from Figure 3.1, for $n = 13$ and $n_b = 4$.

The product of all rotations from a local group is denoted by U_k and represents the aggregated Givens rotations. The application of the whole sequence of Givens rotations to a vector or a matrix,

can now be performed by consecutive multiplication with matrices U_k. Hence, we can use Level 3 BLAS routines instead of Level 1 BLAS routines.

At first, only the left Givens rotations are accumulated in the matrices U_k. When the processing of the block is finished and all left updates are performed, the matrices U_k are reused for storing the aggregated right Givens rotations.

In case of the m-Hessenberg–triangular–triangular reduction the approach with the aggregated Givens rotations can be applied only to those blocks of matrices A, B and E which do not determine any rotations included in the matrices U_k, and to the whole matrix C. We will illustrate this with the example from Figure 3.3. Let $n = 13$, $m = p = 5$, and let one column block consist of $n_b = 4$ columns. In one outer loop step of the blocked algorithm all necessary elements in the current column block of the extended matrix $[\,B\;A\,]$ are annihilated. During this annihilation, only elements involved in determination of the next Givens rotation are updated. At the end of the outer loop step the remaining parts of involved matrices are updated using Level 3 BLAS operations. Two cases can occur: the current column block includes only columns of B, or it contains some columns of A. These situations are illustrated in Figures 3.4 and 3.5.

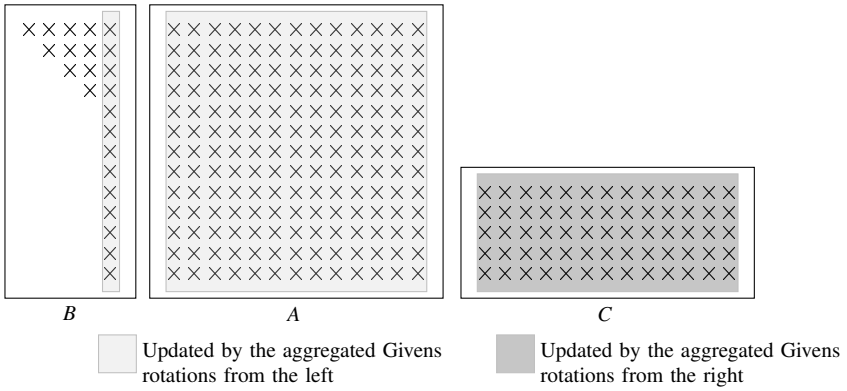


Fig. 3.4: Update after annihilation in the first column block of $[B\,A]$: only columns of B are involved.

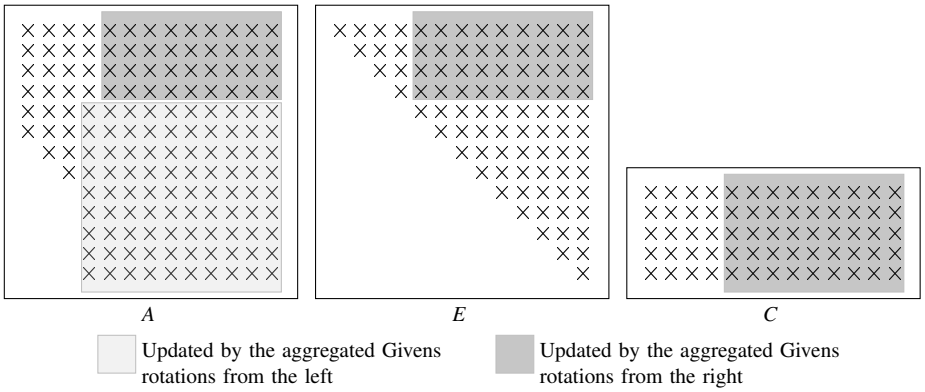


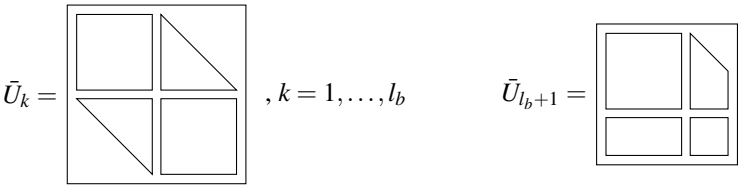
Fig. 3.5: Update after annihilation in the second column block of $[B\,A]$: columns of A are involved.

Now we resume the important facts about aggregated Givens rotations. Let us assume that the first j_c columns of the extended matrix $[\,B\;A\,]$ are already processed, and that all required elements in those columns are annihilated.

1. There is $l_b = \left\lfloor \frac{n-j_c}{n_b} \right\rfloor - 1$ regular groups in the current block (like U_1 and U_2 in Figure 3.3) and one starting group (U_3). Hence, U_k, $k = 1,\ldots,l_b, l_b+1$ have to be determined.
2. The form of matrices U_k is

$$U_k = \begin{bmatrix} I_{j_c+(k-1)n_b} & & \\ & \bar{U}_k & \\ & & I_{n-j_c-(k+1)n_b} \end{bmatrix},\ k = 1,\ldots,l_b, \quad U_{l_b+1} = \begin{bmatrix} I_{n-dim_0} & \\ & \bar{U}_{l_b+1} \end{bmatrix},$$

   where $\bar{U}_k \in \mathbb{R}^{2n_b \times 2n_b}$ for $k = 1,\ldots,l_b$, and $\bar{U}_{l_b+1} \in \mathbb{R}^{dim_0 \times dim_0}$ with $n_b \le dim_0 = n - j_c - l_b n_b < 2n_b$.
3. The form of matrices $\bar{U}_k$ is



4. In each step of the algorithm two Givens rotations are determined, one from the left and one from the right. Thus, for every rotation $G^{(j)}_{i-1,i}$ it is important to determine which local group it belongs to. This means that we have to determine the index k such that $G^{(j)}_{i-1,i}$ is a factor of U_k. It can be done in the following way.
   - If $i > n - dim_0 + j - j_c$ then $G^{(j)}_{i-1,i}$ is a factor of U_{l_b+1},
   - else, $k = \left\lfloor \frac{i-j-1}{n_b} \right\rfloor + 1$.

3.1 Delayed right update

As in [4] and [13], where the notion of "mini-block" is introduced, we can note that the Givens rotations $\tilde{G}^{(j)}_{n-1,n}$, $\tilde{G}^{(j)}_{n-2,n-1}$,..., $\tilde{G}^{(j)}_{j,j+1}$ determined to annihilate all subdiagonal elements of E introduced while annihilating elements in the j-th column of $[\,B\ A\,]$ ($(j-m)$-th column of A), when applied to A from the right have effect on elements which are at least m columns away from the current column. This means, that the update with Givens rotations from the right, as described in the non–blocked algorithm, can be performed after processing m columns of $[\,B\ A\,]$. Thus, each block of n_b columns (for $n_b > m$) is split further into smaller subblocks consisting of m consecutive columns. The last subblock may have less than m columns. Each column inside the current subblock has to be updated only with the aggregated Givens rotations from the left, before annihilating appropriate elements. After all columns of the subblock are processed, the Givens rotations from the right determined by the annihilation of the elements in this subblock are applied to A.

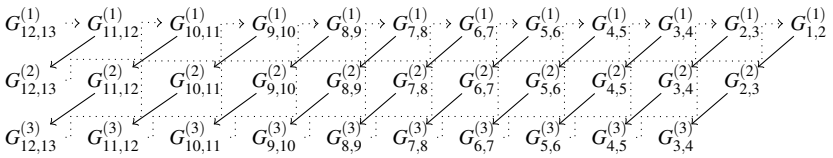


Fig. 3.6: The sequence of Givens rotations from the right determined by the annihilation of elements in a subblock, for $n = 13$ and $m = 3$.

The order of non-aggregated Givens rotations from the right can also be rearranged in a new sequence with emphasized local effect, inspired by the dependencies of rotations described in Figure 3.2. The adjacent rotations in this sequence will have effect on adjacent vector elements, as illustrated in Figure 3.6.

ALGORITHM 1: The blocked algorithm for reduction of matrices A, B, and E to the m-Hessenberg–triangular–triangular form.

