BLOCK ALGORITHMS FOR REORDERING STANDARD AND GENERALIZED SCHUR FORMS LAPACK WORKING NOTE 171

DANIEL KRESSNER*

Abstract. Block algorithms for reordering a selected set of eigenvalues in a standard or generalized Schur form are proposed. Efficiency is achieved by delaying orthogonal transformations and (optionally) making use of level 3 BLAS operations. Numerical experiments demonstrate that existing algorithms, as currently implemented in LAPACK, are outperformed by up to a factor of four.

Key words. Schur form, reordering, invariant subspace, deflating subspace.

AMS subject classifications. 65F15, 65Y20.

1. Introduction.

Applying the QR algorithm to a real square matrix A yields a decomposition of the form

$$A = QTQ^T, \tag{1.1}$$

where Q is orthogonal and T is in real Schur form (also called Murnaghan/Wintner form [19]), i.e.,

$$T = \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1m} \\ 0 & T_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & T_{m-1,m} \\ 0 & \cdots & 0 & T_{mm} \end{bmatrix}, \tag{1.2}$$

where all diagonal blocks of T are of order one or two. Scalar blocks contain the real eigenvalues and two-by-two blocks contain the complex conjugate eigenvalue pairs of A. To compute an orthonormal basis for an invariant subspace $\mathcal{X}$ belonging to k eigenvalues of A, one can reorder the diagonal blocks in T such that the upper left $k \times k$ block of T contains these eigenvalues. Then the first k columns of the updated transformation matrix Q form a basis for $\mathcal{X}$. The LAPACK [1] routine `DTRSEN` provides such a reordering procedure, based on work by Bai and Demmel [2]. In this paper, we will show how to considerably increase the efficiency of `DTRSEN` by delaying the application of the involved orthogonal transformations, which in turn allows the use of level 3 BLAS [10] operations (matrix-matrix multiplications). Similar ideas have been used to speed up other numerical linear algebra algorithms, such as the QR algorithm [4, 6, 18] and the QZ algorithm [9, 14]. In contrast to these works, the orthogonal transformations involved in reordering Schur forms have a much less regular and also less predictable structure. The newly developed block algorithm takes care of this irregularity and attains higher efficiency for a wide range of settings, no matter whether only a few or nearly all of the diagonal blocks of T are to be reordered.

Although we will mainly focus on the computation of invariant subspaces, it should be emphasized that reordering Schur forms plays an important role in many

*`kressner@math.hr`, Department of Mathematics, Bijenička 30, 10000 Zagreb, Croatia. This author has been supported by a DFG Emmy Noether fellowship and in part by the Swedish Foundation for Strategic Research under the Frame Programme Grant A3 02:128.

other applications. For example, this technique is used for performing deflations and restarts in the Jacobi-Davidson algorithm [11, 20] as well as the Krylov-Schur algorithm [22]. It will be briefly discussed how our block algorithms can be adapted to such applications.

The rest of this paper is organized as follows. Section 2 briefly summarizes existing algorithms for reordering Schur forms. In Section 3, the newly developed block algorithm is described. Section 4 contains numerical experiments, investigating the performance of the block algorithm. Finally, in Section 5, we show how the obtained results can be extended to reordering generalized Schur forms.

2. Existing Algorithms. The building block for reordering a given Schur form is the computation of an orthogonal matrix V so that

$$V^TAV = V^T \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} V = \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix}, \quad \begin{array}{l} \lambda(A_{11}) = \lambda(\tilde{A}_{22}), \\ \lambda(A_{22}) = \lambda(\tilde{A}_{11}), \end{array} \tag{2.1}$$

where $A_{11}, \tilde{A}_{22} \in \mathbb{R}^{n_1\times n_1}, A_{22}, \tilde{A}_{11} \in \mathbb{R}^{n_2\times n_2}$ and $n_1, n_2 \in \{1,2\}$; $\lambda(\cdot)$ denotes the set of all eigenvalues of a matrix. This procedure is commonly called *swapping* of A_{11} and A_{22}.

There exist several approaches to perform swapping numerically. Stewart [21] has described an iterative algorithm using QR iterations with the eigenvalues of A_{11} as shifts. Based on earlier work by other authors, Bai and Demmel [2] have developed and analyzed a direct swapping procedures that relies on the solution of a Sylvester equation. In [5], an alternative direct approach is proposed, which instead of solving a Sylvester equation relies on the eigenvectors of A. In the following we focus on the Sylvester equation approach, which is implemented in LAPACK and also forms the basis of our implementation. Nevertheless, it is worth pointing out that the newly developed block algorithms can be used with any swapping procedure.

