SIAM J. MATRIX ANAL. APPL.
Vol. 22, No. 1, pp. 1–19

COMPUTING THE SVD OF A GENERAL MATRIX PRODUCT/QUOTIENT[*]

GENE GOLUB[†], KNUT SØLNA[‡], AND PAUL VAN DOOREN[§]

Abstract. In this paper we derive a new algorithm for constructing a unitary decomposition of a sequence of matrices in product or quotient form. The unitary decomposition requires only unitary left and right transformations on the individual matrices and amounts to computing the generalized singular value decomposition of the sequence. The proposed algorithm is related to the classical Golub–Kahan procedure for computing the singular value decomposition (SVD) of a single matrix in that it constructs a bidiagonal form of the sequence as an intermediate result. When applied to two matrices this new method is an alternative way of computing the quotient and product SVD and is more economical than current methods.

Key words. mumerical methods, generalized singular values, products of matrices, quotients of matrices

AMS subject classification. 65F15

PII. S0895479897325578

Introduction. The two basic unitary decompositions of a matrix A yielding some spectral information are the Schur form $A = UTU^*$—where U is unitary and T is upper triangular—and the singular value decomposition (SVD) $A = U\Sigma V^*$—where U and V are unitary and Σ is diagonal (for the latter A does not need to be square). It is interesting to note that both forms are usually computed by a QR-like iteration [7]. The SVD algorithm of Golub–Kahan [6] is indeed an implicit QR algorithm applied to the Hermitian matrix A^*A. When looking at unitary decompositions involving *two* matrices, say, A and B, a similar implicit algorithm was given in [10] and is known as the QZ algorithm. It computes $A = QT_aZ^*$ and $B = QT_bZ^*$, where Q and Z are unitary and T_a and T_b are upper triangular. This algorithm is in fact the QR algorithm again performed implicitly on the quotient $B^{-1}A$. The corresponding decomposition is therefore also known as the *generalized Schur form.*

When considering the generalized SVD of two matrices, appearing as a quotient $B^{-1}A$ or a product BA, the currently used algorithm is *not* of QR type but of a Jacobi type. The reason for this choice is that Jacobi methods easily extend to products and quotients. Unfortunately, the Jacobi algorithm typically has a (moderately) higher complexity than the QR algorithm. Yet, so far, nobody proposed an implicit QR-like method for the SVD of a product or quotient of two matrices.

[*]Received by the editors August 20, 1997; accepted for publication (in revised form) by L. Eldén November 15, 1999; published electronically May 31, 2000. This paper contains research results of the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister's Office for Science, Technology and Culture. The scientific responsibility rests with its authors.

http://www.siam.org/journals/simax/22-1/32557.html

[†]Computer Science Department, Stanford University, Stanford, CA 94305-9025 (golub@sccm.stanford.edu). This author was partially supported by the National Science Foundation under grants DMS-9105192 and DMS-9403899.

[‡]SC-CM, Stanford University, Stanford, CA 94305-9025 (solna@sccm.stanford.edu). This author was partially supported by The Research Council of Norway.

[§]Cesame, Université Catholique de Louvain, Louvain-la-Neuve B1348, Belgium (vdooren@anma.ucl.ac.be). This author was partially supported by the National Science Foundation under grant CCR-96-19596.

In this paper we show that, in fact, such an implicit algorithm is easy to derive and that it even extends straightforwardly to sequences of products/quotients of several matrices. Moreover, the complexity will be shown to be lower than for the corresponding Jacobi-like methods.

1. Implicit SVD.

Consider the problem of computing the SVD of a matrix A that is an expression of the following type:

$$A = A_K^{s_K} \cdots A_2^{s_2} \cdot A_1^{s_1}, \tag{1}$$

where $s_i = \pm 1$, i.e., a sequence of products or quotients of matrices. For simplicity we assume that the A_i matrices are square $n \times n$ and invertible. It was pointed out in [3] that one can always perform a preliminary QR-like reduction that extracts from a sequence of matrices with compatible dimensions another sequence of square invertible matrices with the same generalized singular values as the original sequence. We refer to [3] for the details of this reduction and will treat here only the case of square invertible matrices. While it is clear that one has to perform left and right transformations on A to get $U^*AV = \Sigma$, these transformations will affect only A_K and A_1. Beyond this, one can insert an expression $Q_i^* Q_i = I_n$ between every pair $A_{i+1}^{s_{i+1}} A_i^{s_i}$ in (1). If we also define $Q_K \doteq U$ and $Q_0 \doteq V$, we arrive at the following expression:

$$U^*AV = (Q_K^* A_K^{s_K} Q_{K-1}) \cdots (Q_2^* A_2^{s_2} Q_1) \cdot (Q_1^* A_1^{s_1} Q_0). \tag{2}$$