Input: matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $E \in \mathbb{R}^{n\times n}$; block dimension n_b
Output: upper m-Hessenberg matrix $A = Q^\tau AZ$, upper triangular matrix $B = Q^\tau B$, matrix $C = CZ$, upper triangular matrix $E = Q^\tau EZ$; Q and Z can be accumulated if needed

Perform the QR factorization on E to determine an orthogonal matrix Q_E such that $E = Q_E^\tau E$ is upper triangular;
$A = Q_E^\tau A$; $B = Q_E^\tau B$;
$imb = 1$;
for $j_c = 0 : n_b : n-1$ **do**
    $j_{c,max} = \min\{j_c + n_b, n-1\}$; $n_b = \min\{n_b, n-1-j_c\}$;
    if $n_b = 0$ **then**
        return;
    end
    $l_b = \left\lfloor \frac{n-j_c}{n_b} \right\rfloor - 1$; $dim_0 = n - j_c - l_b n_b$;
    for $k = 1 : l_b$ **do**
        $\bar{U}_k = I_{2n_b}$;
    end
    $\bar{U}_{l_b+1} = I_{dim_0}$;
    for $j = j_c + 1 : j_{c,max}$ **do**
        for $i = n : -1 : j+1$ **do**
            Annihilate $B(i,j)$ or $A(i,j-m)$ if $j > m$ by a Givens rotation applied from the left;
            Update corresponding matrix $\bar{U}_k$ from the right, and matrix E from the left by this rotation;
            Annihilate $E(i,i-1)$ by a Givens rotation applied from the right, and update rows $j_c+1 : i-1$ of E by this rotation from the right;
        end
        If j is the last column of the subblock, then update $A(j_c+1:n, j-imb+1:n)$ by all Givens rotations from the right that were generated in the current subblock;
        If this is not the last column of the column block, then update $B(j_c+1:n, j+1)$ or $A(j_c+1:n, j-m+1)$ for $j \geq m$ by the aggregated rotations from the left;
    end
    Update $B(j_c+1:n, j_{c,max}+1:m)$ or $A(j_c+1:n, j_{c,max}-m+1:n)$ if $j_{c,max} \geq m$ by the aggregated rotations from the left;
    Generate the aggregated Givens rotations from the right and apply them to $A(1:j_c, j_c+1:n)$, $E(1:j_c, j_c+1:n)$, and $C(1:p, j_c+1:n)$;
end

4 Backward stability

The backward stability of the m-Hessenberg–triangular–triangular reduction follows directly from [11], and the proof is straightforward. First, we will state the notation and the results from [11] that we need, and then we are going to give a concluding backward error bound for our algorithm. In the standard model the accuracy of basic operations is expressed as

$$fl(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta), \qquad |\delta| \leq u, \quad \operatorname{op} = +, -, *, /$$

where $fl(\cdot)$ denotes a result computed in finite precision arithmetic, and u is the unit roundoff. For computations involving vectors and matrices it is convenient to introduce the parameter

$$\gamma_n = \frac{nu}{1-nu}.$$

The properties of this parameter are listed in Lemma 3.3 in [11]. Now we can state the intermediate results following from Theorem 19.4, Lemma 19.9 and Theorem 19.9 in [11], concerning the backward error of the QR factorization by the Householder reflectors and a product of a sequence of the Givens rotations with a general matrix. These results are obtained for the non-blocked algorithm, and for the blocked algorithm the results are similar but with different constants. Let us partition the algorithm in three basic steps.

1. The first step is QR factorization of E by the Householder reflectors, and for that we have the following relation
$$E+\Delta E_1 = Q_1 E_1, \qquad \|\Delta E_1\|_F \leq \gamma_{c_{hr}n^2}\|E\|_F,$$
where E_1 is upper triangular, Q_1 is orthogonal, and c_{hr} is a small integer constant whose exact value is unimportant. Q_1 is then applied to B and A resulting with
$$\begin{aligned} B+\Delta B_1 &= Q_1 B_1, \qquad \|\Delta B_1\|_F \leq \gamma_{c_{hr}n^2}\|B\|_F,\\ A+\Delta A_1 &= Q_1 A_1, \qquad \|\Delta A_1\|_F \leq \gamma_{c_{hr}n^2}\|A\|_F. \end{aligned}$$
2. Next, we accumulate effects of all left Givens rotations. The matrix B is affected only by those Givens rotations which participate in the QR factorization of B, and the matrices A and E are affected by all left rotations that participate in annihilation of the lower left triangular in $[B\ A]$. For a small integer constant c, we obtain
$$\begin{aligned} B_1+\Delta B_2 &= Q_{21}\hat{B}, \qquad \|\Delta B_2\|_F \leq \gamma_{c(n+m)}\|B_1\|_F \leq \gamma_{c(2n+2m)}\|B\|_F,\\ A_1+\Delta A_2 &= Q_2 A_2, \qquad \|\Delta A_2\|_F \leq \gamma_{2cn}\|A_1\|_F \leq \gamma_{4cn}\|A\|_F,\\ E_1+\Delta E_2 &= Q_2 E_2, \qquad \|\Delta E_2\|_F \leq \gamma_{2cn}\|E_1\|_F \leq \gamma_{4cn}\|E\|_F, \end{aligned}$$
where $\hat{B}$ is the final computed triangular form of B, Q_{21} and Q_2 are orthogonal, and $Q_2 = Q_{21}Q_{22}$.
3. Finally, we accumulate effects of all right Givens rotations, obtaining
$$\begin{aligned} A_2+\Delta A_3 &= \hat{A}Z^\tau, \qquad \|\Delta A_3\|_F \leq \gamma_{2cn}\|A_2\|_F \leq \gamma_{4cn}\|A\|_F,\\ E_2+\Delta E_3 &= \hat{E}Z^\tau, \qquad \|\Delta E_3\|_F \leq \gamma_{2cn}\|E_2\|_F \leq \gamma_{4cn}\|E\|_F, \end{aligned}$$
where $\hat{A}$ and $\hat{E}$ are the final computed m-Hessenberg and triangular forms of A and E respectively, and Z is orthogonal.

Putting all this together, we can conclude that $\hat{A}$, $\hat{B}$ and $\hat{E}$ represent the exact m-Hessenberg–triangular–triangular form of the matrices $A+\Delta A$, $B+\Delta B$ and $E+\Delta E$, such that for $Q = Q_1Q_2$ the following holds
$$\begin{aligned} \hat{A} &= Q^\tau(A+\Delta A)Z, && \|\Delta A\|_F \leq (\gamma_{c_{hr}n^2}+\gamma_{8cn})\|A\|_F,\\ \hat{E} &= Q^\tau(E+\Delta E)Z, && \|\Delta E\|_F \leq (\gamma_{c_{hr}n^2}+\gamma_{8cn})\|E\|_F,\\ \hat{B} &= Q^\tau(B+\Delta B), && \|\Delta B\|_F \leq (\gamma_{c_{hr}n^2}+\gamma_{c(2n+2m)})\|B\|_F. \end{aligned}$$

In case of the blocked algorithm, we have to analyze the accumulation and application of the aggregated Givens rotations $\bar{U}_k$. By Lemma 19.9 in [11] we have that the backward error for computed $\hat{U}_k$ satisfies the relation
$$\hat{U}_k = (I+\Delta I_k)\bar{U}_k, \qquad \|\Delta I_k\|_F \leq \gamma_{dn_b}\|I\|_F \leq \sqrt{2n_b}\gamma_{dn_b},$$
for a small integer constant d, since $\bar{U}_k$ can be factorized as $\bar{U}_k = W_1W_2\cdots W_{2n_b-1}$, and each W_i is a product of disjoint Givens rotations.

Let M_k denote a block of $2n_b$ (or $dim_0 < 2n_b$) consecutive rows of A, B or E affected by $\bar{U}_k$ from the left. In finite precision arithmetic, by section 3.5 in [11] on matrix multiplication we have the following relations

$$\begin{aligned}\hat{M}_k &= fl(\hat{U}_k^\tau M_k)\\ \hat{M}_k &= \hat{U}_k^\tau M_k + \Delta M_{k,1} = \bar{U}_k^\tau (I + \Delta I_k^\tau) M_k + \Delta M_{k,1}\\ &= \bar{U}_k^\tau (M_k + \Delta I_k^\tau M_k + \bar{U}_k \Delta M_{k,1}) = \bar{U}_k^\tau (M_k + \Delta M_k),\end{aligned}$$

where for a suitable constant e we get

$$\begin{aligned}\|\Delta M_{k,1}\|_F &\le \gamma_{2n_b} \|\hat{U}_k^\tau\|_F \|M_k\|_F \le \sqrt{2n_b}\,\gamma_{(2+d)n_b} \|M_k\|_F,\\ \|\Delta M_k\|_F &\le \sqrt{2n_b}\,\gamma_{(2+2d)n_b} \|M_k\|_F \le \gamma_{en_b} \|M_k\|_F.\end{aligned}$$

Following the proof of Lemma 19.9 in [11] for the Givens rotations, we can obtain the similar results for the aggregated Givens rotations. It can be easily shown that the matrix Q is of the form $Q = V_1 V_2 \cdots V_{2\lceil \frac{n-1}{n_b} \rceil - 2}$ where V_i is a product of disjoint aggregated Givens rotations. Hence, it follows that there exists a constant f such that

$$\begin{aligned}A_1 + \Delta A_2 &= Q A_2, \qquad \|\Delta A_2\|_F \le \gamma_{e\left(2\lceil \frac{n-1}{n_b} \rceil - 2\right) n_b} \|A_1\|_F \le \gamma_{fn} \|A_1\|_F,\\ E_1 + \Delta E_2 &= Q E_2, \qquad \|\Delta E_2\|_F \le \gamma_{e\left(2\lceil \frac{n-1}{n_b} \rceil - 2\right) n_b} \|E_1\|_F \le \gamma_{fn} \|E_1\|_F.\end{aligned}$$

The same result can be obtained for the right aggregated Givens rotations. Hence, we can conclude that the backward error bounds for the blocked algorithm are of the same form as for the non-blocked algorithm, but with different constants.

5 Shifted systems and transfer function

As mentioned in the introduction, the m-Hessenberg–triangular-triangular form is used as the first phase of an efficient solver of shifted systems, in particular when evaluating the transfer function $\mathcal{G}(\sigma) = C(\sigma E - A)^{-1} B + D$ of a descriptor system (1.1) at many values of complex scalar σ. In [4] it is described in details how to efficiently evaluate the transfer function in case when $E = I$, by using the controller Hessenberg (m-Hessenberg–triangular) form. Evaluation of $\mathcal{G}(\sigma)$ for a single shift is a straightforward generalization of Algorithm 11 in [4], which implements an incomplete RQ factorization by Householder reflectors. The most important part of the block algorithm implemented in [4, Algorithm 11] is illustrated in Algorithm 2. Matrices Y and V represent the WY representation of the aggregated Householder reflectors exploited in RQ factorization.

