**Universiteit Utrecht**

*Mathematics Institute*

**Matlab Code for Sorted Real Schur Forms**

by

**Jan Brandts**

Preprint

No. 1180

January, 2001

Also available at URL `http://www.math.uu.nl/people/brandts`

# Matlab Code for Sorted Real Schur Forms

Jan Brandts*

January, 2001

**Abstract**

In Matlab, there exists a standard command to generate a real Schur form, and another command transforms a real Schur form into a complex one. In Golub and Van Loan (1996), a Matlab-like routine is sketched that sorts a complex Schur form: given a target value $\tau$ in the complex plane, the diagonal elements of the upper triangular factor $T$ are ordered according to their distance to $\tau$. In this note, we implement a procedure to construct sorted real Schur forms in Matlab. This implementation is based on a block-swapping procedure by Bai and Demmel (1993). Moreover, we describe how to compute a partial Schur form (see Saad (1992)) in case the matrix $A$ is too large to compute a complete Schur form and to order it *a posteriori*. The sorting of real Schur forms, both partially and completely, have important applications in the computation and tracking of invariant subspaces.

## 1 Schur forms

Here we briefly recall several appearances of the Schur form, the partial Schur form, and their relation to non-normality as well as to invariant subspaces. Then we formulate our objectives and motivate the relevance of the chosen topic.

### 1.1 The classical Schur form and non-normality

A basic fact in numerical algebra [8, 13] is, that any square matrix $A$ can be unitarily transformed to upper triangular form,

$$AU = UT, \quad U^*U = I \quad \text{and} \quad T \text{ is upper triangular.} \tag{1}$$

The diagonal $D$ of $T$ contains the eigenvalues of $A$, and the strict upper triangular part $N := T - D$ of $T$ gives information about the non-normality of $A$.

$$A \text{ is normal} \quad \Leftrightarrow \quad AA^* = A^*A \quad \Leftrightarrow \quad N = 0. \tag{2}$$

The Schur form is not unique. In fact, for each diagonal matrix $D$ containing the eigenvalues of $A$ there exist corresponding $U = U(D)$ and $N = N(D)$ generating a Schur form $Q^*AQ = T = D + N$. Although $N$ depends on $D$, the Frobenius norm

*Mathematical Institute, Utrecht University, P.O.Box 80.010, 3508 TA, Utrecht, The Netherlands. E-mail: brandts@math.uu.nl

of $N$ does not depend on $D$ (nor on $U$) but is a quantity uniquely determined by $A$. Denoted by $\nu(A)$, it is usually referred to as $A$'s *departure from normality*. In this context, the formula

$$\nu(A)^2 = \|N\|^2 = \|A\|^2 - \sum_{j=1}^{n} |\lambda_j|^2 \tag{3}$$

is well-known. It can be interpreted as Pythagoras' Theorem in the Hilbert space of $n \times n$ matrices with Frobenius inner product $< G, H >= \mathrm{trace}(G^*H)$ applied to the orthogonal decomposition $U^*AU = D + N$.

### 1.2 The real Schur form

If $A$ is a real matrix, its non-real eigenvalues come in complex conjugate pairs, and this fact can be used to produce a so-called *real Schur form* in which both $U$ and $T$ are real matrices. This goes at the cost of allowing two-by-two blocks on the diagonal of $T$, so, strictly speaking, $T$ is no longer an upper triangular matrix, but *quasi-triangular*. This is illustrated in Figure 1.

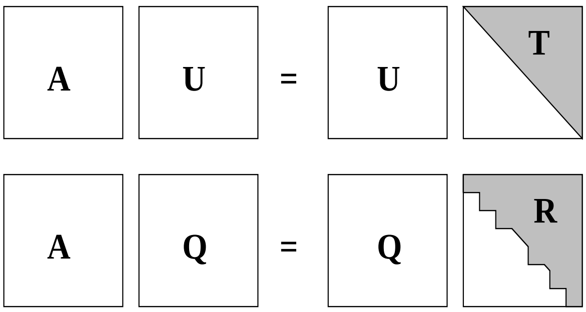


**Figure 1.** Complex Schur form $AU = UT$ versus Real Schur form $AQ = QR$. In the latter, there may be diagonal blocks corresponding to real representation of complex conjugate eigenvalues. Any ordering of the diagonal blocks can be obtained.

The eigenvalues of the two-by-two blocks are exactly the complex conjugate eigenpairs of $A$. Writing $\lambda = \gamma + \mu i$ for one of the conjugates and $v = y + iz$ for a corresponding eigenvector, the real invariant subspace belonging to the conjugate pair is spanned by $y$ and $z$ and

$$A(y|z) = (y|z) \begin{bmatrix} \gamma & \mu \\ -\mu & \gamma \end{bmatrix}. \tag{4}$$

In the context of real Schur forms, we will use the notation $AQ = QR$ as opposed to $AU = UT$. Here, $Q$ is real orthogonal and $R$ the corresponding quasi-triangular real Schur form. Like the complex Schur form, the real Schur form is not unique, in the sense that the diagonal elements and blocks can appear in any prescribed order.

### 1.3 The purpose of ordering

The importance of ordering a Schur form comes from the fact that the first $k$ columns of $U$ and $Q$ form an orthonormal basis for the invariant subspace belonging to the set of eigenvalues in the $k \times k$ top-left part of $T$ and $R$. Of course, in the real case, $k$ should be such, that no two-by-two block in $R$ is cut into two. Depending on the interest of the user, it is therefore welcomed to have a tool that brings the eigenvalues of interest to the top-left part of the (quasi-)triangular matrix. This can be realized by re-ordering algorithms, which results in ordered Schur forms.

For an ordering algorithm for a Schur form with only one-by-one diagonal blocks (which includes the complex case), we refer to Section 7.6.2 of Golub and Van Loan [8]. For a Matlab code based on this algorithm to sort a complex Schur form, see [6]. A direct (as opposed to iterative) ordering algorithm for real Schur forms has been described and thoroughly analyzed by Bai and Demmel in [2]. Their procedure will be briefly reviewed in Section 2 of this paper, and implemented in a slightly different form in Matlab in Section 2.3. Numerical experiments are provided that illustrate the success of the code when swapping two two-by-two blocks having small separation. As a more practical application, a real Schur form of the GRCAR(n) matrix from the Matlab Matrix Gallery will be ordered.