Assuming that $\lambda(A_{11}) \cap \lambda(A_{22}) = \emptyset$, the approach described in [2] first computes the solution of the Sylvester equation

$$A_{11}X - XA_{22} = \gamma A_{12}, \tag{2.2}$$

where $\gamma \in (0,1]$ is a scaling factor to prevent possible overflow in the solution X. This yields the following block diagonal decomposition:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} = \begin{bmatrix} I_{n_1} & -X \\ 0 & \gamma I_{n_2} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I_{n_1} & X/\gamma \\ 0 & I_{n_2}/\gamma \end{bmatrix}.$$

By a QR decomposition, an orthogonal matrix V is constructed so that

$$V^T \begin{bmatrix} -X \\ \gamma I_{n_2} \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix}, \quad R \in \mathbb{R}^{n_2\times n_2}.$$

Partition $V = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}$ so that $V_{12} \in \mathbb{R}^{n_1\times n_1}$, then V_{12} is invertible and

$$\begin{aligned} V^T \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} V &= \begin{bmatrix} \star & R \\ V_{12}^T & 0 \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} 0 & V_{12}^{-T} \\ R^{-1} & \star \end{bmatrix} \\ &= \begin{bmatrix} RA_{22}R^{-1} & \star \\ 0 & V_{12}^TA_{11}V_{12}^{-T} \end{bmatrix}. \end{aligned}$$

Thus, V produces the desired swapping. In finite-precision arithmetic, the zero $(2,1)$ block becomes polluted by roundoff errors. Explicitly setting this block to zero is not feasible if its entries are significantly larger than the unit roundoff multiplied with the norm of A. The perturbation analysis given in [2] results in an error bound which suggests that such an unfortunate situation may occur if the separation, see [12], between the matrices A_{11} and A_{22} is small. In practice, however, this situation is an extremely rare event, even in the presence of tiny separations. Developing an error analysis which explains this phenomenon is an open problem.

Algorithm 1 Reordering a real Schur form

Input: A matrix $T \in \mathbb{R}^{n\times n}$ in real Schur form (1.2) with m diagonal blocks, an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and a subset of eigenvalues Λ_s, closed under complex conjugation.

Output: A matrix $\tilde{T} \in \mathbb{R}^{n\times n}$ in real Schur form and an orthogonal matrix $\tilde{Q} \in \mathbb{R}^{n\times n}$ so that $\tilde{T} = \tilde{Q}^T T \tilde{Q}$. For some integer j, the set Λ_s is the union of eigenvalues belonging to the j upper-left-most diagonal blocks of $\tilde{T}$. The matrices T and Q are overwritten by $\tilde{T}$ and $Q\tilde{Q}$, respectively.

```
$j \leftarrow 0$
for $i \leftarrow 1, \dots, m$ do
  if $\lambda(T_{ii}) \subset \Lambda_s$ then
    $j \leftarrow j + 1$, select$(j) \leftarrow i$
  end if
end for
top $\leftarrow 0$
for $l \leftarrow 1, \dots, j$ do
  for $i \leftarrow$ select$(l)$, select$(l) - 1, \dots,$ top $+ 1$ do
    Swap $T_{i-1,i-1}$ and $T_{ii}$ by an orthogonal similarity transformation and apply this
    transformation to the rest of the columns and rows of $T$, and the columns of $Q$.
  end for
  top $\leftarrow$ top $+ 1$
end for
```

Having a swapping procedure on hand, we can reorder a selected set of eigenvalues in a bubble sort fashion to the top left corner of a given real Schur form, see Algorithm 1. This algorithm is implemented in the LAPACK routine `DTRSEN`, which also provides (estimates of) condition numbers for the eigenvalue cluster Λ_s and the corresponding invariant subspace. If Λ_s contains k eigenvalues then Algorithm 1 requires $\mathcal{O}(kn^2)$ flops. The exact computational cost depends on the distribution of selected eigenvalues over the block diagonal of T.

3. A Block Algorithm.

For large matrices, Algorithm 1 performs poorly on modern computers with a deep memory hierarchy, ranging from large and slow to small and fast memory. This is due to the fact that each outer loop of Algorithm 1 performs only $\mathcal{O}((\text{select}(l) - \text{top})n)$ flops while moving $\mathcal{O}((\text{select}(l) - \text{top})n)$ memory, resulting in an $\mathcal{O}(1)$ communication/computation ratio. This ratio can be considerably reduced by delaying the update of those parts of T that are far off the diagonal.