On the other hand, batch processing in case of multiple shifts described in Algorithm 12 in [4] for $E = I$ has to be remodeled in order to function for general matrices E. The auxiliary $(p+n) \times (m+n_b)$ array Z, introduced in Algorithm 2 which stores $(m+n_b)$ columns of the transformed matrices C and $A - \sigma I$ for a single shift, in [4] is split into two parts:

- The first part consists of the first n_b columns of Z and is denoted by Z_1. This part is (almost) the same for all shifts and relates to the original elements of A and C.
- The second part consists of the last m columns of Z that are specific to the shift σ_i, and is denoted by $Z_2(i)$, where $i = 1, \ldots, n_s$ and n_s is the number of shifts processed simultaneously. These submatrices are the result of the update applied to the shifted matrix from the previous outer loop step, and are different for different shifts.

ALGORITHM 2: The blocked algorithm for computing $\mathscr{G}(\sigma) = C(\sigma E - A)^{-1}B + D$.

Input: $(A,B,C,D,E) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{p\times n} \times \mathbb{R}^{p\times m} \times \mathbb{R}^{n\times n}$ ((A,B,E) in the m-Hessenberg–triangular-triangular form); $\sigma \in \mathbb{C}$, and block dimension n_b
Output: $\mathscr{G}(\sigma) \in \mathbb{C}^{p\times m}$

$l_b = \lfloor (n-m)/n_b \rfloor$;
$mink = n - (l_b - 1)\cdot n_b$;
Fill the first m columns of Z with the last m columns of $\left[\begin{smallmatrix} C \\ A-\sigma E \end{smallmatrix}\right]$;
for $k = n : -n_b : mink$ **do**
    Copy the first m columns Z to the last m columns of Z, and fill the first n_b columns of Z with columns $k-m-n_b+1,\ldots,k-m$ of $\left(\begin{smallmatrix} C \\ A-\sigma E \end{smallmatrix}\right)$;
    Compute V and Y by the RQ algorithm, such that $Z(p+k-n_b+1 : p+k, 1 : m+n_b)(I - YV^*)$ is upper triangular;
    Compute the first m columns of Z, by calling `xTRMM` and `xGEMM`:
    $Z(1 : p+k-n_b, 1 : m) = Z(1 : p+k-n_b, :)(I - YV^*)(:, 1 : m)$;
end
$l = n - l_b \cdot n_b$;
for $k = l : -1 : m+1$ **do**
    Perform the point algorithm;
end
Reduce the first m rows of $A - \sigma E$ to the triangular form $\hat{T}$ and simultaneously solve $\hat{T}\hat{X} = \hat{B}$;
$\mathscr{G}(\sigma) = D - Z(1 : p, 1 : m)\cdot \hat{X}$;

In case when $E = I$, only elements corresponding to diagonal elements of A stored in Z_1 are processed separately, since the shift $\sigma(i)$ has to be subtracted from them. With general E the situation is more complicated. The columns of E multiplied with $\sigma(i)$ have to be subtracted from the corresponding columns of A, in order to obtain columns of $A - \sigma(i)E$. Thus, besides columns of C and A, Z_1 has to store columns of E, too. $Z_2(i)$ is also extended to $p + 2n$ rows, where the additional rows accommodate intermediate results. Rows of Z_1 are organized as $\left[\begin{smallmatrix} C \\ A \\ E \end{smallmatrix}\right]$.

ALGORITHM 3: Details of the update in case of aggregated shifts.

for $k = n : -n_b : mink$ **do**
    Initialize data;
    for $1 = 1 : n_s$ **do**
        Perform necessary copying, and form Z_{block} which is input for the RQ algorithm;
6   Compute $V(i)$ and $Y(i)$ by the RQ algorithm, such that $Z_{block}(I - Y(i)V(i)^*)$ is upper triangular;
7.1   Call `xTRMM` to compute $Y(i)(1 : m+n_b, 1 : m) = Y(i)(1 : m+n_b, 1 : m)\cdot V(i)(1 : m, 1 : m)^*$;
7.2   Call `xGEMM` to compute $Z_2(i)(1 : p+k-n_b, 1 : m) = Z_1(1 : p+k-n_b, 1 : m)$
        $-Z_2(i)(1 : p+k-n_b, 1 : m)\cdot Y(i)(n_b+1 : n_b+m, 1 : m)$;
7.3   Set $Z_2(i)(p+k-n_b+1 : p+2k-2n_b, 1 : m) = Z_1(p+k-n_b+1 : p+2k-2n_b, 1 : m)$;
    end
7.4 Call `xGEMM` to compute
    $[\,Z_2(1)\ Z_2(2)\ \cdots\ Z_2(n_s)\,](1 : p+2k-2n_b, 1 : m\cdot ns) = [\,Z_2(1)\ Z_2(2)\ \cdots\ Z_2(n_s)\,](1 : p+2k-2n_b, 1 : m\cdot ns) - Z_1(1 : p+2k-2n_b, 1 : n_b)\cdot[\,Y(1)(1 : n_b, 1 : m)\ Y(2)(1 : n_b, 1 : m)\ \cdots\ Y(n_s)(1 : n_b, 1 : m)\,]$;
    for $1 = 1 : n_s$ **do**
7.3 $Z_2(i)(1 : p+1 : p+k-n_b, 1 : m) = Z_2(i)(1 : p+1 : p+k-n_b, 1 : m)$
        $-\sigma(i)Z_2(i)(p+k-n_b+1 : p+2k-2n_b, 1 : m)$;
    end
end

The main idea of batch processing is to perform the update of Z_1 simultaneously for all shifts, as one call to the BLAS 3 routine `xGEMM`. That is what we are going to do for general E, but this update will be less efficient than the one for the case with $E = I$. The update for general E includes extra rows of the matrix E, and hence has larger operation count. At the end of this process, rows

corresponding to updated rows of E multiplied by the shift $\sigma(i)$ are subtracted from the corresponding updated rows of A. As a result, the update described in line 7 of Algorithm 2 is transformed to the sequence of BLAS 3 operation described in Algorithm 3.

Here we assume that $m \leq n_b$. $Y(i)$ stores the product $Y(i)(1:m+n_b, 1:m) \cdot V(i)(1:m, 1:m)^*$, where matrices $Y(i)$ and $V(i)$ represent the WY representation of the aggregated Householder reflectors determined in the current outer loop step for the shift $\sigma(i)$. For details, see subsection 4.2 in [4].

6 The staircase form

The m-Hessenberg–triangular–triangular reduction algorithm can be efficiently applied to computation of a useful canonical form, used to determine whether the descriptor system (1.1) is controllable or not, or to identify its controllable part. Such a canonical form is a staircase form, which can be computed with a numerically stable algorithm, and is described in [19]. For the nonsingular matrix E, the staircase form directly reveals controllability of the system.

ALGORITHM 4: The standard staircase reduction algorithm.

Input: the system $(A,B,E) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{n\times n}$
Output: the system $(\hat{A},\hat{B},\hat{E})$ in the staircase form

1 Compute the RRD $B = U \begin{bmatrix} \bar{B} \\ 0 \end{bmatrix}$, where $\bar{B}$ has full row rank n_1;
2 Update from left $A = U^\tau A$, $E = U^\tau E$;
3 Compute the RQ factorization $E = EV^\tau$;
4 Update from right $A = AV^\tau$;
5 Set $prev = 0$, $curr = n_1$, $i = 1$;
6 **while** $curr < n$ **do**
7 Set $Z = A(curr+1:n, prev+1:prev+n_i)$;
8 **if** *Z is a zero matrix* **then**
9 break: the system is uncontrollable;
10 **end**
11 Compute the RRD $Z = U \begin{bmatrix} \bar{Z} \\ 0 \end{bmatrix}$, where $\bar{Z}$ has full row rank n_{i+1};
12 Update from left: $A(curr+1:n, prev+1:n) = U^\tau A(curr+1:n, prev+1:n)$,
 $E(curr+1:n, curr+1:n) = U^\tau E(curr+1:n, curr+1:n)$;
13 Compute the RQ factorization $E(curr+1:n, curr+1:n) = E(curr+1:n, curr+1:n)V^\tau$;
14 Update from right: $A(1:n, curr+1:n) = A(1:n, curr+1:n)V^\tau$,
 $E(1:curr, curr+1:n) = E(1:curr, curr+1:n)V^\tau$;
15 $prev = curr$, $curr = curr + n_{i+1}$, $i = i+1$;
16 **end**
17 Set $\hat{A} = A$, $\hat{B} = \begin{bmatrix} \bar{B} \\ 0 \end{bmatrix}$, $\hat{E} = E$;

For any matrices $A, E \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ there exist orthogonal matrices $Q, Z \in \mathbb{R}^{n\times n}$ such that

$$Q^\tau E Z = \begin{bmatrix} E_{11} & E_{12} & \cdots & E_{1k} \\ 0 & E_{22} & \cdots & E_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_{kk} \end{bmatrix}, \tag{6.1}$$

$$\left[Q^\tau B \middle| Q^\tau A Z \right] = \left[\begin{array}{c|ccc} A_{10} & A_{11} & A_{12} & \cdots & A_{1k} \\ 0 & A_{21} & A_{22} & \cdots & A_{2k} \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \cdots & A_{k,k-1} & A_{kk} \end{array}\right], \tag{6.2}$$

where $k \leq n$, E_{ii} and A_{ii} are of order $n_i \times n_i$ for $i = 1, \ldots, k$, $A_{i,i-1}$ are of order $n_i \times n_{i-1}$ and have full row rank n_i for $i = 1, \ldots, k-1$, and $n_0 = m$. Then, if

- $A_{k,k-1}$ has rank $n_k > 0$, the system (1.1) is controllable,
- $A_{k,k-1} = 0$, the system is not controllable.