### 1.4 Partial Schur forms

Suppose that we are only interested in a very low dimensional invariant subspace of a very large real matrix $A$. In that case, applying the $QR$-algorithm to compute a complete Schur form, to order it, and to discard the data that we are not interested in, is not a feasible approach. An alternative is, to build a so-called *partial Schur form* [12]. This can be done by an inductive procedure in which gradually a tall skinny matrix $U$ resp. $Q$, and a corresponding small (quasi)triangular real matrix $T$ resp. $R$ are computed such that $AU = UT$ in the complex case, resp. $AQ = QR$ in the real case, as illustrated in Figure 2.

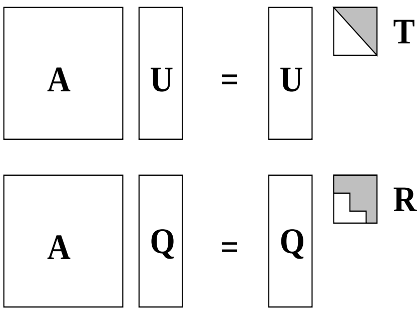


**Figure 2.** Partial complex and real Schur forms $AU = UT$ and $AQ = QR$. They are constructed iteratively as information about the spectrum of $A$ becomes available.

This approach actually represents a constructive version of the usual existence proof of Schur forms by induction. Starting off either with a single eigenvector belonging to the eigenvalue closest to a given target, or with a complex conjugate pair of which one of the two is closest (over all eigenvalues) to the target, more eigenvalues and Schur vectors are computed and added. In the complex case, this procedure is exactly the one followed in [7]. In Section 3, we outline the real case of this procedure.

### 1.5 Motivation

Our interest in a *complete* sorted real Schur form of a matrix results from the problem of *selection* in Ritz-Galerkin methods to compute invariant subspaces of very large though sparse matrices. The Jacobi-Davidson QR and QZ algorithms [7] and related methods [3, 10] aim to compute an invariant subspace of a non-hermitian real matrix $A$ belonging to a cluster of $k << n$ eigenvalues closest to a prescribed target $\tau$ in the complex plane. By Ritz-Galerkin projection of $A$ on a test-and-trial space $\mathcal{V}$, an approximation for this invariant subspace of $A$ is extracted from the projected matrix $M := V^*AV$ where $V$ is an orthonormal matrix spanning $\mathcal{V}$.

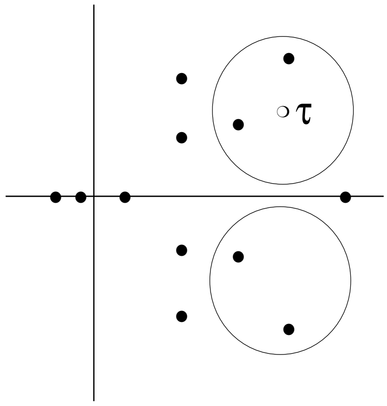


**Figure 3.** Ritz values (black dots) in the complex plane and a target value (open dot) $\tau$. A selection approach based on the complex Schur form would select the two complex eigenvectors corresponding to the two Ritz values in the circle, while an approach based on the real Schur form selects the real four-dimensional subspace corresponding to the two complex-conjugate pairs.

Since $M$, though small, may be defective or ill-conditioned, it seems best to select the first $k$ Schur vectors belonging to the Schur decomposition of $M$ for which the $k$ eigenvalues of $M$ (called Ritz-values) closest to $\tau$ are in the $k \times k$ upper left portion of the triangular factor - 'best' in the sense that they represent an orthonormal basis for the Ritz invariant subspace. Here, we propose to select, together with each complex Ritz value, also its complex conjugate, even when it is far from the target (which may happen when $\tau$ is not real). This can be realized by employing sorted real Schur forms. This might increase the real dimension of the approximation of the invariant subspace to select from $\mathcal{V}$ up to $2k$, but it avoids complex arithmetic, which in turn may be (more than) twice as costly as real arithmetic. Also, it seems natural not to split up complex conjugate pairs, or alternatively, unnatural to be interested in a non-real $\tau$ without being interested in $\overline{\tau}$ as well.
Our interest in a *partial* sorted real Schur form lies in the same area. After an approximation of a real eigenvector, or a real two-dimensional eigenspace belonging to a complex eigenvalue has been computed, this can be interpreted as a minimum size partial real Schur form. Depending on the wishes of the user of a particular algorithm, it should then be possible to continue the computation of a next real Schur vector or two-dimensional real Schur subspace. In Section 3 we will show how to do this.

### 1.6 Objectives and outline

This paper is mainly concerned with presenting Matlab programs for the practical computation of real Schur forms with prescribed conditions for the ordering of the diagonal blocks. It should be noted that the use of Matlab in scientific computing, both at research level as well as in teaching applications, has in recent years become more and more the standard. Therefore we believe that the functions that we will present, will be a valuable tool for those who want to avoid (and can permit to avoid, in the sense that in general, Matlab routines will be relatively slow) the use of LAPACK and related packages. We will distinguish between the two essentially different approaches that were described in more detail above.

• A complete sorted real Schur form is needed. Given an arbitrary real Schur form, we perform the re-ordering of the diagonal blocks as given by Bai and Demmel in [2] in a slightly different form. This will be the topic of Section 2.

• A partial sorted real Schur form of a very large matrix is needed. For this we suppose we have a black-box large eigenvalueproblem solver at our disposal. Section 3 contains the details.

As mentioned before, Matlab includes a routine that computes a real Schur form when the given matrix $A$ is real, and also a routine that transforms a given real Schur form into a complex Schur form. It does not include any ordering algorithm for Schur forms. For an algorithm that orders a complex Schur form we refer to [8], and for a Matlab version of this algorithm to [6].

# 2 Computing sorted real Schur forms

Let $A$ be a real matrix, and $AQ = QR$ a real Schur form. We will show how to swap the positions of two adjacent diagonal blocks, assuming that those blocks do not have eigenvalues in common. Once it is known how to swap two adjacent blocks, a bubblesort-like method can easily be described to obtain almost any ordering of the diagonal blocks: the only restriction is that blocks that have eigenvalues in common, stay in their original order.

## 2.1 The Bai-Demmel algorithm

The algorithm by Bai and Demmel [2] is an improved version of earlier ideas of Ruhe [11], of Dongarra, Hammarling and Wilkinson [5], of Cao and Zhang [4] and of Ng and Parlett [9]. The following theorem from the last-mentioned paper forms the basis of the algorithm.