With the degrees of freedom present in these $K+1$ unitary transformations Q_i at hand, one can now choose each expression $Q_i^* A_i^{s_i} Q_{i-1}$ to be upper triangular. Note that the expression $Q_i^* A_i^{s_i} Q_{i-1} = T_i^{s_i}$ with T_i upper triangular can be rewritten as

$$Q_i^* A_i Q_{i-1} = T_i \quad \text{for } s_i = 1 \,, \qquad\qquad Q_{j-1}^* A_j Q_j = T_j \quad \text{for } s_j = -1. \tag{3}$$

From the construction of a normal QR decomposition, it is clear that while making the matrix A upper triangular, this "freezes" only one matrix Q_i per matrix A_i. The remaining unitary matrix leaves enough freedom to finally diagonalize the matrix A as well. Since (2) computes the singular values of (1), it is clear that such a result can be obtained only by an *iterative procedure*. On the other hand, one intermediate form that is used in the Golub–Kahan SVD algorithm [6] is the bidiagonalization of A and this can be obtained in a *finite recurrence*. We show in the next section that the matrices Q_i in (2) can be constructed in a finite number of steps in order to obtain a bidiagonal $Q_K^* A Q_0$ in (2). In carrying out this task one should try to do as much as possible implicitly. Moreover, one would like the total complexity of the algorithm to be comparable to, or less than, the cost of K singular value decompositions. This means that the complexity should be $\mathrm{O}(Kn^3)$ for the whole process.

2. Implicit bidiagonalization.

We now derive such an implicit reduction to bidiagonal form. Below $\mathcal{H}(i,j)$ denotes the group of *Householder* transformations having (i,j) as the range of rows/columns they operate on. Similarly $\mathcal{G}(i,i+1)$ denotes the group of *Givens* transformations operating on rows/columns i and $i+1$. We first consider the case where all $s_i = 1$. We thus have only a product of matrices A_i and in order to illustrate the procedure we show its evolution operating on a product of three matrices only, i.e., $A_3A_2A_1$. Below is a sequence of displays of the matrix product that illustrates the evolution of the bidiagonal reduction. Each display indicates the pattern of zeros ("0") and nonzeros ("x") in the three matrices.

First perform a Householder transformation $Q_1^{(1)} \in \mathcal{H}(1,n)$ on the rows of A_1 and the columns of A_2. Choose $Q_1^{(1)}$ to annihilate all but one element in the first column of A_1:

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

Then perform a Householder transformation $Q_2^{(1)} \in \mathcal{H}(1,n)$ on the rows of A_2 and the columns of A_3. Choose $Q_2^{(1)}$ to annihilate all but one element in the first column of A_2:

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

Then perform a Householder transformation $Q_3^{(1)} \in \mathcal{H}(1,n)$ on the rows of A_3. Choose $Q_3^{(1)}$ to annihilate all but one element in the first column of A_3:

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

Note that this third transformation yields the same form also for the product of the three matrices:

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

At this stage we are interested in the *first row* of this product (indicated by boldface $\mathbf{x}$'s above). This row can be constructed as the product of the first row of A_3 with the matrices to the right of it, and this requires only $\mathrm{O}(Kn^2)$ flops. Once this row is constructed we can find a Householder transformation $Q_0^{(1)} \in \mathcal{H}(2,n)$ operating on the last $(n-1)$ elements which annihilates all but two elements (the colon,":", is used as it is in MATLAB):

$$A_3(1,:)A_2A_1Q_0^{(1)} = \begin{bmatrix} x & x & 0 & 0 & 0 \end{bmatrix}. \tag{4}$$

This transformation is then applied to A_1 only and completes the first stage of the bidiagonalization since

$$Q_K^{(1)*} A Q_0^{(1)} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

The second stage of the bidiagonalization is analogous to the first; it differs only in that the transformations operate only on rows/columns 2 to n. The Householder transformations $Q_i^{(2)} \in \mathcal{H}(2,n)$ for $1 \le i \le 3$ are chosen to eliminate elements 3 to n in the second columns of A_i in the manner described above. The transformation $Q_0^{(2)} \in \mathcal{H}(3,n)$ operates on the last $(n-2)$ elements of the second row of the product and annihilates all but two elements:

$$A_3(2,:) A_2 A_1 Q_0^{(2)} = \begin{bmatrix} 0 & x & x & 0 & 0 \end{bmatrix}. \tag{5}$$

This transformation is applied to A_1 only, completing the second step of the bidiagonalization of A:

$$Q_K^{(2)*} Q_K^{(1)*} A Q_0^{(1)} Q_0^{(2)} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{bmatrix}.$$

It is now clear from the context how to proceed further with this algorithm to obtain after $n-1$ stages:

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 \\ 0 & 0 & x & x & 0 \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix}.$$

Note that we never construct the whole product $A = A_3 A_2 A_1$, but rather compute one of its rows when needed for constructing the transformations $Q_0^{(i)}$. The only matrices that are kept in memory and updated are the A_i matrices and possibly Q_K and Q_0 if we require the singular vectors of A afterwards.