The basic idea of the resulting block reordering algorithm is best explained by an example. Let us consider a 16×16 upper triangular matrix T having the eigenvalues at diagonal positions $2, 6, 12, 13, 15$ and 16 selected, marked by the black discs in Figure 3.1 (a). We activate the eigenvalues in the $ev = 4$ upper-left-most positions $(2, 6, 12, 13)$, those correspond to the gray disks in Figure 3.1 (b). The active eigen-

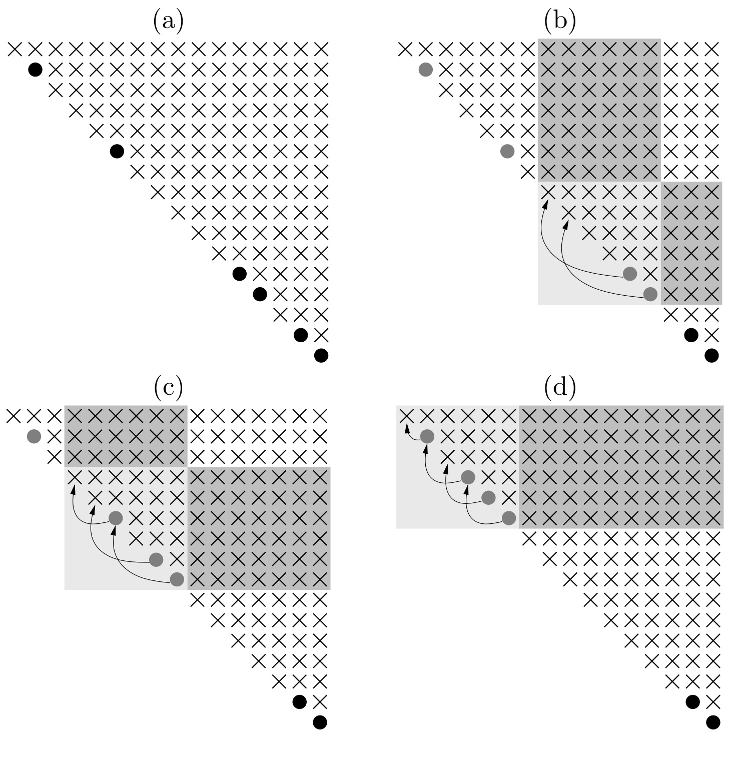


FIG. 3.1. *Illustration of Algorithm 2.*

values will be reordered to the top in a window-by-window fashion. The first window of order $w = 6$ contains the bottom active eigenvalue in its bottom right corner. This window is illustrated by the light gray area in Figure 3.1 (b). The eigenvalues at positions 12 and 13 are reordered to the top of the window, i.e., positions 8 and 9. The corresponding orthogonal transformations are saved and applied afterwards to the rest of the matrix, this will be discussed in more detail below. The next 6×6 window contains active eigenvalues at positions $6, 8$ and 9, see Figure 3.1 (c). Again, these eigenvalues are reordered to the top of the window. The last window contains all active eigenvalues, which reach their final positions after having been reordered within this window. This process is repeated with the next bunch of at most ev disordered eigenvalues, which in our example are the eigenvalues sitting at positions 15 and 16.

Algorithm 2 contains an extension of the idea to the general setting. To simplify the presentation, it is assumed that all diagonal blocks of T are scalars, i.e., T has only real eigenvalues. In order to cover complex conjugate pairs of eigenvalues, the active window $T(i_{\text{low}} : i_{\text{hi}}, i_{\text{low}} : i_{\text{hi}})$ must be slightly adjusted to avoid the window borders crossing two-by-two blocks of T. Here, the colon notation $T(i_1 : i_2, j_1 : j_2)$ is used to designate the submatrix of T defined by rows i_1 through i_2 and columns j_1 through j_2.