**Theorem 2.1** *Let $A$ be a matrix that is partitioned as follows,*

$$A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \tag{5}$$

*where $A_{11}$ is $p \times p$, $A_{22}$ is $q \times q$ and $p, q \in \{1, 2\}$. Assume that $A_{11}$ and $A_{22}$ have no eigenvalues in common, and that $X$ is the solution of the Sylvester equation*

$$A_{11}X - XA_{22} = A_{12}. \tag{6}$$

*Let $Q$ be an orthogonal matrix of size $(p+q) \times (p+q)$ such that*

$$Q^T \begin{bmatrix} -X \\ I_q \end{bmatrix} = \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{7}$$

*for some invertible matrix $R$. Then,*

$$Q^TAQ = \begin{bmatrix} \tilde{A_{11}} & \tilde{A_{12}} \\ 0 & \tilde{A_{22}} \end{bmatrix}, \tag{8}$$

*and $\tilde{A_{ii}}$ is similar to $A_{ii}$ for $i \in \{1, 2\}$.*

It seems that this completely solves the problem of how to swap adjacent diagonal blocks. Indeed, it does so in exact arithmetic. In practice however, one should be careful how to implement the above theorem.

### 2.2 Swapping in finite precision arithmetic

As observed in the analysis of [2], there are a number of stability issues that deserve some more attention. For example, the Sylvester equation (6) may be ill-conditioned, even when the eigenvalues of $A_{11}$ and $A_{22}$ are not very close. The correct measure for the stability of the Sylvester equation is the smallest singular value $\sigma$ of the linear operator $X \mapsto A_{11}X - XA_{22}$, also called the *separation* between $A_{11}$ and $A_{22}$. In case $A$ is non-normal, $\sigma$ may be much smaller than the eigenvalue gap between the two blocks. In [2], a number of safety-measures are proposed to handle this and other stability problems:

1 The two-by-two blocks are kept in standardized form: the diagonal entries are equal, and the off-diagonal elements of opposite sign.

2 Solve $A_{11}X - XA_{22} = \gamma A_{12}$ by Gaussian elimination with complete pivoting on the Kronecker product formulation of this equation, where $\gamma \leq 1$ is chosen to prevent overflow in (6).

3 The $QR$-decomposition in (7) is performed through Householder reflections. Note that if in the previous step we used a $\gamma \neq 1$, in (7) we need to replace $I_q$ by $\gamma I_q$.

4 Swapping is only performed if the norm of the $(2,1)$ block $\tilde{A}_{21}$ of the result after swapping will be small enough. In the LAPACK [1] implementation this condition is $\|\tilde{A}_{21}\|_\infty \leq 10\xi\|A\|_\infty$, where $\xi$ is the machine precision.

The authors of [2] claim (p. 79), that if these rules are obeyed, "this gives an absolute guarantee of backward stability", and they support this claim with an elaborate error analysis. However, they cannot guarantee that all swaps necessary for a given ordering will made. Indeed, in the last of the four safety measures above they suggest not to perform a swap in case instability is suspected.

### 2.3 Matlab algorithm design

In the Matlab code that we will present, we follow a slightly different approach as suggested in [2]. We will respect items 1-3 in the list of safety-measures above, but refrain from item 4. Being aware that this might introduce instability, we propose to do the following to detect instability *a posteriori*.

- Given a real Schur decomposition, compute *a priori* a complete list of all $n$ swaps that are necessary to obtain a certain ordering.

- Perform those swaps one by one, regardless of item 4 of the safety-measures, where standardization, if necessary, takes place before each swap. At each swap $j$, compute the quotient $q_j := \|\tilde{A}_{21}\|_\infty / (10\xi\|A\|_\infty)$ of that particular swap. The vector $q = (q_1, \ldots, q_n)$ will be called an *a posteriori* error indicator.

- If it turns out *a posteriori*, that all those quotients are smaller than one, accept the computed sorted real Schur form. If there are one or more quotients too large, one should be aware of a possibly inaccurate result.

### 2.3.1 Standardization of the two-by-two blocks

Prior to swapping two blocks, both blocks will be brought onto standardized form by an orthogonal similarity transformation. So, let a block $B$ and an orthogonal matrix $C$ be given as follows,

$$B := \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \quad \text{and} \quad C := \begin{bmatrix} c & s \\ -s & c \end{bmatrix}. \tag{9}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$ for some angle $\theta$, and assume that $b_{11} \neq b_{22}$, so standardization is needed. Computation of the $(1,1)$ and $(2,2)$ entries of $C^T BC$ and demanding that they are equal leads to the following equation for $c$ and $s$,

$$b_{11}(c^2 - s^2) + b_{22}(s^2 - c^2) = -2(b_{12} + b_{21})cs. \tag{10}$$

Since $c = 0$ contradicts $b_{11} \neq b_{22}$, we can introduce the variable $t = s/c$. This transforms (10) into

$$t^2 - 2\tau t - 1 = 0, \quad \text{where} \quad \tau := \frac{b_{12} + b_{21}}{b_{11} - b_{22}}. \tag{11}$$

With $t_s$ the smallest root of (11), we get that $\theta \leq \pi/4$ and

$$c = \frac{1}{\sqrt{1 + t_s^2}} \quad \text{and} \quad s = t_s c. \tag{12}$$

This completes the definition of an orthogonal transformation $C$ standardizing the block $B$.

### 2.3.2 Construction of an *a priori* list of swaps

We will use the bubble-sort algorithm to obtain the ordering of the diagonal blocks that the user has in mind. A useful data-structure is formed by an array $s$ that has as $j$-th entry, the position of the $j$-th block on the diagonal of the given real Schur form. For example, as depicted in Figure 4, if the quasi-triangle has, from upper left to lower right, blocks of sizes 2,2,1,2,1,2, then the array $s$ has as entries 1,3,5,6,8,9,11, where the last entry denotes the size of the matrix plus one for convenience.

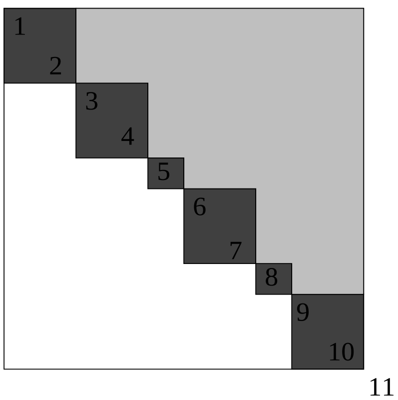


**Figure 4.** A real Schur form with block-sizes 2,2,1,2,1,2 gives rise to an array 1,3,5,6,8,9,11 determining the top-left position of each block. The bottom-right value 11 is a dummy value for convenience of programming.