This approach can be easily extended to systems with the singular matrix E, with a few additional steps. Even in this case, reduction to the staircase form (6.1)–(6.2) constitutes its most important part, see [5] and [25]. The singularity of the matrix E is not important for the actual reduction algorithm. The staircase reduction is also used as a preprocessing phase when solving the pole (eigenvalue) assignment problem by state feedback [16].

If the matrices A and E have no structure, then a common way of computing the staircase form is shown in Algorithm 4, which is a generalization of Algorithm 8 from [4] (see also [19]).

Here RRD denotes a rank-revealing decomposition of a matrix, and for that usually a rank-revealing QR-factorization is used. This algorithm performs $\mathcal{O}(n^4)$ floating point operations, and its efficiency substantially depends on dimensions n_i which represent its blocking dimensions. Varga in [25] proposes a combination between the non–blocked m-Hessenberg–triangular–triangular reduction algorithm described in section 2 and Algorithm 4 with only $\mathcal{O}(n^3)$ operations. Each RRD is performed by QR factorization with column pivoting, where in the reduction algorithm elements are annihilated one by one from the left, followed immediately by correction of the introduced subdiagonal element in E from the right. We propose a more efficient version of the staircase reduction algorithm which combines the blocked m-Hessenberg–triangular–triangular reduction implemented in Algorithm 1 with Algorithm 4. Since for small m, $n_i \leq m$ represent blocking dimensions for Algorithm 4, the idea is to use the blocking strategy of Algorithm 1, but with variable band width of the reduced matrix A. On the other hand, after a column block $A(curr+1:n, prev+1:prev+n_i)$ is reduced to the n_i-Hessenberg form, the RRD is performed on the small $n_i \times n_i$ diagonal block with any suitable algorithm providing that eventual column pivoting interchanges small dimensional columns. The obtained algorithm is quite complicated since the band width of the reduced A and the blocking dimension are not fixed any more. The maximal blocking dimension n_b is predetermined as in Algorithm 1, but the actual dimension of each block is tailored to include the maximal number of whole subblocks with variable dimensions n_i. After every RRD, the dimension of the next subblock is determined by the computed rank, as well as the band width of its n_i-Hessenberg reduction. The new staircase reduction algorithm is described in Algorithm 5.

ALGORITHM 5: The new staircase reduction algorithm via blocked m-Hessenberg–triangular–triangular reduction algorithm — part 1.

Input: the system $(A,B,E) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m} \times \mathbb{R}^{n\times n}$
Output: the system $(\hat{A},\hat{B},\hat{E})$ in the staircase form

Perform the QR factorization on E to determine the orthogonal matrix Q_E such that $E = Q_E^\tau E$ is upper triangular;
$A = Q_E^\tau A$; $B = Q_E^\tau B$;
$j_c = 0$; $prev = -m$; $curr = 0$; $mm = m$;
while $curr < n$
    `// if no subblock fits into the block step out of the loop.`
    if $j_c + mm > n-1$ **then**
        break;
    end
    $os = mm - m$; $n_b = \min\{n_b, n-1-j_c\}$; $l_b = \lfloor (n-j_c-os)/n_b \rfloor - 1$; $dim_0 = n - j_c - os - l_b n_b$;
    for $k = 1 : l_b$ **do**
        $\bar{U}_k = I_{2n_b}$;
    end
    $\bar{U}_{l_b+1} = I_{dim_0}$;

ALGORITHM 5: The new staircase reduction algorithm via blocked m-Hessenberg–triangular–triangular reduction algorithm — part 2.

```
12
13    j = j_c + 1;
14    while true do
15        for j_sb = 1 : mm do
16            for i = n : −1 : j + os + 1 do
17                Annihilate B(i, j) or A(i, j − m) if j > m by a Givens rotation applied from the left;
18                Update corresponding matrix Ū_k from the right, and matrix E from the left by this rotation;
19                Annihilate E(i, i − 1) by a Givens rotation applied from the right, and update rows
                  j_c + os + 1 : i − 1 of E by this rotation from the right;
20            end
21            If j_sb < mm and this is not the last column of the subblock, then update B(j_c + 1 : n, j + 1) or
              A(j_c + 1 : n, j − m + 1) for j ≥ m by the aggregated rotations from the left;
22            j = j + 1;
23        end
          // The subblock is reduced to the mm-Hessenberg form.
24        Set Z = B(1 : m, 1 : m) if j = m + 1, or else Z = A(curr + 1 : curr + mm, prev + 1 : prev + mm);
25        Compute the RRD Z = P [Z̄; 0], where Z̄ has full row rank nmm;
26        if nmm = 0 then
27            break: the system is uncontrollable;
28        end
29        if nmm < mm then
30            In case of a rank drop set B(1 : m, 1 : m) = [Z̄; 0] if j = m + 1, or else
              A(curr + 1 : curr + mm, prev + 1 : prev + mm) = [Z̄; 0];
31            Update Ū_1 by P from the right, and matrix E by P^τ from the left;
32            Compute the RQ factorization
              E(curr + 1 : curr + mm, curr + 1 : curr + mm) = E(curr + 1 : curr + mm, curr + 1 : curr + mm)Q, and
              store Q;
33        end
34        Update A(j_c + os + 1 : n, curr + 1 : n) by all Givens rotations from the right that were generated in the
          current subblock, and in case of rank drop update A(j_c + os + 1 : n, curr + 1 : curr + mm) by Q^τ from the
          right;
35        if m + curr + nmm − j_c > n_b then
              // The next subblock does not fit into the block, step out of the loop.
36            prev = curr; curr = curr + nmm; mm = nmm;
37            break;
38        else
39            Update the first column of the next subblock A(j_c + 1 : n, j − m) by the aggregated rotations from the
              left;
40        end
41        prev = curr; curr = curr + nmm; mm = nmm;
42    end
43    j_max = j − 1;
44    Update A(j_c + os + 1 : n, max(prev, 0) + 1 : n) by the aggregated rotations from the left;
45    Generate the aggregated Givens rotations from the right and include the stored matrices Q from RQ
      factorizations;
46    Apply the aggregated Givens rotations to A(1 : j_c + os, j_c + os + 1 : n) and E(1 : j_c + os, j_c + os + 1 : n);
47    j_c = j_max;
48    If the system is uncontrollable exit the routine;
49 end
50 Apply Algorithm 4 to A(curr + 1 : n, prev + 1 : n) and E(curr + 1 : n, curr + 1 : n);
```

7 The GPU algorithm

The blocked CPU algorithm for m-Hessenberg–triangular–triangular reduction is not completely blocked. The only part that is left unblocked is the sequence of consecutive determination of the left

and right Givens rotations, since we cannot change the order in this sequence. On the other hand, the total time spent on:

1. determination of the left Givens rotation and update of the corresponding aggregated Givens rotation $\bar{U}_k$,
2. left update of E,
3. determination of the right Givens rotation and right update of E

in all inner–loop steps takes 51-63% of execution time of the whole reduction. This is the place in the algorithm which we tried to improve in our GPU version, and we managed to parallelize it successfully. The details of the GPU algorithm for the m-Hessenberg–triangular–triangular reduction will be published in another paper, devoted only to GPU algorithms. Here, we will only report that the GPU algorithm is up to two times faster than the CPU algorithm.

8 Numerical tests

In this section we illustrate better efficiency of the blocked algorithm for the m-Hessenberg–triangular–triangular reduction compared to the non–blocked version of the same algorithm, and the routine `TG01BD` of the control and systems library SLICOT [22]. SLICOT does not include any routine for evaluation of the transfer function of a descriptor system. There only exists routine `TG01BD` which reduces the matrices A and E to the Hessenberg–triangular form using a non–blocked algorithm based on Givens rotations ([9, Algorithm 7.7.1].

Extensive numerical test were performed on random matrices with different sizes of n, m, p and n_b, and on different platforms. Here we present test results obtained on the architecture whose hardware and software specifications are listed in Table 8.1.

processor	Intel® Xeon® X5470 (quad-core)
Frequency	3.33 GHz
Cache (Level 2)	$6+6$ MB
RAM	8 GB
Operating system	Ubuntu Linux 8.10
Compiler	Intel® Fortran Compiler Version 11.0
Optimization flag	-O3
BLAS	Intel® Math Kernel Library 10.1

Table 8.1: Hardware and software specifications of the test platform.