3.1. Implementation details. Some remarks concerning the actual implementation of Algorithm 2 are in order.

Algorithm 2 Reordering a Schur form (block algorithm)

Input and Output: See Algorithm 1. Additional input: block parameters ev (max #eigenvalues in each window) and w (window size).

```
    $i_{\rm ord} \leftarrow 0$    % $i_{\rm ord}$ = #number of eigenvalues already in order
    while $i_{\rm ord} < \#\Lambda_s$ do
      % Find first $n_{ev} \leq ev$ disordered eigenvalues from top.
      $n_{\rm ev} \leftarrow 0$,  $i_{\rm hi} \leftarrow i_{\rm ord} + 1$
      while $n_{\rm ev} \leq ev$ and $i_{\rm hi} \leq n$ do
        if $T_{ii} \in \Lambda_s$ then $n_{\rm ev} \leftarrow n_{\rm ev} + 1$ end if
        $i_{\rm hi} \leftarrow i_{\rm hi} + 1$
      end while
      % Reorder these eigenvalues window-by-window to top.
      while $i_{\rm hi} > i_{\rm ord} + n_{\rm ev}$ do
        $i_{\rm low} \leftarrow \max\{i_{\rm ord} + 1, i_{\rm hi} - w + 1\}$,  $n_w \leftarrow i_{\rm hi} - i_{\rm low} + 1$
(1)     Apply Algorithm 1 to the active window $T(i_{\rm low} : i_{\rm hi}, i_{\rm low} : i_{\rm hi})$ in order to reorder the
        $k \leq n_{\rm ev}$ selected eigenvalues that reside in this window to top of window. Let the
        corresponding orthogonal transformation matrix be denoted by $U$.
(2a)    Update $T(i_{\rm low} : i_{\rm hi}, i_{\rm hi} + 1 : n) \leftarrow U^T T(i_{\rm low} : i_{\rm hi}, i_{\rm hi} + 1 : n)$.
(2b)    Update $T(1 : i_{\rm low} - 1, i_{\rm low} : i_{\rm hi}) \leftarrow T(1 : i_{\rm low} - 1, i_{\rm low} : i_{\rm hi})U$.
(2c)    Update $Q(1 : n, i_{\rm low} : i_{\rm hi}) \leftarrow Q(1 : n, i_{\rm low} : i_{\rm hi})U$.
        $i_{\rm hi} \leftarrow i_{\rm low} + k - 1$
      end while
      $i_{\rm ord} \leftarrow i_{\rm ord} + n_{\rm ev}$
    end while
```

Step (1). To maintain data locality, the orthogonal transformations used for swapping diagonal blocks in the active window $T(i_{\rm low} : i_{\rm hi}, i_{\rm low} : i_{\rm hi})$ are only applied within this window. The information encoding these transformations (Givens rotations or Householder reflectors of order 3) is stored successively in a one-dimensional array `DTRAF`. An upper bound on the length of this array is given by $5(n_w - k)k$, provided that $k \leq n_w/2$.

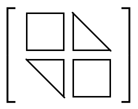
Step (2a)–(2c). In these three steps, the transformations pipelined in `DTRAF` are used to update the rest of T as well as the orthogonal matrix Q. This can be done in at least two different ways. First, the transformations can be used in their original factored form, which amounts to applying successively the stored Givens rotations and Householder reflectors. Rows are updated in stripes of n_b columns in order to maintain locality of the memory reference pattern. (In our experiments, n_b was set to 32.) Second, the transformations can be accumulated into an $n_w \times n_w$ matrix U, which is then be applied using calls to the level 3 BLAS routine `DGEMM` (matrix-matrix multiplication). Both alternatives are provided in our implementation. The decision about which one to use is based on the ratio

$$r_{\rm flops} = \frac{\#\text{flops for applying transformations in factored form}}{\#\text{flops for applying transformations in accumulated form}}. \tag{3.1}$$

Note that the value of $r_{\rm flops}$ depends not only on the number of selected eigenvalues in the active window but also on their distribution over the block diagonal of T. Matrix-matrix multiplications are used whenever $r_{\rm flops}$ exceeds a certain threshold $r_{\rm mmult} \in [0, 1]$. The optimal value for $r_{\rm mmult}$ depends very much on the performance of `DGEMM`.

If all eigenvalues in the active window are real then only 1×1 blocks are swapped. In this case, the matrix U has the following block structure:

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} = \tag{3.2}$$



i.e., the submatrices $U_{12} \in \mathbb{R}^{(n_w-k)\times(n_w-k)}$ and $U_{21} \in \mathbb{R}^{k\times k}$ are lower and upper triangular, respectively. This structure is exploited in our implementation, replacing the single call to `DGEMM` by two calls to `DGEMM` and `DTRMM` (triangular matrix-matrix multiplication). If there are pairs of complex conjugate eigenvalues then the off-diagonal blocks of U have only banded structure, which is more difficult to exploit using level 3 BLAS operations.