The eigenvalues of each block are computed and an objective ordering is defined, depending on the wishes of the user. This objective ordering is a permutation of the given ordering and can therefore be realized by a sequence of, say, $n$ swaps of neighboring pairs. Such a sequence will be represented by a vector $p = (p_1, \ldots, p_n)$ where $p_j = k$ means that in swap number $j$, the $k$-th and $(k+1)$-st block should be swapped. We will call such a vector $p$ a *swaplist*.

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 3 | 3 | 3 | 3 |
| 2 | 3 | 1 | 1 | 4 | 4 | 4 |
| 3 | 2 | 2 | 4 | 1 | 2 | 2 |
| 4 | 4 | 4 | 2 | 2 | 1 | 5 |
| 5 | 5 | 5 | 5 | 5 | 5 | 1 |

**Example.** Applying the six swaps from the swaplist $p = (2, 1, 3, 2, 3, 4)$ turns the initial ordering $(1, 2, 3, 4, 5)$ into the objective ordering $(3, 4, 2, 5, 1)$. The step-by-step results are visible in the tabular on the left.

As mentioned before, the list of swaps necessary to obtain the objective ordering by bubble-sort will be made *a priori* and will be applied regardless of what may go wrong due to instability during the process. The error indicator $q = (q_1, \ldots, q_n)$ will give *a posteriori* information about the success of the procedure.

#### 2.3.3 Further building blocks: stable $LU$- and $QR$-factorization

Since Matlab does not have a function for $LU$-decomposition with complete pivoting, we include one ourselves. It is based on Algorithm 3.4.2 on page 118 of Golub and Van Loan [8]. The scaling factor $\gamma$ to prevent overflow (see the list of safety measures), will be chosen as the the smallest diagonal entry of the upper triangular factor $U$ that is computed.

The Matlab $QR$-factorization is built upon the LAPACK routine DGEQRF [1], which employs Householder factorization in double precision. So we can safely use the Matlab function **qr** for this purpose.

### 2.4 The Matlab function SRSchur

Our Matlab function **SRSchur**, which stands for **S**ort **R**eal **Schur**, orders a given real Schur form. The form in which we present the Matlab code of this function in the Appendix is not optimal. It contains many (sub-)function calls and is relatively slow. The advantage is that its structure, depicted in Figure 5 below, is clear.

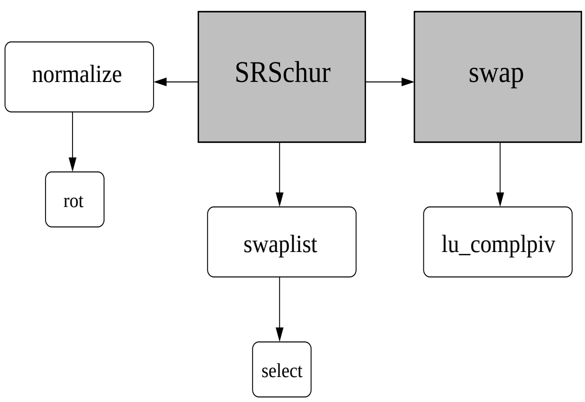


**Figure 5.** Sorting real Schur forms. Main function **SRSchur** calling sub-functions, of which the function **swap** is the most important.

The user interested in a faster implementation can easily put all pieces together in one file and remove all calls of subfunctions. We now list those sub-functions and explain their tasks.

- **[U,S] = normalize(U,S,v)** applies a Givens rotation such that the two-by-two diagonal block of $S$ situated at diagonal positions $v(1), v(2)$ is in standardized form. See Section 2.3.1.
- **Q = rot(X)** computes such a Givens rotation needed for normalization.
- **v = swaplist(p,s,z,b)** produces the list $v$ of swaps of neighboring blocks needed to order the eigenvalues assembled in the vector $v$ from closest to $z$ to farthest away from $z$, taking into account the parameter $b$ (see Section 2.3.2).
- **[val,pos] = select(p,z)** determines which element is next in the ordering.
- **[U,S] = swap(U,S,v,w)** swaps the two diagonal blocks at positions symbolized by the entries of $v$ and $w$. See Theorem 2.1.
- **[L,U,P,Q] = lu_complpiv(A)** computes the $LU$-decomposition of $A$ with complete pivoting, i.e., $PAQ = LU$, with permutations $P$ and $Q$ symbolized by vectors.

## 2.5 Numerical tests

We performed some numerical experiments to test our routines. Matlab version 5.3 was used, for which $\xi = 2.22e-16$. For comparison, we took the first three $4 \times 4$ matrices from Table 1 of [2]. Since in [2], the machine precision was only $\hat{\xi} = 1.192e-7$, we include a fourth matrix with a machine precision separation between the two blocks to be swapped. As in [2], we tested if the quantities

$$E_Q = \frac{\|I - Q^T Q\|_1}{\xi} \quad \text{and} \quad E_A = \frac{\|A - QAQ^T\|_1}{\xi\|A\|_1} \tag{13}$$

are around one. Instead of giving the eigenvalues before and after the swap, we present their relative perturbation resulting from the swap, i.e., if $\lambda$ is an eigenvalue before the swap, and $\hat{\lambda}$ the corresponding eigenvalue after the swap, we list the quantity

$$E_\lambda = \frac{|\lambda - \hat{\lambda}|}{\xi|\lambda|}. \tag{14}$$

We only present this quantity for one of the complex conjugates, so only one number per block.

The results are summarized in Table 1 below, and show that even in the case of the extremely small separation $\sim 10^{-17}$, the swaps are performed satisfactorily, although the relative error $\sim 10^3\xi$ in the eigenvalues is much bigger than for the first three matrices.

**Remark 2.2** In a version of the Bai-Demmel algorithm in which we did not normalize the two-by-two blocks and in which standard $LU$-decomposition was used, the results were comparable for the first three matrices, and only about a factor two worse for the last. This suggests that for applications in which the separation is moderate, this faster version could be feasible. However, the separation is usually not known in advance. This faster version of the algorithm can easily be obtained by adapting the code above. Just replace the function **lu_complpiv** with the Matlab built-in function **lu** and skip all the normalizations.