All tests were run using the IEEE double precision arithmetic. The `TG01BD` routine was executed without forming orthogonal factors. Size n of the matrices A and E ranges from 100 to 6000 with the step of 100. m — the number of columns of the matrix B, and p — the number of rows of the matrix C are chosen to be $m = p = 1, 5, 10, 15, 20, 100$. All matrices are generated with random elements. The block size n_b for each dimension setup is chosen to be optimal in sense that it produces the best execution time of our blocked algorithm. It turned out to be $n_b = 32$ for $n < 1400$, and $n_b = 64$ otherwise. The test results are shown in Figure 8.1.

As we can see, we obtained almost identical speedup graphs for any choice of m and p, since the non–blocked m-Hessenberg–triangular–triangular reduction has almost the same execution time as `TG01BD`. The crucial role plays dimension n: for $m > 1$ and for $n = 100$ up to $n = 1000, 1200$, the efficiency of our routine rapidly grows compared to the SLICOT routine. The peek efficiency is obtained for $n = 1100$ and $m = 100$ where our blocked algorithm for the m-Hessenberg–triangular–triangular reduction was at most 3.3906 times faster than `TG01BD`. For larger dimensions n there is a drop in our efficiency, but the speed-up factor stabilizes between 2.6 and 2.8, for $m > 1$. Our blocked

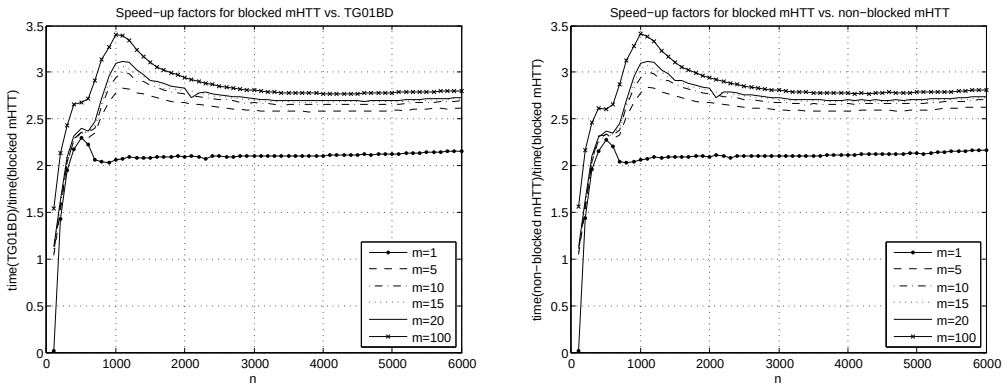


Fig. 8.1: Speed–up factors for the blocked m-Hessenberg–triangular–triangular reduction vs. `TG01BD`, and for the blocked m-Hessenberg–triangular–triangular reduction vs. its non–blocked version.

algorithm was least efficient for $m = 1$ as expected, since the right updates have to be performed after processing of each column.

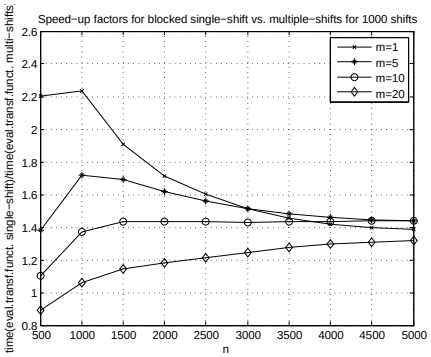
The test for two applications of m-Hessenberg–triangular–triangular form were also performed. Again, we used double precision and double complex arithmetic. When evaluating the transfer function we compared execution times of the single shift and the multiple shift algorithms for $r = 1000$ shifts, on random matrices with dimensions n ranging from 500 up to 5000 with the step 500, and $m = 1,5,10,20$. For both algorithms we chose the optimal block dimension n_b, offering the best execution time. For the single shift algorithm it turned out to be $n_b = 32$ for all dimensions, except for $n = 500$ and $m = 1,5$. For the multiple shifts algorithm the optimal n_b varies between 32, 64 and 96 for $n <= 2000$, and 64, 96 and 128 for $n > 2000$. The number of shifts n_s that are simultaneously processed is also chosen to be optimal, and it varies between 128, 256 and 512. $n_s = 512$ predominates for larger dimensions $n \geq 3000$. The results are presented in Figure 8.2a. The speed-up factor stabilizes round 1.4 for larger dimensions. Thus, we can conclude that the multiple shifts algorithm for the evaluation of the transfer function for the general E is half as efficient as the corresponding algorithm for $E = I$, as expected. Still, we gained some speed-up. Besides the new algorithms, we also implemented the standard algorithm for transfer function evaluation performing only the Hessenberg–triangular reduction. This implementation is similar to the SLICOT routine `TB05AD` when $E = I$. The routines were executed as follows.

– Our routines: the reduction to the m-Hessenberg–triangular–triangular form was executed once followed by executions of our blocked algorithm for computing $\mathscr{G}(\sigma_i), i = 1,\ldots,1000$ with aggregated shifts;

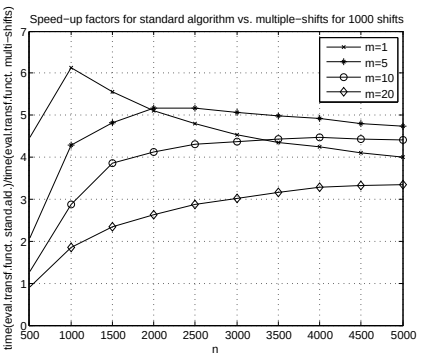
– The standard algorithm in SLICOT style: the routine was first executed with the parameter `INITIA='G'`, indicating that the matrices A and E are general matrices. This was followed by 999 executions of the same routine with `INITIA='H'` indicating that the matrices A and E are in the upper Hessenberg–triangular form, which is computed using non-blocked `xGGHRD` from LAPACK [2] in the first step.

The speed-up factors gained by our routines compared to the standard algorithm are presented in Figure 8.2b. The results are similar to those presented in [4] for $E = I$, but the factors are approximately halved, since the efficiency of the algorithm for computing the transfer function with aggregated shifts for general E is less efficient than the one for $E = I$. Our routines are at most 6 times faster than the standard algorithm for $n = 1000$ and $m = 1$, but for the larger n-s the speed-up

factors stabilize between 3 and 5. Hence, our approach is much more efficient than the standard one, when evaluating the transfer function for a larger number of shifts.



(a) Speed-up factors for blocked single-shift algorithm for the evaluation of the transfer function vs. multiple-shifts algorithm. Both algorithms process 1000 shifts.



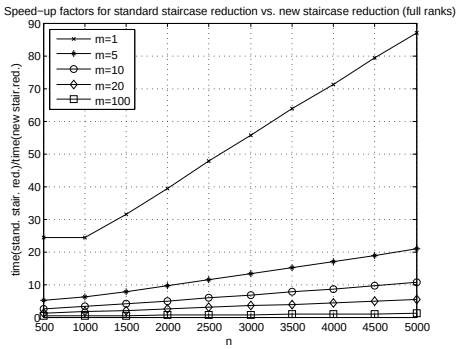
(b) Speed-up factors for the standard algorithm for the evaluation of the transfer function vs. multiple-shifts algorithm. Both algorithms process 1000 shifts.

Fig. 8.2: The evaluation of the transfer function.

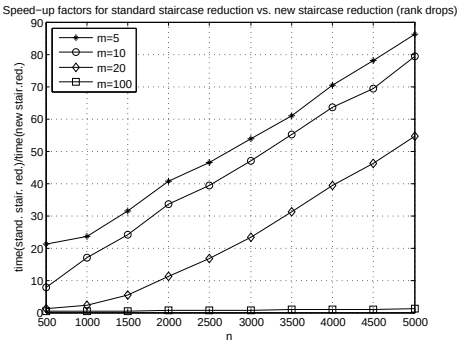
For the staircase reduction, first we compared the execution times of two algorithms implementing Algorithm 4 and Algorithm 5. Both algorithms include the accumulation of the orthogonal matrices Q and Z. Dimensions n are chosen as in case of the evaluation of the transfer function, and $m = p = 1,5,10,20,100$. We performed two sets of tests. In all tests we chose the optimal block dimension n_b in implementation of Algorithm 5, which turned out to be $n_b = 32$ for $n \leq 500$, $n_b = 64$ for $1000 \leq n \leq 2000$, $n_b = 96$ for $2500 \leq n \leq 4500$, and $n = 128$ for $n \geq 5000$. In the first test round all the matrices were random matrices, and there were now rank drops in RRD-s in both algorithms. The results are presented in Figure 8.3a. As we can see in this case, the new staircase reduction implemented in Algorithm 5 is much more efficient for smaller m-s than the standard approach. The reason for this is the fact that m can be interpreted as a block dimension in Algorithm 4, and for larger m-s this algorithm is faster. The execution times of Algorithm 5 for fixed n varies little when changing m. We obtain speed-up factors over 80 for $m = 1$, over 20 for $m = 5$, over 10 for $m = 10$, and over 5 for $m = 20$. In case of $m = 100$ the new algorithm is faster than the standard one only when $n > 4000$, since the blocking dimension for Algorithm 4 is larger and the aggregated Householder reflectors employed in this algorithm are more efficient than the aggregated Givens rotation in Algorithm 5. It is worth mentioning here that the execution times of Algorithm 5 are comparable with the execution times of the m-Hessenberg–triangular–triangular reduction algorithm implemented in Algorithm 1 with the same parameters n and m.