Complex matrices. The purpose of the two-by-two diagonal blocks in the Schur form (1.2) is to preserve the realness of A. If A is a complex matrix, the corresponding (complex) Schur form is an upper triangular matrix T, i.e., all diagonal blocks are one-by-one. This considerably simplifies the implementation of Algorithms 1 and 2, which both can be extended in a direct manner to complex matrices. Moreover, the unitary transformation matrix U always has the structure displayed in (3.2).

Extension to other orderings. Some applications, such as the Jacobi-Davidson algorithm, require all eigenvalues of T to be sorted in a certain, specified order. To simplify the discussion, let us assume that one wants to sort the eigenvalues in descending magnitude. Algorithm 2 can be effectively applied to achieve this goal. First, the largest ev eigenvalues are selected and ordered to the upper left part of T, using Algorithm 2. Then these ev eigenvalues are sorted in descending magnitude, by at most ev applications of Algorithm 1 to $T(1:ev, 1:ev)$. The off-diagonal part $T(1:ev, ev{+}1:n)$ and the transformation matrix Q are updated as in Steps (2a)–(2c) of Algorithm 2. This procedure is repeated with the next largest ev eigenvalues, which are sorted to $T(ev+1:2\times ev, ev+1:2\times ev)$. Proceeding in this manner, $\lceil n/ev \rceil$ runs of Algorithm 2 yield all eigenvalues in descending magnitude on the diagonal of the updated matrix T .

4. Numerical Experiments.

Algorithm 2 has been implemented in a Fortran 77 routine called `BDTRSEN`, partly based on slightly modified versions of the LAPACK routines `DTREXC` and `DLAEXC`. The numerical experiments have been executed on a dual Athlon MP2000+ (1.66 Ghz) with 1 GB memory. The relevant functionality of the LAPACK library as well as our implementation were compiled with the Portland Group Inc. Fortran 90/95 compiler 6.0 using the options `-Kieee -O3 -Mscalarsse -Mnoframe -Munroll -Mnontemporal -tp athlonxp -fast` and the BLAS implementation provided by ATLAS [24].

Choice of block parameters. First, several numerical experiments were performed with 1500×1500 random matrices, reduced to real Schur form, having 750 selected eigenvalues randomly distributed over the block diagonal. The purpose of these experiments was to find good choices for the parameters ev (max #eigenvalues in each window), w (window size), r_{mmult} (matrix-matrix multiplication threshold), and to gain insight into the sensitivity of the performance of `BDTRSEN` with respect to changes of these parameters. Within a search space covering all possible combinations of $ev \in \{20, 24, \ldots, 120\}$, $w \in \{2\times ev, 5/2\times ev, \ldots, 4\times ev\}$, $r_{\text{mmult}} \in \{0\%, 5\%, \ldots, 100\%\}$, we found $ev = 60$, $w = 2 \times ev$, $r_{\text{mmult}} = 30\%$ to be optimal.

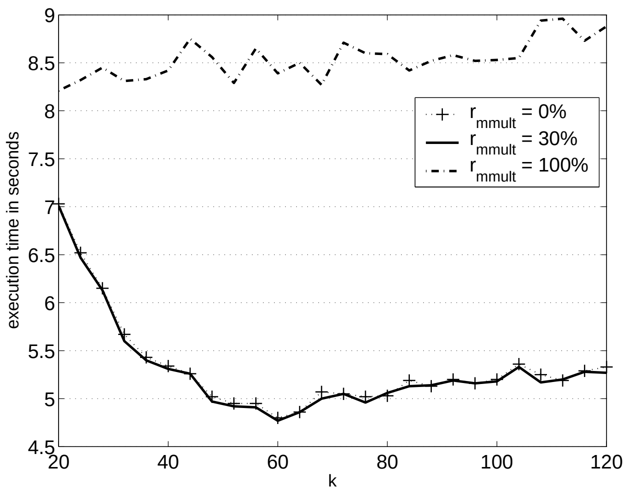


FIG. 4.1. *Execution time of* BDTRSEN *for* $ev \in \{20, 24, \ldots, 120\}$*,* $w = 2$*, and* $r_{\text{mmult}} \in \{0\%, 30\%, 100\%\}$

.