| matrix $A$ | separation | $E_Q$ | $E_A$ | $E_{\lambda_1}$ | $E_{\lambda_2}$ |
|---|---|---|---|---|---|
| $\begin{bmatrix} 2 & -87 & -20000 & 10000 \\ 5 & 2 & -20000 & -10000 \\ 0 & 0 & 1 & -11 \\ 0 & 0 & 37 & 1 \end{bmatrix}$ | $3 \times 10^{-1}$ | 2.005 | 3.2753 | 1.5280 | 3.1824 |
| $\begin{bmatrix} 1 & -3 & 3576 & 4888 \\ 1 & 1 & -88 & -1440 \\ 0 & 0 & 1.001 & -3 \\ 0 & 0 & 1.001 & 1.001 \end{bmatrix}$ | $8 \times 10^{-4}$ | 2.182 | 1.617 | 0 | 2.498 |
| $\begin{bmatrix} 1 & -100 & 400 & -1000 \\ 0.01 & 1 & 1200 & -10 \\ 0 & 0 & 1.001 & -0.01 \\ 0 & 0 & 100 & 1.001 \end{bmatrix}$ | $2 \times 10^{-7}$ | 2.014 | 1.958 | 0.707 | 3.161 |
| $\begin{bmatrix} 1 & -10^4 & 8812 & 4566 \\ 10^{-4} & 1 & -9 & 1200 \\ 0 & 0 & 10^{-5}+1 & -10^{-4} \\ 0 & 0 & 10^4 & 10^{-5}+1 \end{bmatrix}$ | $\sim 10^{-17}$ | 1.663 | 0.370 | 836.9 | 500.1 |

**Table 1.**

### 2.5.1 Ordering the eigenvalues of the GRCAR matrix

As a second experiment, we ordered the eigenvalues of the highly non-normal matrix GRCAR(n) taken from the Matlab Gallery Testmatrices. For the values $n = 50, 100, 200$, we first used the Matlab function **schur** to compute a real Schur decomposition. Then, **SRSchur** was called with target $z = 0$. In Table 2 below, we give the number of swaps that were made, and the quantities $E_Q$ and $E_A$. It should be noted that those contain the errors of both **schur** and **SRSchur**. In Figure 6, the a *posteriori* error indicators for the swaps needed to sort GRCAR(200) are given. They are all well below one, indicating a successful and stable ordering process.

We conclude from our experiments that the function **SRSchur** seems to perform the task of ordering the diagonal blocks relative to a given target in the complex well in a satisfactory manner.

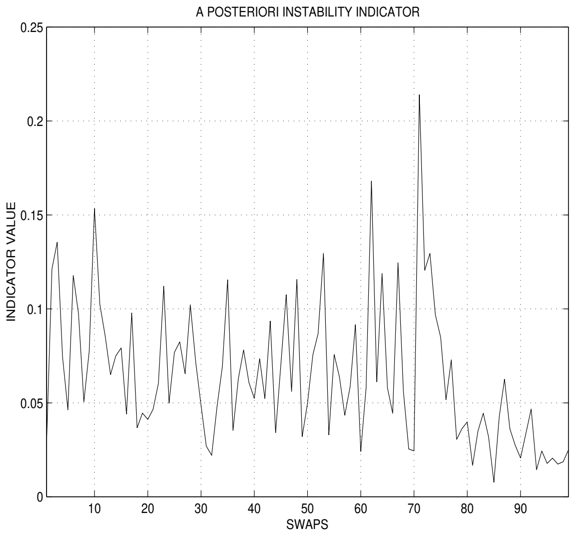


**Figure 6.** Instability indicator values for the 99 swaps needed to order the blocks of a Schur form of the GRCAR(200) matrix. This matrix is highly non-normal. All values are safely below one.

| grcar(n) | swaps | $E_Q$ | $E_A$ |
|---|---|---|---|
| 50 | 24 | $9.21 \times 10^1$ | $6.45 \times 10^1$ |
| 100 | 49 | $1.96 \times 10^2$ | $1.06 \times 10^2$ |
| 200 | 99 | $3.63 \times 10^2$ | $2.25 \times 10^2$ |

**Table 2.** Values of $E_Q$ and $E_A$ for the sorting of the GRCAR matrix. The table suggests an accumulation of errors linear in the number of swaps.

## 2.6 Alternative orderings

Using the Matlab code, either (i) $-b$ blocks or (ii) $b$ or $b+1$ eigenvalues or (iii) all blocks/eigenvalues, are ordered with respect to a target $z$ in the complex plane. After ordering, the top-left block is the one containing an eigenvalue closest to $z$.

Since basically, all eigenvalues of $A$ can be computed cheaply through the input real Schur form of $A$, the parameters $z$ and $b$ of the function **SRSchur** provide a wide range of possibilities to order the real Schur form according to the wishes of the user. For example, if it is needed to separate the eigenvalues with positive real part from the ones with non-positive real part, this can be established for example, by calling

**[q,r] = schur(A);**
**z = max(abs(eig(r)));**
**[Q,R,ap] = SRSchur(q,r,z,0);**

As a matter of fact, by using the parameter value $b = -1$, individual blocks can be moved to the top-left corner one by one. If the user wants the first three blocks to contain the eigenvalues $\lambda, \mu$ and $\nu$, then three consecutive calls with $b = 1$ and the targets in reverse order, $z = \nu, z = \mu$ and $z = \lambda$, will realize this ordering of the three blocks. Finally, note that also the function **select** can be changed easily according to the wishes of the user.

### 2.7 Finding eigenvectors of real eigenvalues

The ordering algorithm can be used to find invariant subspaces belonging to each group of eigenvalues by swapping this group to the top-left corner of the quasi-triangle. In case one is merely interested in an eigenvector belonging to a real eigenvalue $\lambda$, this can be done alternatively as follows.

Let $\lambda \in \mathbb{R}$ be given, then $M := R - \lambda I$ is singular. Let $j$ be such, that $m_{jj}$ is the first (closest to top-left) zero element on the diagonal of $M$. Then the top-left $(j-1) \times (j-1)$ part of $M$ is non-singular, and so the $j$-th column $m_j$ of $M$ is a linear combination of the first $j-1$ columns $m_1, \ldots, m_{j-1}$ of $M$. Which linear combination this is, can easily be deducted from solving the corresponding upper triangular system (using Matlab notation)

$$M(1:j-1, 1:j-1)y = M(1:j-1, j), \tag{15}$$

which produces an eigenvector $w := (y^*, -1, 0, \ldots, 0)^*$ of $R$. This eigenvector transforms to an eigenvector $v = Q^* w$ of $A$.

If the same eigenvalue appears more than once on the diagonal of $R$, then the dimension of the nullspace of $M$ is not necessarily larger than one. Only if $M(j,k) = 0$ as well, where $k$ is the position of the next zero diagonal entry, then another independent eigenvector can be found, as illustrated in Figure 7.