In the second set of tests the systems have predefined ranks n_i in their staircase form (6.1)–(6.2). Rank drop of 1 occurs for every fifth RRD, until n_i reaches the value 1. The results are shown in Figure 8.3b. The execution times of Algorithm 5 vary between the execution times for the given m and $m = 1$ of the same algorithm when performed without the rank drops.

Next, we compare execution times of the SLICOT routine `TG01HX` which implements the algorithm from [25] and Algorithm 5. In all tests we chose the same dimensions n, m and p as in the previous test rounds, and the optimal block dimension n_b for Algorithm 5. The results presented in Figure 8.4 reveal that Algorithm 5 is up to 5.7631 times faster than `TG01HX`. The speed-up factors stabilizes between 5.28 and 5.45 for larger n and all $m > 1$. This speed-up graph resembles the graphs in Figure 8.1, since the largest influence on the speed-up has the better efficiency of blocked m-Hessenberg–triangular–triangular reduction over its non–blocked version.



(a) Speed-up factors for the staircase reduction implemented in Algorithm 4 vs. Algorithm 5, with full ranks n_i.



(b) Speed-up factors for the staircase reduction implemented in Algorithm 4 vs. Algorithm 5, with rank drops.

Fig. 8.3: The evaluation of the staircase reduction algorithms.

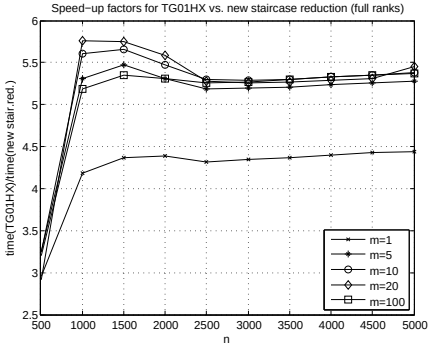


Fig. 8.4: Speed-up factors for the staircase reduction implemented in `TG01HX` vs. Algorithm 5, with full ranks n_i.

9 Conclusion

We are proposing a new blocked algorithm which ensures efficient evaluation of the transfer function $\mathscr{G}(\sigma) = C(\sigma E - A)^{-1}B + D$. The algorithm simultaneously reduces A to the m-Hessenberg form, and B and E to the triangular form, thus enabling efficient computation of the transfer function $C(\sigma E - A)^{-1}B$ for large number of shifts σ, as described in [4]. The non–blocked version of the algorithm for reduction to the m-Hessenberg–triangular–triangular form has execution time comparable to the execution time of the SLICOT routine `TG01BD`. On the other hand, the blocked version using the aggregated Givens rotations is much faster, up to 3.4 times compared to the non–blocked algorithms. Our algorithm for evaluation of the transfer function is up to 6 times faster than the standard one based only on the Hessenberg–triangular reduction of A and E. Besides the transfer function evaluation, the blocked algorithm for the m-Hessenberg–triangular–triangular reduction is efficiently incorporated into another reduction algorithm which computes the staircase form used to identify the controllable part of the descriptor system. Our staircase reduction algorithm is up to 80 times faster than the standard approach, and up to 5.8 times faster than the SLICOT routine `TG01HX`.

References

1. K. Ahuja, E. de Sturler, R. Chang and S. Gugercin, *Recycling BiCG with an application to model reduction*, SIAM J. Sci. Comput., 34, pp. A1925-A1949 (2012).
2. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, and D. C. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia (1999).
3. C.A. Beattie, Z. Drmač, and S. Gugercin, *A note on shifted Hessenberg systems and frequency response computation*, ACM Trans. Math. Softw., 38, pp. 12:1–12:16 (2011).
4. N. Bosner, Z. Bujanović, and Z. Drmač, *Efficient generalized Hessenberg form and applications*, ACM Trans. Math. Softw., 39 (2013).
5. K.-W. E. Chu, *A controllability condensed form and a state feedback pole assignment algorithm for descriptor systems*, IEEE Trans. Autom. Control, 33, pp. 366–370 (1988).
6. K. Dackland and B. Kågström, *Blocked algorithms and software for reduction of a regular matrix pair to generalized Schur form*, ACM Trans. Math. Softw., 25, pp. 425–454 (1999).
7. J. J. Dongarra, J. J. Du Croz, I. Duff, and S. Hammarling, *A set of Level 3 basic linear algebra subprograms*, ACM Trans. Math. Soft., 16, pp. 1-17 (1990).
8. M. B. van Gijzen, Y. A. Erlangga, and C. Vuik, *Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian*, SIAM J. Sci. Comput., 29, pp. 1942-1958 (2007).
9. G. H. Golub and C. F. van Loan, *Matrix Computations*, Third Edition, M. D. Johns Hopkins University Press, Baltimore (1996).
10. S. Gugercin, A.C. Antoulas and C.A. Beattie, *H2 model reduction for large-scale linear dynamical systems*, SIAM J. Matrix Anal. Appl., 30, pp. 609–638 (2008).
11. N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., Philadelphia, SIAM. (2002).
12. B. Kågström, D. Kressner, E. S. Quintana-Ortí, and G. Quintana-Ortí, *Blocked algorithms for the reduction to Hessenberg-triangular form revisited*, BIT, 48, pp. 563–584 (2008).
13. L. Karlsson and B. Kågström, *Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures*, Parallel Computing, 37, pp. 771–782 (2011).
14. B. Lang, *Using level 3 BLAS in rotation-based algorithms*, SIAM J. Sci. Comput., 19, pp. 626–634 (1998).
15. A. J. Laub and A. Linnemann, *Hessenberg and Hessenberg/triangular forms in linear system theory*, Int. J. Control, 44, pp. 1523–1547 (1986).
16. G. S. Miminis, *Deflation in eigenvalue assignment of descriptor systems using state feedback*, IEEE Trans. Autom. Control, 38, pp. 1322–1336 (1993).
17. G. S. Miminis and C. C. Paige, *An algorithm for pole assignment of time invariant linear systems*, Int. J. Control, 35, pp. 341–354 (1982).
18. G. S. Miminis and C. C. Paige, *An algorithm for pole assignment of time invariant multi-input linear systems*, Proceedings 21st IEEE Conference on Decision and Control, pp. 62–67 (1982).
19. C. C. Paige, *Properties of numerical algorithms related to computing controllability*, IEEE Trans. Autom. Control, 26, pp. 130–138 (1981).
20. V. Simoncini, *Restarted full orthogonalization method for shifted linear systems*, BIT, 43, pp. 459–466 (2003).
21. V. Simoncini and F. Perotti, *On the numerical solution of* $(\lambda^2 A+\lambda B+C)x=b$ *and application to structural dynamics*, SIAM J. Scientific Comput., 23, pp. 1876-1898 (2002).
22. *SLICOT*, `http://www.slicot.org`
23. P. M. van Dooren and M. Verhaegen, *On the use of unitary state-space transformations*, Linear algebra and its role in systems theory, Proc. AMS-IMS-SIAM Conf., Brunswick/Maine 1984, Contemp. Math. 47, pp. 447–463 (1985).
24. A. Varga, *Numerical algorithms and software tools for analysis and modelling of descriptor systems*, Prepr. of 2nd IFAC Workshop on System Structure and Control, Prague, Czechoslovakia, pp. 392–395 (1992).
25. A. Varga, *Computation of irreducible generalized state-space realizations*, Kybernetika, 26, pp. 89–106 (1990).
26. A. Varga and P. M. Van Dooren, *Task I.A - Basic software tools for standard and generalized state-space systems and transfer matrix factorizations*, SLICOT Working Note 17 (1999).

A The algorithm details

In this appendix we describe the blocked algorithm for reduction of matrices A, B, and E to the m-Hessenberg–triangular–triangular form in more details, and we start with notation. Let us denote by $[c,s,r]=\texttt{givens}(a,b)$ a routine that computes $c=\cos\phi$ and $s=\sin\phi$ which determine a Givens rotation such that

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}^{\tau} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix},$$

where $r=\sqrt{a^2+b^2}$. Next, let us denote by $[x,y]=\texttt{givens\_update}(c,s,x,y)$ a routine that applies a Givens rotation determined by $c=\cos\phi$ and $s=\sin\phi$ to a pair of row or column vectors:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}^{\tau} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{if } x \text{ and } y \text{ are row vectors}$$

$$\begin{bmatrix} x & y \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \qquad \text{if } x \text{ and } y \text{ are column vectors}$$

Besides the routine `givens_update`, for the blocked algorithm we need a routine that applies a sequence of Givens rotations to a matrix. Since the rotations applied from the left and from the right are generated simultaneously, and at the beginning of the algorithm only aggregated rotations from the left are used, cosines and sines of the rotations applied from the right, generated in one column block sweep are stored into two-dimensional fields *CC* and *SS*. When the matrices U_k are not used for rotations applied from the left any more, they are reused for storing aggregated rotations from the right. Before that happens, we have to update the matrix A after every m columns with the sequence of Givens rotations from the right, generated in one subblock. To be precise, if j is the last column of the subblock, then we have to apply the Givens rotations from the right to $A(j_c+1:n, j-m+1:n)$. For that we use $X = \texttt{seq\_givens\_update}(cc, ss, X)$, which is similar to the LAPACK routine `dlasr`. This routine updates a matrix $X \in \mathbb{R}^{p\times r}$ with the following sequence of Givens rotations from the right:

$$P = \tilde{G}^{(1)}_{r-1,r}\tilde{G}^{(1)}_{r-2,r-1}\cdots\tilde{G}^{(1)}_{1,2}\tilde{G}^{(2)}_{r-1,r}\tilde{G}^{(2)}_{r-2,r-1}\cdots\tilde{G}^{(2)}_{2,3}\cdots\tilde{G}^{(m)}_{r-1,r}\tilde{G}^{(m)}_{r-2,r-1}\cdots\tilde{G}^{(m)}_{m,m+1}.$$

$(r-1)\times m$ fields *cc* and *ss* contain cosines and sines for the Givens rotations $\tilde{G}^{(j)}_{k,k+1}$ from the sequence, where

$$\tilde{G}^{(j)}_{k,k+1} = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ k \\ k+1 \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & cc(k,j) & -ss(k,j) & & & \\ & & & ss(k,j) & cc(k,j) & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \\ \\ k \qquad k+1 \end{array}$$

The matrix X is transformed to $X = XP$ where P is an orthogonal matrix obtained as the product of the Givens rotations. Hence, the Givens rotations are reordered as illustrated in Figure 3.6.