Moreover, it was observed that the execution time is not very sensitive to modest parameter changes. This is demonstrated in Figure 4.1, which shows that the optimal value of ev resides in a rather flat trough of the performance curve for $r_{\text{mmult}} = 30\%$. The corresponding curve for $r_{\text{mmult}} = 0\%$ (off-diagonal update is always performed by matrix-matrix multiplications) is nearly indistinguishably close. This does not comes as a surprise; with so many eigenvalues to be reordered the ratio r_{flops} defined in (3.1) rarely falls below 30%. For $r_{\text{mmult}} = 100\%$ (off-diagonal update is never performed by matrix-matrix multiplications), the performance is significantly worse. Still, the obtained optimal execution time of 8.2 seconds is favorable compared with the 20.7 seconds needed by the LAPACK routine DTRSEN.

Comparison with existing LAPACK implementation. To demonstrate the performance of BDTRSEN for a wider range of settings, we varied the matrix dimension $n \in \{500, 1000, 1500\}$ as well as the portion of selected eigenvalues $d \in \{5\%, 25\%, 50\%\}$. Two different distributions of eigenvalues were considered. In the "random" distribution, the eigenvalue(s) in each diagonal block of T are selected with probability d. In the "bottom" distribution, all selected eigenvalues are located in the $dn \times dn$ bottom right submatrix. All block parameters were chosen as determined above.

Table 4.1, which contains the execution times, shows that BDTRSEN performs significantly better than the LAPACK routine DTRSEN; if n is sufficiently large it requires less than 25% of the time needed by DTRSEN. To test the numerical stability of BDTRSEN, we measured the orthogonality of Q, $\|Q^TQ - I\|_F$, as well as the residual $\|Q^TTQ - \tilde{T}\|_F / \|T\|_F$ and found all values satisfactorily close to the machine precision.

5. Extension to Generalized Schur Forms.

A matrix pair $(S, T) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times n}$ is said to be in generalized Schur form if S is in real Schur form while T is upper triangular. Note that the two-by-two blocks in S now correspond to complex conjugate pairs of (generalized) eigenvalues of (S, T), defined as the roots of $\det(S - \lambda T)$. Reordering generalized Schur forms by an orthogonal similarity transformation

			Update of T alone		Update of T, Q	
n	sel.	distr.	`DTRSEN`	`BDTRSEN`	`DTRSEN`	`BDTRSEN`
500	5%	random	0.10	0.06	0.16	0.09
500	5%	bottom	0.16	0.07	0.27	0.12
500	25%	random	0.30	0.14	0.53	0.23
500	25%	bottom	0.59	0.25	1.08	0.35
500	50%	random	0.34	0.18	0.63	0.26
500	50%	bottom	0.76	0.34	1.35	0.48
1000	5%	random	0.95	0.24	1.44	0.44
1000	5%	bottom	1.41	0.37	2.25	0.37
1000	25%	random	2.90	0.72	4.46	1.20
1000	25%	bottom	5.53	1.39	8.78	2.24
1000	50%	random	3.60	1.00	5.73	1.61
1000	50%	bottom	7.23	1.89	11.45	3.00
1500	5%	random	2.94	0.62	4.29	1.14
1500	5%	bottom	5.31	1.22	8.16	2.11
1500	25%	random	10.78	2.10	16.04	3.64
1500	25%	bottom	20.36	4.03	31.15	6.72
1500	50%	random	13.53	2.90	20.73	4.78
1500	50%	bottom	26.86	5.46	40.91	9.01

TABLE 4.1
Execution time in seconds for unblocked (`DTRSEN`*) and blocked (*`BDTRSEN`*) reordering of an $n \times n$ matrix in Schur form. The figures in columns 4 and 5 exclude the time needed for updating the orthogonal transformation matrix Q.*

$(Q^T SZ, Q^T TZ)$ serves a similar purpose as for standard Schur forms; it admits the computation of so called deflating subspaces [12], which generalize the concept of invariant subspaces to matrix pairs.