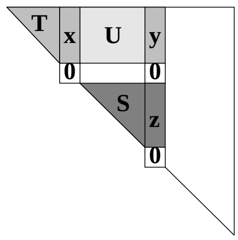


**Figure 7.** Upper triangular matrix $M$ with multiple zero eigenvalue. Solving $Tu = x$ yields a first eigenvector $(u^*, -1, 0, \ldots, 0)^*$. Then, solving the two upper triangular systems $Sw = z$ and $Tv = y - Uw$ provides a next independent eigenvector $(v^*, 0, w^*, -1, 0, \ldots, 0)^*$.

This procedure generalizes to higher dimensional eigenspaces. If, however, the eigenvalue is defective, we need to consider other methods to compute them. Recall that a defective eigenvalue is one with algebraic multiplicity strictly larger than the dimension of its eigenspace.

## 3 Computing the partial sorted real Schur form

Let $A$ be a real matrix, and $\tau$ a target value in the complex plane. We will outline how to gradually build a sorted real Schur decomposition of $A$, without computing a complete real Schur decomposition. This approach is necessary if the matrix is too large to compute a complete Schur decomposition, and is also feasible if only relatively few Schur vectors are needed. We will assume that we have a suitable eigensolver available.

### 3.1 Expanding the Real Schur Form

Suppose we are given an eigenvalue $\lambda$ of a real matrix $A$ closest to some target $\tau$. Since there may be more than one such eigenvalue, we just select one of them. We now distinguish two possibilities:

• If $\lambda$ is real, let $v$ be a unit length eigenvector belonging to $\lambda$. Then $Av = v\lambda$ is the first SPR Schur decomposition for $A$.

• If $\lambda = \gamma + \mu i$ with $\mu \neq 0$ then the first sorted partial real Schur decomposition is constructed from (4) by $QR$-decomposition of $(y|z)$ as follows,

$$QR := (y|z) \quad \text{such that} \quad AQ = QR \begin{bmatrix} \gamma & \mu \\ -\mu & \gamma \end{bmatrix} R^{-1}. \tag{16}$$

Going for an inductive approach, suppose that we have a sorted partial real Schur decomposition available consisting of an orthogonal matrix $Q$ with $k$ columns and a quasi-triangular matrix $T$ such that $AQ = QT$. Again, we distinguish two cases.

• Let $\lambda$ be a real eigenvalue of $A$ closest to $\tau$ and not yet included in $T$. The problem is that $\lambda$ may well be the second (or more) of a defective multiple eigenvalue whose eigenspace is already completely spanned by the columns of $Q$. So, merely trying to find a corresponding eigenvector and orthogonalizing it to $Q$ might eventually fail. The following observation leads to a way out. See also [6] Section 6.2.3. We need to find a real vector $q$ satisfying $Q^*q = 0$ and

$$A(Q|q) = (Q|q) \begin{bmatrix} R & s \\ 0 & \lambda \end{bmatrix}, \tag{17}$$

in order to expand our sorted partial real Schur decomposition. From (17) it follows that $q$ satisfies

$$Q^*q = 0 \quad \text{and} \quad (A - \lambda I)q - Qs = 0, \tag{18}$$

which yields that $s = Q^*(A - \lambda I)q$. Using this together with $(I - QQ^*)q = q$, we find

$$Q^*q = 0 \quad \text{and} \quad (I - QQ^*)(A - \lambda I)(I - QQ^*)q = 0. \tag{19}$$

This means that for all $y$, $q + Qy$ is in fact an eigenvector of the deflated matrix $\tilde{A}$

$$\tilde{A} := (I - QQ^*)A(I - QQ^*), \tag{20}$$

even when $\lambda$ is defective as eigenvalue of $A$ itself. Consequently, $q$ can be computed as such. Given a sorted partial real Schur decomposition and assuming that the next eigenvalue in line is real, we compute any eigenvector of $\tilde{A}$ belonging to the eigenvalue $\lambda$, and orthonormalize it to $Q$. After this, $s$ can be computed, and hence the sorted partial real Schur form is successfully expanded.

• Alternatively, suppose that the next candidate $\lambda = \gamma + \mu i$ in line is not real. Then we aim to find real $(u|v)$ such that $(Q|u|v)$ is orthogonal and

$$A(Q|u|v) = (Q|u|v) \begin{bmatrix} R & e & f \\ 0 & a & b \\ 0 & c & d \end{bmatrix}, \tag{21}$$

while $a, b, c, d, e, f$ are real and such that the two-by-two block formed by $a, b, c$, has $\lambda$ and $\overline{\lambda}$ as eigenvalues. In the analysis we will allow complex arithmetic. Indeed, the real case above does not use that $\lambda$ was real. So, we can conclude that there exists a non-real eigenvector $q$ of the deflated matrix $\tilde{A}$ from (20) belonging to $\lambda$, and with $s = Q^*(A - \lambda I)q$, the Schur form in (17) has real $Q$ and $R$ but generally non-real $q, s$ and $\lambda$. However, substituting $q = u + vi$ and $s = x + ti$, with real $u, v, x, t$, gives

$$A(u + vi) = (Q|u + vi) \begin{bmatrix} x + ti \\ \gamma + \mu i \end{bmatrix} = Q(x + ti) + (\gamma + \mu i)(u + vi). \tag{22}$$

Comparing real and imaginary parts then leads to

$$A(u|v) = Q(x|t) + (u|v) \begin{bmatrix} \gamma & -\mu \\ \mu & \gamma \end{bmatrix}. \tag{23}$$

Let $UT$ be a $QR$-decomposition of $(u|v)$, then

$$AU = Q(x|t)T^{-1} + UT \begin{bmatrix} \gamma & -\mu \\ \mu & \gamma \end{bmatrix} T^{-1}. \tag{24}$$

This means that in (21) we get $(e|f) = (x|t)T^{-1}$, and

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = T \begin{bmatrix} \gamma & -\mu \\ \mu & \gamma \end{bmatrix} T^{-1}. \tag{25}$$

In theory, it is now clear how a given partial real Schur form can be expanded. In practice, we will need an iterative eigensolver that can be fed with deflated matrices.

### 3.2 Choosing the eigensolver

In order to build a sorted partial real Schur form, we need to be able to compute eigenvalues of the deflated matrix $\tilde{A}$ closest to a given target. Each time that an eigenpair is computed, the matrix needs to be deflated with the Schur vectors found so far before computing the next eigenpair. This gives the computational scheme given as Algorithm 3.1.