The complexity of this bidiagonalization step is easy to evaluate. Each matrix A_i gets pre- and postmultiplied with essentially n Householder transformations of decreasing range. For updating all A_i we therefore need $10Kn^3/3$ flops, and for updating Q_K and Q_0 we need $4n^3$ flops. For constructing the required row vectors of A we need $(K-1)n^3/3$ flops. Overall we thus need on the order of $11Kn^3/3$ flops for the construction of the triangular T_i and $4n^3$ for the outer transformations Q_K and Q_0. Essentially this is $11n^3/3$ flops per updated matrix.

If we now have some of the $s_i = -1$, we cannot use Householder transformations anymore on all matrices. Indeed, in order to construct the rows of A when needed, the matrices A_i for which $s_i = -1$ have to be triangularized first, say, with a QR factorization. The QR factorization is performed in an initial step and uses Householder transformations. From there on the same procedure as above is followed, but this implies using Givens rotations in certain steps of the bidiagonalization. For simplicity, we illustrate this on a matrix $A = A_3 A_2^{-1} A_1$. We first apply a left transformation

$Q_2^{(0)}$ (using a sequence of Householder transformations) that triangularizes A_2 from the left and also apply this to the rows of A_1. The resulting triple then has the form

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix}.$$

We then apply a unitary transformation $Q_1^{(1)}$ to the rows of A_1 to eliminate elements 2 to n in its first column using Givens transformations $G_1 \in \mathcal{G}(n-1, n)$ until $G_{n-1} \in \mathcal{G}(1, 2)$. Below we indicate in which order these zeros are created in the first column of A_1 by their index:

$$\begin{bmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0_4 & x & x & x & x \\ 0_3 & x & x & x & x \\ 0_2 & x & x & x & x \\ 0_1 & x & x & x & x \end{bmatrix}.$$

These transformations also have to be applied to the left of A_2, but the use of Givens rotations allows us to update the triangularized matrix A_2, while keeping it upper triangular: each time a Givens rotation applied to the left of A_2 destroys its triangular form, another Givens rotation is applied to the right of A_2 in order to restore its triangular form. (The same technique is used, for instance, in keeping the B matrix upper triangular in the QZ algorithm applied to $B^{-1}A$.) Let $Q_2^{(1)}$ be the product of the Givens rotations applied to the right of A_2, then $Q_2^{(1)}$ also has to be applied to the right of A_3. Finally, for the column transformation $Q_3^{(1)}$ of A_3 eliminating elements 2 to n of its first column, we can again use a Householder transformation $H_5 \in \mathcal{H}(1, n)$. After this fifth transformation, the resulting triple has the form

$$\begin{bmatrix} x & x & x & x & x \\ 0_5 & x & x & x & x \\ 0_5 & x & x & x & x \\ 0_5 & x & x & x & x \\ 0_5 & x & x & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \end{bmatrix} \begin{bmatrix} x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix},$$

which clearly has a first column with only its leading element different from 0. Its first row can easily be constructed, and we then apply a Householder transformation $Q_0^{(1)} \in \mathcal{H}(2, n)$ annihilating all but two elements as indicated in (4). This completes the first stage of the bidiagonalization of $A = A_3 A_2^{-1} A_1$. Subsequent steps are similar but operate on matrices of decreasing dimensions.

Notice that all transformations Q_i and Q_{i-1} applied to a matrix A_i with index $s_i = -1$ have to be of Givens type, which require more flops than Householder transformations for the same number of annihilated elements. So the more negative exponents we have, the more expensive the overall algorithm becomes. Without loss of generality, we can assume that at most half of the indices are equal to -1, since otherwise we can compute the SVD of A^{-1} rather than that of A (all matrices A_i were assumed to be invertible). The situation with the highest computational complexity is thus when every other index s_i is negative, since then all transformations but one have to be of Givens type. Let us analyze the case where K is even and $s_{2i} = -1,\ s_{2i-1} = 1,\ i = 1, \ldots, \frac{K}{2}$. The preliminary QR reduction of the matrices A_{2i} requires $\frac{4}{3}n^3\frac{K}{2}$ flops and $2n^3\frac{K}{2}$ flops for also updating the matrices A_{2i-1} with these transformations. From there on, each matrix undergoes $\frac{n(n-1)}{2}$ Givens rotations on the left and on the right. For the triangular matrices A_{2i} this requires a total of $3n^2\frac{K}{2}$ flops, whereas for the (originally dense) matrices A_{2i-1} this requires a total of $5n^3\frac{K}{2}$ flops. For constructing the required row vectors of A we need $(K-1)n^3/3$ flops as before. Finally, updating the matrix Q_0 via Givens transformations and Q_K via Householder transformations requires $2n^2$ and $3n^3$ flops, respectively. The worst-case complexity of the general case is thus $6n^3K$ flops for obtaining the triangular matrices T_i and $5n^3$ flops for the outer transformations Q_0 and Q_K. This is about 60% more than for the product case.