The aggregated Givens rotations from the right are generated when all rotations are available. We can generate the matrices U_k one at a time, and the rotations are multiplied in the order similar to the product generated by `seq_givens_update`.

We need some extra memory storage for auxiliary variables in the blocked algorithm, such as $\bar{U}_1, \ldots, \bar{U}_{l_b+1}$, *CC* and *SS*. Dimensions of this variables are given in the following table:

Variable	**Dimension**
$\bar{U}_k$, $k = 1, \ldots, l_b$	$2n_b \times 2n_b$
$\bar{U}_{l_b+1}$	$dim_0 \times dim_0$
CC	$n \times n_b$
SS	$n \times n_b$

where $l_b = \left\lfloor \frac{n}{n_b} \right\rfloor - 1$ and $dim_0 = n - l_b n_b$.

A.1 Form of the aggregated Givens rotations U_k

As we see in Figure 3.3 the starting local group of Givens rotations forming U_3 (U_k with the largest index) is a bit different from all other local groups. The groups forming U_1 and U_2 consist of the same number of rotations and have a similar form. We can call them as in [14] "regular groups". Let us illustrate the accumulation of U_2 for the example with $n = 13$ and $n_b = 4$ first (see [12]). The Givens rotation $G^{(j)}_{i-1,i}$ affects only the $i-1$-th and i-th coordinates, for any j. In Figure 3.3 we see that among all Givens rotations forming U_2 the one with the smallest index i is $G^{(1)}_{5,6}$, and with the largest index i is $G^{(4)}_{11,12}$. Thus, U_2 affects only coordinates $5, 6, \ldots, 12$, and has the basic form

$$U_2 = \begin{bmatrix} I_4 & & \\ & \bar{U}_2 & \\ & & 1 \end{bmatrix},$$

where $I_4 \in \mathbb{R}^{4\times 4}$ is the identity matrix, and $\bar{U}_2 \in \mathbb{R}^{8\times 8} = \mathbb{R}^{2n_b\times 2n_b}$. Further, we illustrate the form of $\bar{U}_2$ in Figure A.1.

- Elements denoted by $j = 1$ are introduced by multiplication $G^{(1)}_{8,9}G^{(1)}_{7,8}G^{(1)}_{6,7}G^{(1)}_{5,6}$.
- Elements denoted by $j = 2$ are introduced by postmultiplication with $G^{(2)}_{9,10}G^{(2)}_{8,9}G^{(2)}_{7,8}G^{(2)}_{6,7}$.
- Elements denoted by $j = 3$ are introduced by postmultiplication with $G^{(3)}_{10,11}G^{(3)}_{9,10}G^{(3)}_{8,9}G^{(3)}_{7,8}$.

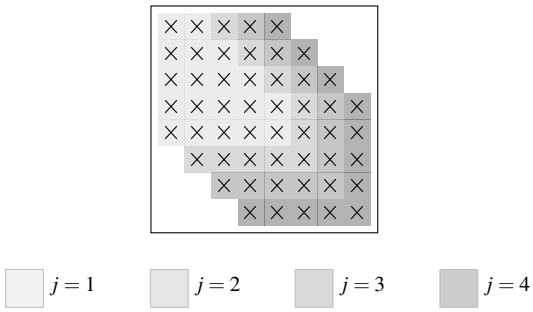


Fig. A.1: Form of $\bar{U}_2$.

– Elements denoted by $j = 4$ are introduced by postmultiplication with $G^{(4)}_{11,12} G^{(4)}_{10,11} G^{(4)}_{9,10} G^{(4)}_{8,9}$.

U_1 also affects 8 coordinates: $1, 2, \ldots, 8$, with the basic form

$$U_1 = \begin{bmatrix} \bar{U}_1 & \\ & I_5 \end{bmatrix},$$

where $\bar{U}_1$ has the same form as $\bar{U}_2$ shown in Figure A.1.

On the other hand, the matrix U_3 affects only 5 coordinates: $9, 10, \ldots, 13$, and has the form

$$U_3 = \begin{bmatrix} I_8 & \\ & \bar{U}_3 \end{bmatrix},$$

where $I_8 \in \mathbb{R}^{8\times 8}$ is the identity matrix, and $\bar{U}_3 \in \mathbb{R}^{5\times 5}$ has a smaller dimension than $\bar{U}_1$ and $\bar{U}_2$.

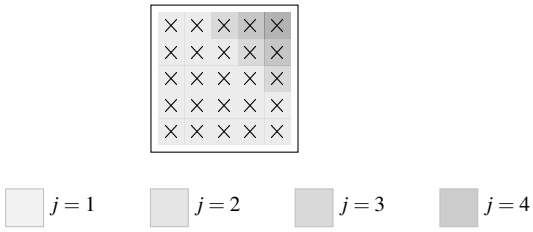


Fig. A.2: Form of $\bar{U}_3$.

ALGORITHM 6: The blocked algorithm for reduction of matrices A, B, and E to the m-Hessenberg–triangular–triangular form.

Input: matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $E \in \mathbb{R}^{n\times n}$; block dimension n_b
Output: upper m-Hessenberg matrix $A = Q^\tau AZ$, upper triangular matrix $B = Q^\tau B$, matrix $C = CZ$, upper triangular matrix $E = Q^\tau EZ$; Q and Z can be accumulated if needed

Perform the QR factorization on E to determine the orthogonal matrix Q_E such that $E = Q_E^\tau E$ is upper triangular;
$A = Q_E^\tau A$;
$B = Q_E^\tau B$;
$imb = 1$;
for $j_c = 0 : n_b : n-1$ **do**
  $j_{c,max} = \min\{j_c + n_b, n-1\}$;
  $n_b = \min\{n_b, n-1-j_c\}$;
  if $n_b = 0$ **then**
    return;
  end
  $l_b = \left\lfloor \frac{n-j_c}{n_b} \right\rfloor - 1$;
  $dim_0 = n - j_c - l_b n_b$;
  for $k = 1 : l_b$ **do**
    $\bar{U}_k = I_{2n_b}$;
  end
  $\bar{U}_{l_b+1} = I_{dim_0}$;
  for $j = j_c + 1 : j_{c,max}$ **do**
    for $i = n : -1 : j+1$ **do**
      // Annihilate $B(i,j)$ or $A(i,j-m)$ if $j > m$ by a Givens rotation applied from the left. Update the corresponding matrix $\bar{U}_k$ from the right, and the matrix E from the left by this rotation. Annihilate $E(i,i-1)$ by a Givens rotation applied from the right, and update rows $j_c+1 : i-1$ of E by this rotation from the right.
      block_1;
    end
    // If j is the last column of the subblock, then update $A(j_c+1:n, j-imb+1:n)$ by all Givens rotations from the right that were generated in the current subblock. If this is not the last column of the column block, then update $B(j_c+1:n, j+1)$ or $A(j_c+1:n, j-m+1)$ for $j \geq m$ by the aggregated rotations from the left.
    block_2;
  end
  // Update $B(j_c+1:n, j_{c,max}+1:m)$ or $A(j_c+1:n, j_{c,max}-m+1:n)$ if $j_{c,max} \geq m$ by the aggregated rotations from the left.
  block_3;
  // Generate the aggregated Givens rotations from the right and apply them to $A(1:j_c, j_c+1:n)$, $E(1:j_c, j_c+1:n)$, and $C(1:p, j_c+1:n)$.
  block_4;
end

Further, due to the special form of the matrices $\bar{U}_k$ (see Figure A.1 and Figure A.2) we can reduce the operation count in the accumulation process.