Clearly, not any eigensolver can be used to solve such deflated matrix eigenvalue problems. We have supposed that $A$ is large and sparse, so explicit multiplication with the projection matrices $I - QQ^*$ is not feasible since this would, in general, ruin the sparsity. Apart from that, $I - QQ^*$ will be a full matrix itself. So, the best option would be an eigensolver that only uses the action $v \mapsto (I - QQ^*)A(I - QQ^*)v$, which can be performed in relatively cheap separate steps as long as $Q$ is a long and tall matrix.

The ideal and natural option in this context would be the Jacobi-Davidson algorithm [14] or the related Riccati based algorithms [3]. Those algorithms themselves already include deflation by the current eigenvector approximation in order to ease the computation of orthogonal corrections to those current approximations. Henceforth, they can easily be adapted to include the deflation due to already found Schur vectors assembled in the matrix $Q$. The Jacobi-Davidson QR and QZ algorithms [7] use precisely this idea, though merely for the general complex Schur form. The template given in Algorithm 3.1 is the adaption for real Schur forms.

ALGORITHM 3.1: Computing a Sorted Partial Real Schur Form

**input:** $A, n, \tau$
$Q = [\ ], R = [\ ]$;
**for** $k = 1$ **to** $n$
  $\tilde{A} = (I - QQ^*)A(I - QQ^*)$;
  $[q, \lambda]$ = *eigenpair of* $\tilde{A}$ *closest to* $\tau$ *with* $Q^*q = 0$;
  $s = Q^*(A - \lambda I)q$;
  **if** $\lambda$ **is real**
    $Q = [Q, q]$;
    $R = [R, s; 0, \lambda]$;
  **else**
    $[\gamma, \mu] = [\mathrm{re}(\lambda), \mathrm{im}(\lambda)]$;
    $[x, t] = [\mathrm{re}(s), \mathrm{im}(s)]$;
    $[u, v] = [\mathrm{re}(q), \mathrm{im}(q)]$;
    $[U, T] = \mathrm{qr}([u, v])$;
    $Q = [Q, U]$;
    $\Sigma = [x, t]T^{-1}$;
    $\Lambda = T[\gamma, -\mu; \mu, \gamma]T^{-1}$;
    $R = [R, \Sigma; 0, \Lambda]$;
  **end (if)**
**end (for)**

## 3.3 Conclusions

When the computation of a complete real Schur form of a real matrix $A$ is too expensive, a partial Schur form can be constructed using an inductive procedure in which $A$ is deflated in every step, in order to turn the Schur vectors into eigenvectors of the deflated matrix. The Jacobi-Davidson algorithm and variations seem good candidates for computing the eigendata of the deflated matrix.

## Acknowledgments

This paper was written in the framework of a project of the Royal Netherlands Academy of Arts and Sciences (KNAW). The support of KNAW is gratefully acknowledged. Moreover, the author thanks Mark Friedman from the University of Alabama in Huntsville for useful comments and questions and for applying the Matlab code in the context of continuation methods for invariant subspaces.