Van Dooren [23], Kågström [13], as well as Kågström and Poromaa [15, 16] have developed reordering algorithms for matrix pairs. In the following, we focus on the variant described in [16], which is in the spirit of [2] and implemented in LAPACK. The building block of this algorithm is the computation of orthogonal matrices V and W such that

$$V^T \left(\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} \right) W = \left(\begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ 0 & \tilde{A}_{22} \end{bmatrix}, \begin{bmatrix} \tilde{B}_{11} & \tilde{B}_{12} \\ 0 & \tilde{B}_{22} \end{bmatrix} \right) \tag{5.1}$$

and

$$\lambda(A_{11}, B_{11}) = \lambda(\tilde{A}_{22}, \tilde{B}_{22}), \quad \lambda(A_{22}, B_{22}) = \lambda(\tilde{A}_{11}, \tilde{B}_{11}),$$

where $A_{11}, B_{11}, \tilde{A}_{22}, \tilde{B}_{22} \in \mathbb{R}^{n_1 \times n_1}$ and $A_{22}, B_{22}, \tilde{A}_{11}, \tilde{B}_{11} \in \mathbb{R}^{n_2 \times n_2}$ with $n_1, n_2 \in \{1, 2\}$. Here, $\lambda(\cdot, \cdot)$ denotes the set of generalized eigenvalues of a matrix pair.

To achieve (5.1), first the solution of the so called generalized Sylvester equation

$$\begin{array}{rcl} A_{11}X - YA_{22} & = & \gamma A_{12}, \\ B_{11}X - YB_{22} & = & \gamma B_{12}, \end{array} \tag{5.2}$$

is computed, where $\gamma \in (0, 1]$ is a scaling factor to prevent overflow in X and Y. Under the assumption $\lambda(A_{11}, B_{11}) \cap \lambda(A_{22}, B_{22}) = \emptyset$, this system of matrix equations

			Update of (S,T)		$(S,T),Q,Z$	
n	sel.	distr.	`DTGSEN`	`BDTGSEN`	`DTGSEN`	`BDTGSEN`
500	5%	random	0.40	0.29	0.55	0.42
500	5%	bottom	0.69	0.51	0.97	0.75
500	25%	random	1.22	0.79	1.72	0.97
500	25%	bottom	2.51	1.50	3.58	1.74
500	50%	random	1.74	1.18	2.49	1.45
500	50%	bottom	3.31	2.01	4.70	2.31
1000	5%	random	3.07	1.50	5.10	2.09
1000	5%	bottom	4.32	1.78	7.02	2.22
1000	25%	random	8.65	4.13	14.09	5.54
1000	25%	bottom	16.09	6.77	26.10	8.41
1000	50%	random	10.09	4.93	16.27	6.58
1000	50%	bottom	21.22	8.90	34.37	11.08
1500	5%	random	11.51	3.81	18.49	5.59
1500	5%	bottom	16.00	5.30	27.46	7.52
1500	25%	random	33.87	10.00	57.08	13.86
1500	25%	bottom	60.66	16.70	106.15	22.08
1500	50%	random	39.52	12.85	70.42	17.95
1500	50%	bottom	79.77	22.29	142.07	29.40

TABLE 5.1
Execution times in seconds for unblocked (`DTGSEN`*) and blocked (*`BDTGSEN`*) reordering of an $n \times n$ matrix pair in generalized Schur form. The figures in columns 4 and 5 exclude the time needed for updating the orthogonal transformation matrices Q and Z.*

yields a unique solution (X, Y). Then, using QR and RQ decompositions, orthogonal matrices V and W are constructed such that

$$V^T \begin{bmatrix} -Y \\ \gamma I_{n_2} \end{bmatrix} = \begin{bmatrix} R_Y \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \gamma I_{n_2} & X \end{bmatrix} W = \begin{bmatrix} 0 & R_X \end{bmatrix},$$

with upper triangular matrices $R_X, R_Y \in \mathbb{R}^{n_2 \times n_2}$. Similar to the standard case, it is simple to show that the matrices V and W produce the desired swapping (5.1).

Along the lines of Algorithm 1, the LAPACK routine `DTGSEN` applies this swapping procedure in a bubble sort fashion to reorder a generalized Schur form. A block variant of `DTGSEN` can be derived along the lines of Algorithm 2. We have implemented this block algorithm in a Fortran 77 routine called `BDTGSEN` and repeated the numerical experiments from Section 4, see Table 5.1.

Again, the performance improvements are significant albeit a bit less than for reordering standard Schur forms. For example, the execution time for reordering 750 eigenvalues in a generalized Schur form from bottom to top is divided by a factor of 3.6, while the corresponding factor for performing the same task in a standard Schur form is given by 4.9. A possible explanation for this effect is that the matrices V and W from (5.1) are used in accumulated form in `DTGSEN`, which turns out to be slightly more efficient than the factored form used in `DTRSEN`.