line 10 in `block_1` can be replaced by the following lines

$lo = ii - j + j_c$;
$up = dim0$;
$[\bar{U}_{l_b+1}(lo{:}up,ii-1), \bar{U}_{l_b+1}(lo{:}up,ii)] = \texttt{givens\_update}(c, s, \bar{U}_{l_b+1}(lo{:}up,ii-1), \bar{U}_{l_b+1}(lo{:}up,ii))$;

line 14 in `block_1` can be replaced by the following lines

$lo = ii - j + j_c$;
$up = n_b + j - j_c$;
$[\bar{U}_k(lo{:}up,ii-1), \bar{U}_k(lo{:}up,ii)] = \texttt{givens\_update}(c, s, \bar{U}_k(lo{:}up,ii-1), \bar{U}_k(lo{:}up,ii))$;

line 8 in `block_4` can be replaced by the following lines

$lo = ii - j + j_c$;
$up = dim0$;
$[\bar{U}_{l_b+1}(lo{:}up,ii-1), \bar{U}_{l_b+1}(lo{:}up,ii)] =$
$\texttt{givens\_update}(CC_{(i+j-j_c-1,j-j_c)}, -SS_{(i+j-j_c-1,j-j_c)}, \bar{U}_{l_b+1}(lo{:}up,ii-1), \bar{U}_{l_b+1}(lo{:}up,ii))$;

line 15 in `block_4` can be replaced by the following lines

$lo = ii - j + j_c$;
$up = n_b + j - j_c$;
$[\bar{U}_k(lo{:}up,ii-1), \bar{U}_k(lo{:}up,ii)] =$
$\texttt{givens\_update}(CC_{(i+j-j_c-1,j-j_c)}, -SS_{(i+j-j_c-1,j-j_c)}, \bar{U}_k(lo{:}up,ii-1), \bar{U}_k(lo{:}up,ii))$;

ALGORITHM 7: `block_1`: Annihilate $B(i,j)$ or $A(i,j-m)$ if $j > m$ by a Givens rotation applied from the left. Update the corresponding matrix $\bar{U}_k$ from the right, and the matrix E from the left by this rotation. Annihilate $E(i,i-1)$ by a Givens rotation applied from the right, and update rows $j_c+1 : i-1$ of E by this rotation from the right.

if $j <= m$ **then**
  $[c, s, B_{(i-1,j)}] = \texttt{givens}(B_{(i-1,j)}, B_{(i,j)})$;
  $B_{(i,j)} = 0$;
else
  $[c, s, A_{(i-1,j-m)}] = \texttt{givens}(A_{(i-1,j-m)}, A_{(i,j-m)})$;
  $A_{(i,j-m)} = 0$;
end
if $i > n - j_c - dim_0 + j$ **then**
  $ii = i - n + dim_0$;
  $[\bar{U}_{l_b+1}(1{:}dim_0,ii-1), \bar{U}_{l_b+1}(1{:}dim_0,ii)] = \texttt{givens\_update}(c, s, \bar{U}_{l_b+1}(1{:}dim_0,ii-1), \bar{U}_{l_b+1}(1{:}dim_0,ii))$;
else
  $k = \left\lfloor \frac{i-j-1}{n_b} \right\rfloor + 1$;
  $ii = i - j_c - (k-1)n_b$;
  $[\bar{U}_k(1{:}2n_b,ii-1), \bar{U}_k(1{:}2n_b,ii)] = \texttt{givens\_update}(c, s, \bar{U}_k(1{:}2n_b,ii-1), \bar{U}_k(1{:}2n_b,ii))$;
end
$[E_{(i-1,i-1:n)}, E_{(i,i-1:n)}] = \texttt{givens\_update}(c, s, E_{(i-1,i-1:n)}, E_{(i,i-1:n)})$;
$[CC_{(i,j-j_c)}, SS_{(i,j-j_c)}, E_{(i,i)}] = \texttt{givens}(E_{(i,i)}, E_{(i,i-1)})$;
$E_{(i,i-1)} = 0$;
$[E_{(j_c+1:i-1,i-1)}, E_{(j_c+1:i-1,i)}] = \texttt{givens\_update}(CC_{(i,j-j_c)}, -SS_{(i,j-j_c)}, E_{(j_c+1:i-1,i-1)}, E_{(j_c+1:i-1,i)})$;

ALGORITHM 8: `block_2`: If j is the last column of the subblock, then update $A(j_c+1:n, j-imb+1:n)$ by all Givens rotations from the right that were generated in the current subblock. If this is not the last column of the column block, then update $B(j_c+1:n, j+1)$ or $A(j_c+1:n, j-m+1)$ for $j \geq m$ by the aggregated rotations from the left.

if $(imb = m)$ *or* $(j = j_{c,max})$ **then**
  $A_{(j_c+1:n,\, j-imb+1:n)} = \texttt{seq\_givens\_update}\big(CC_{(j-imb+2:n,\, j-j_c-imb+1:j-j_c)}, -SS_{(j-imb+2:n,\, j-j_c-imb+1:j-j_c)}, A_{(j_c+1:n,\, j-imb+1:n)}\big)$;
  $imb = 0$;
end
$imb = imb + 1$;
if $j < j_{c,max}$ **then**
  $ii = n - dim_0$;
  if $j < m$ **then**
    $B_{(ii+1:n,\, j+1)} = \bar{U}_{l_b+1}^{\tau} B_{(ii+1:n,\, j+1)}$;
  else
    $j_a = \max\{j-m, 0\}$;
    $A_{(ii+1:n,\, j_a+1)} = \bar{U}_{l_b+1}^{\tau} A_{(ii+1:n,\, j_a+1)}$;
  end
  for $i = l_b : -1 : 1$ **do**
    $ii = j_c + (i-1)n_b$;
    if $j < m$ **then**
      $B_{(ii+1:ii+2n_b,\, j+1)} = \bar{U}_i^{\tau} B_{(ii+1:ii+2n_b,\, j+1)}$;
    else
      $A_{(ii+1:ii+2n_b,\, j_a+1)} = \bar{U}_i^{\tau} A_{(ii+1:ii+2n_b,\, j_a+1)}$;
    end
  end
end

ALGORITHM 9: `block_3`: Update $B(j_c+1:n, j_{c,max}+1:m)$ or $A(j_c+1:n, j_{c,max}-m+1:n)$ if $j \geq m$ by the aggregated rotations from the left.

$j_a = \max\{j_{c,max} - m, 0\}$;
$ii = n - dim_0$;
if $j_{c,max} < m$ **then**
  $B_{(ii+1:n,\, j_{c,max}+1:m)} = \bar{U}_{l_b+1}^{\tau} B_{(ii+1:n,\, j_{c,max}+1:m)}$;
end
$A_{(ii+1:n,\, j_a+1:n)} = \bar{U}_{l_b+1}^{\tau} A_{(ii+1:n,\, j_a+1:n)}$;
for $i = l_b : -1 : 1$ **do**
  $ii = j_c + (i-1)n_b$;
  if $j_{c,max} < m$ **then**
    $B_{(ii+1:ii+2n_b,\, j_{c,max}+1:m)} = \bar{U}_i^{\tau} B_{(ii+1:ii+2n_b,\, j_{c,max}+1:m)}$;
  end
  $A_{(ii+1:ii+2n_b,\, j_a+1:n)} = \bar{U}_i^{\tau} A_{(ii+1:ii+2n_b,\, j_a+1:n)}$;
end

ALGORITHM 10: `block_4`: Generate the aggregated Givens rotations from the right and apply them to $A(1:j_c, j_c+1:n)$, $E(1:j_c, j_c+1:n)$, and $C(1:p, j_c+1:n)$.

for $k = 1:l_b$ **do**
  $U_k = I_{2n_b}$;
end
$U_{l_b+1} = I_{dim_0}$;
for $i = n:-1:n-dim_0+2$ **do**
  for $j = j_c+1:\min\{j_c+n-i+1, j_{c,max}\}$ **do**
    $ii = i+j-j_c-1-n+dim_0$;
    $[\bar{U}_{l_b+1}(1{:}dim_0,ii-1), \bar{U}_{l_b+1}(1{:}dim_0,ii)] =$
    `givens_update`$\big(CC_{(i+j-j_c-1,j-j_c)}, -SS_{(i+j-j_c-1,j-j_c)}, \bar{U}_{l_b+1}(1{:}dim_0,ii-1), \bar{U}_{l_b+1}(1{:}dim_0,ii)\big)$;
  end
end
for $k = l_b:-1:1$ **do**
  for $i = j_c+kn_b+1:-1:j_c+(k-1)n_b+2$ **do**
    for $j = j_c+1:j_{c,max}$ **do**
      $ii = i+j-2j_c-1-(k-1)n_b$;
      $[\bar{U}_k(1{:}2n_b,ii-1), \bar{U}_k(1{:}2n_b,ii)] =$
      `givens_update`$\big(CC_{(i+j-j_c-1,j-j_c)}, -SS_{(i+j-j_c-1,j-j_c)}, \bar{U}_k(1{:}2n_b,ii-1), \bar{U}_k(1{:}2n_b,ii)\big)$;
    end
  end
end
$ii = n-dim_0$;
$A_{(1:j_c,ii+1:n)} = A_{(1:j_c,ii+1:n)}\bar{U}_{l_b+1}$;
$E_{(1:j_c,ii+1:n)} = E_{(1:j_c,ii+1:n)}\bar{U}_{l_b+1}$;
$C_{(1:p,ii+1:n)} = C_{(1:p,ii+1:n)}\bar{U}_{l_b+1}$;
for $i = l_b:-1:1$ **do**
  $ii = j_c+(i-1)n_b$;
  $A_{(1:j_c,ii+1:ii+2n_b)} = A_{(1:j_c,ii+1:ii+2n_b)}\bar{U}_i$;
  $E_{(1:j_c,ii+1:ii+2n_b)} = E_{(1:j_c,ii+1:ii+2n_b)}\bar{U}_i$;
  $C_{(1:p,ii+1:ii+2n_b)} = C_{(1:p,ii+1:ii+2n_b)}\bar{U}_i$;
end