## Appendix

In this Appendix we present a complete version of the Matlab function **SRSchur.m** that can be used to sort the diagonal blocks of a given real Schur form. The main function **SRSchur.m** and its subfunctions can all be stored in the same file.

```
function [Q,R,ap] = SRSchur(Q,R,z,b);

% SYNTAX: [Q,R,ap] = SRSchur(Q,R,z,b)
%
% WRITTEN BY JAN BRANDTS, UTRECHT UNIVERSITY, FEBRUARY 2001
% The author does not wish to be held responsible for incorrect results that
% are due to possible errors in this code, nor for any of the consequences of
% such errors.
%
% INPUT: orthogonal real Q and quasi-triangular real R such that AQ=QR and a
% target z in the complex plane. The fourth parameter b determines the length
% of the ordering with respect to z to be produced:
%
% if b < 0 then -b blocks will be sorted,
% if b > 0 then  b or b+1 eigenvalues will be sorted, depending on the sizes
% of the blocks,
% if b = 0 then the whole real Schur form will be sorted.
%
% OUTPUT: orthogonal real Q and quasi-triangular real R such that AQ=QR with
% the diagonal blocks ordered with respect to the target z. The number of
% ordered blocks/eigenvalues is determined by the parameter b.
%
% A vector ap warns for inaccuracy of the solution: if an entry of ap exceeds
% one, the solution is unreliable.
%
% SUBFUNCTIONS: normalize.m, swaplist.m, select.m, swap.m, lu_complpiv.m
%
% SEE ALSO: schur.m, rsf2csf.m

r = find(abs(diag(R,-1)) > 100*eps);    % Detect subdiagonal nonzero entries,
s = 1:size(R,1)+1;                     % construct from them a vector s with
s(r+1) = [];                          % the top-left positions of each block.

for k=1:length(s)-1;                                % Ranging over all blocks,
  sk = s(k);
  if s(k+1)-sk == 2                                     % If the block is 2x2,
    [Q,R] = normalize(Q,R,sk:s(k+1)-1);                          % normalize it
    p(k)  = R(sk,sk)+sqrt(R(sk+1,sk)*R(sk,sk+1));    % store the eigenvalues,
  else                % (the one with positive imaginary part is sufficient).
    p(k)  = R(s(k),s(k));   % If the block is 1x1, only store the eigenvalue.
  end
end

for k = swaplist(p,s,z,b);           % For k ranging over all neighbor-swaps
  v      = s(k):s(k+1)-1;            % collect the coordinates of the blocks,
  w      = s(k+1):s(k+2)-1;
  nrA    = norm(R([v,w],[v,w]),inf); % compute norm of the matrix A from (6),
```

```
  [Q,R]  = swap(Q,R,v,w);                                     % swap the blocks,
  s(k+1) = s(k)+s(k+2)-s(k+1);                      % update positions of blocks,
  v      = s(k):s(k+1)-1;                             % update block-coordinates,
  w      = s(k+1):s(k+2)-1;
  if length(v)==2                                  % if the first block is 2 x 2,
    [Q,R] = normalize(Q,R,v);                                     % normalize it,
  end
  if length(w)==2                                 % if the second block is 2 x 2,
    [Q,R] = normalize(Q,R,w);                                     % normalize it,
  end
  ap(k) = norm(R(w,v),inf)/(10*eps*nrA); % measure size of bottom-left block
end                                                       % (see p.6, Sect. 2.3).

R = R - tril(R,-2);                               % Zero the below-block entries
for j=2:length(s)-1; R(s(j),s(j)-1)=0; end   % to get a quasi-triangle again.

function [U,S] = normalize(U,S,v);
Q    = rot(S(v,v));       % Determine the Givens rotation needed for standar-
S(:,v) =  S(:,v)*Q;       % dization - and apply it left and right to S, and
S(v,:) = Q'*S(v,:);      % right to U. Only rows and columns with indices in
U(:,v) =  U(:,v)*Q;                    % the vector v can be affected by this.

function Q = rot(X);
c = 1; s = 0;                         % Start with the identity transformation,
if X(1,1)~=X(2,2);                          % and if needed, change it into ...
  tau   = (X(1,2)+X(2,1))/(X(1,1)-X(2,2));
  off   = sqrt(tau^2+1);
  v     = [tau - off, tau + off];
  [d,w] = min(abs(v));
  c     = 1/sqrt(1+v(w)^2);            % ... the cosine and sine as given in
  s     = v(w)*c;                                             % Section 2.3.1.
end
Q = [c -s;s c];

function v = swaplist(p,s,z,b);
n = length(p);
k = 0; v = [];
srtd = 0;                                         % Number of sorted eigenvalues.
q = diff(s);                                               % Compute block sizes.
fini = 0;
while ~fini
  k        = k+1;
  [dum,j]  = select(p(k:n),z); % Determine which block will go to position k,
  p(k:n+1) = [p(j+k-1) p(k:n)];               % insert this block at position k,
  p(j+k)   = [];                       % and remove it from where it was taken.
  q(k:n+1) = [q(j+k-1) q(k:n)];                 % Similarly for the block-sizes.
  q(j+k)   = [];
  v        = [v,j+k-2:-1:k];       % Update the list of swaps for this block.
  srtd     = srtd + q(k);          % Update the number of sorted eigenvalues.
  fini    = (k==n-1)|(k==-b)|(srtd==b)|((srtd==b+1)&(b~=0));
end
```

```
function [val,pos] = select(p,z);
y = real(z)+abs(imag(z))*i;            % Move target to the upper half plane.
[val pos] = min(abs(p-y));                % Find block closest to the target.

function [U,S] = swap(U,S,v,w);
[p,q] = size(S(v,w)); Ip = eye(p); Iq = eye(q);    % p and q are block-sizes
r = [];
for j=1:q                    % Vectorize right-hand side for Kronecker product
  r = [r;S(v,w(j))];            % formulation of the Sylvester equation (7).
end
K = kron(Iq,S(v,v))-kron(S(w,w)',Ip);      % Kronecker product system matrix.
[L,H,P,Q] = lu_complpiv(K);                 % LU-decomposition of this matrix.
gamma = min(abs(diag(H)));   %            Scaling factor to prevent overflow.
sigp = 1:p*q;
for k = 1:p*q-1;                           % Implement permutation P of the LU-
  sigp([k,P(k)]) = sigp([P(k),k]);              % decomposition PAQ = LU ...
end
r = gamma*r(sigp);              % ... scale and permute the right-hand side.
x = (H\(L\r));                       % and solve the two triangular systems.
sigq = 1:p*q;
for k = 1:p*q-1;                           % Implement permutation Q of the LU-
  sigq([k,Q(k)]) = sigq([Q(k),k]);              % decomposition PAQ = LU ...
end
x(sigq) = x;                                     % ... and permute the solution.
X = [];
for j=1:q                       % De-vectorize the solution back to a block,
   X = [X,x((j-1)*p+1:j*p)];           % or, quit the Kronecker formulation.
end
[Q,R]      = qr([-X;gamma*Iq]);          % Householder QR-decomposition of X.
S(:,[v,w]) = S(:,[v,w])*Q;      % Perform the actual swap by left- and right-
S([v,w],:) = Q'*S([v,w],:);                   % multiplication of S by Q, and,
U(:,[v,w]) = U(:,[v,w])*Q;                  % right-multiplication of U by Q.

function [L,U,P,Q] = lu_complpiv(A);
P = []; Q = []; n = size(A,1);
for k=1:n-1;                             % See Golub and Van Loan, p. 118 for
  [a,r] = max(abs(A(k:n,k:n)));          % comments on this LU-decomposition
  [dummy,c] = max(abs(a));                             % with complete pivoting.
  cl  = c+k-1;
  rw  = r(c)+k-1;
  A([k,rw],:) = A([rw,k],:);
  A(:,[k,cl]) = A(:,[cl,k]);
  P(k) = rw; Q(k) = cl;
  if A(k,k) ~= 0;
    rs = k+1:n;
    A(rs,k)  = A(rs,k)/A(k,k);
    A(rs,rs) = A(rs,rs)-A(rs,k)*A(k,rs);
  end
end
U = tril(A')'; L = tril(A,-1) + eye(n);
```

## References


[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney, S. Ostrouchov and D. Sorensen (1992). LAPACK Users' Guide, Release 1.0, SIAM.

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

[3] J.H. Brandts (2000). A Riccati Algorithm for Eigenvalues and Invariant Subspaces, *Preprint nr. 1150 of the Department of Mathematics, Utrecht University, Netherlands.*

[4] Z. Cao and F. Zhang (1981). *Direct methods for ordering eigenvalues of a real matrix (in Chinese)*, Chinese Univ. J. Comput. Math., 1:27–36.

[5] J. Dongarra, S. Hammarling and J. Wilkinson (1992). *Numerical considerations in computing invariant subspaces*, SIAM J. Math. Anal. Appl., 13:145–161.

[6] D. Fokkema (1996). Subspace methods for linear, nonlinear, and eigen problems, *PhD thesis, Mathematics Department, Utrecht University, Netherlands*, ISBN 90-393-1097-1.

[7] D.R. Fokkema, G.L.G. Sleijpen and H.A. Van der Vorst (1999). Jacobi-Davidson style QR and QZ algorithms for the reduction of matrix pencils, *SIAM J. Sci. Comput.*, 20:94–125.

[8] G.H. Golub and C.F. van Loan (1996). Matrix Computations (third edition), *The John Hopkins University Press, Baltimore and London.*

[9] K.C. Ng and B.N. Parlett (1988). Development of an accurate algorithm for EXP(Bt), Part I, Programs to swap diagonal blocks, Part II, CPAM-294, *Univ. of California, Berkeley.*

[10] M. Robbé and M. Sadkane (2000). *Riccati-based preconditioner for computing invariant subspaces of large matrices. Part I: theoretical aspect.* Report.

[11] A. Ruhe (1970). *An algorithm for numerical determination of the structure of a genneral matrix*, BIT 10:196–216.

[12] Y. Saad (1992). Numerical methods for large eigenvalue problems. *Manchester University Press, Manchester.*

[13] I. Schur (1909). *On the characteristic roots of a linear substitution with an application to the theory of integral equations (in German).* Math. Ann. 66, 488-510.

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