6. Conclusions.

In this paper, we have described efficient algorithms for reordering eigenvalues in a standard or generalized Schur form. It is demonstrated that Fortran 77 implementations of these new algorithms outperform the corresponding

LAPACK implementations significantly. Parallel versions are currently under investigation. Combined with other improvements of the QR algorithm [6, 7, 18], these developments might eventually lead to a competitive alternative to sign function based methods for the parallel computation of invariant subspaces, see, e.g., [3, 8]. Though, the pleasant simplicity of sign function based methods is unlikely to be achieved that way.

Final Note and Acknowledgments. The work presented in this article is based on preliminary results derived in [17]. The author is grateful to Robert Granat for several discussions on the work presented in this paper. The computational experiments in Sections 4 and 5 were performed using facilities of the High Performance Computing Center North (HPC2N) in Umeå, Sweden. The developed Fortran codes are available on request from the author.

REFERENCES

[1] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, third edition, 1999.

[2] Z. Bai and J. W. Demmel. On swapping diagonal blocks in real Schur form. *Linear Algebra Appl.*, 186:73–95, 1993.

[3] Z. Bai and J. W. Demmel. Using the matrix sign function to compute invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 19(1):205–225, 1998.

[4] C. H. Bischof, B. Lang, and X. Sun. A framework for symmetric band reduction. *ACM Trans. Math. Software*, 26(4):581–601, 2000.

[5] A. Bojanczyk and P. Van Dooren. Reordering diagonal blocks in the real Schur form. In M. S. Moonen, G. H. Golub, and B. L. R. De Moor, editors, *Linear algebra for large scale and real-time applications (Leuven, 1992)*, volume 232 of *NATO Adv. Sci. Inst. Ser. E Appl. Sci.*, pages 351–352, Dordrecht, 1993. Kluwer Acad. Publ.

[6] K. Braman, R. Byers, and R. Mathias. The multishift *QR* algorithm. I. Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[7] K. Braman, R. Byers, and R. Mathias. The multishift *QR* algorithm. II. Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[8] R. Byers, C. He, and V. Mehrmann. The matrix sign function method and the computation of invariant subspaces. *SIAM J. Matrix Anal. Appl.*, 18(3):615–632, 1997.

[9] K. Dackland and B. Kågström. Blocked algorithms and software for reduction of a regular matrix pair to generalized Schur form. *ACM Trans. Math. Software*, 25(4):425–454, 1999.

[10] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1990.

[11] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1):94–125, 1998.

[12] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[13] B. Kågström. A direct method for reordering eigenvalues in the generalized real Schur form of a regular matrix pair (A, B). In M. S. Moonen, G. H. Golub, and B. L. R. De Moor, editors, *Linear algebra for large scale and real-time applications (Leuven, 1992)*, volume 232 of *NATO Adv. Sci. Inst. Ser. E Appl. Sci.*, pages 195–218. Kluwer Acad. Publ., Dordrecht, 1993.

[14] B. Kågström and D. Kressner. Multishift variants of the QZ algorithm with aggressive early deflation. Report UMINF-05.11, Department of Computing Science, Umeå University, Umeå, Sweden, 2005.

[15] B. Kågström and P. Poromaa. Computing eigenspaces with specified eigenvalues of a regular matrix pair (A, B) and condition estimation: theory, algorithms and software. *Numer. Algorithms*, 12(3-4):369–407, 1996.

[16] B. Kågström and P. Poromaa. LAPACK-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs. *ACM Trans. Math. Software*, 22(1):78–103, 1996.

[17] D. Kressner. *Numerical Methods and Software for General and Structured Eigenvalue Problems.* PhD thesis, TU Berlin, Institut für Mathematik, Berlin, Germany, 2004.

[18] B. Lang. Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift, 1997.

[19] F. D. Murnaghan and A. Wintner. A canonical form for real matrices under orthogonal transformations. *Proc. Natl. Acad. Sci. USA*, 17:417–420, 1931.

[20] G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

[21] G. W. Stewart. Algorithm 407: HQR3 and EXCHNG: FORTRAN programs for calculating the eigenvalues of a real upper Hessenberg matrix in a prescribed order. *ACM Trans. Math. Software*, 2:275–280, 1976.

[22] G. W. Stewart. A Krylov-Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614, 2002.

[23] P. Van Dooren. Algorithm 590: DSUBSP and EXCHQZ: Fortran subroutines for computing deflating subspaces with specified spectrum. *ACM Trans. Math. Softw.*, 8:376–382, 1982.

[24] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.