3. Error analysis. In the previous section we showed how to obtain an estimate of a bidiagonal decomposition of the matrix product/quotient. We now turn to the problem of obtaining accurate estimates of the singular values. This warrants a discussion of the errors committed in the bidiagonalization step.

The use of Householder and Givens transformations for all operations in the bidiagonalization step guarantees that the obtained matrices T_i in fact correspond to slightly perturbed data as follows:

$$T_i = Q_i^*(A_i + \delta A_i)Q_{i-1},\ s_i = 1, \qquad T_j = Q_{j-1}^*(A_j + \delta A_j)Q_j,\ s_j = -1, \tag{6}$$

where

$$\|\delta A_i\| \le \epsilon c_n \|A_i\|\ , \qquad \|Q_i^* Q_i - I_n\| \le \epsilon d_n, \tag{7}$$

with ϵ the machine precision and $c_n,\ d_n$ moderate constants depending on the problem size n. This is obvious since each element transformed to zero can indeed be put equal to zero without affecting the ϵ bound (see [11], [7]).

The situation is different for the elements of A since they are not stored explicitly in the computer. How does one proceed further to compute the generalized singular values of A? Once the triangular matrices T_i are obtained, it is easy and cheap to *reconstruct* the bidiagonal:

$$T_K^{s_k} \cdots T_2^{s_2} \cdot T_1^{s_1} = B = \begin{bmatrix} q_1 & e_2 & o_{1,3} & \cdots & o_{1,n} \\ & q_2 & e_3 & \ddots & \vdots \\ & & \ddots & \ddots & o_{n-2,n} \\ & & & \ddots & e_n \\ & & & & q_n \end{bmatrix}, \tag{8}$$

and then compute the singular values of the bidiagonal in a *standard* way. The diagonal elements q_i are indeed just a product of the corresponding diagonal elements of the T_j matrices, possibly inverted:

$$q_i = t_{K_{i,i}}^{s_K} \cdots t_{2_{i,i}}^{s_2} \cdot t_{1_{i,i}}^{s_1},$$

and the off-diagonal elements e_i can be computed from the corresponding 2×2 diagonal blocks (with index $i-1$ and i) of the T_j matrices. It is clear that the q_i can be computed in a backward stable way since all errors can be superimposed on the diagonal elements $t_{j_{i,i}}$ of the matrices T_j. For the errors incurred when computing the e_i one needs a more detailed analysis. We show below that the backward errors can be superimposed on the off-diagonal elements $t_{j_{i-1,i}}$ of T_j without creating any conflicts with previously constructed backward errors, and we derive bounds for these backward errors. From the vector recurrence

$$\begin{bmatrix} e \\ q \end{bmatrix} := \begin{bmatrix} t_{j_{i-1,i-1}} & t_{j_{i-1,i}} \\ 0 & t_{j_{i,i}} \end{bmatrix}^{s_j} \cdot \begin{bmatrix} e \\ q \end{bmatrix} \tag{9}$$

we easily derive the following algorithm used for computing q_i and e_i for $i = 1, \ldots, n$.

```
$q := 1;\ e := 0;$
    for $j = 1 : K$
        if $s_j = 1$, then $e := e * t_{j_{i-1,i-1}} + q * t_{j_{i-1,i}};\ q := q * t_{j_{i,i}};$
                  else $q := q/t_{j_{i,i}};\ e := (e - q * t_{j_{i-1,i}})/t_{j_{i-1,i-1}};$
    end
    $q_i := q;\ e_i := e;$
```

Note that for $i = 1$ the same recurrence holds without the expressions involving e. From these recurrences it is clear that the calculation of q_i involves one flop per step j and hence a total of K rounding errors which can be superimposed on the diagonal elements $t_{j_{i,i}}$:

$$\begin{aligned} q_i &= comp(t_{K_{i,i}}^{s_K} \cdots t_{1_{i,i}}^{s_1}) \\ &= \bar{t}_{K_{i,i}}^{s_K} \cdots \bar{t}_{1_{i,i}}^{s_1} \qquad \text{with} \quad \bar{t}_{j_{i,i}} = t_{j_{i,i}}(1 + \epsilon_{i,j}), \ \ |\epsilon_{i,j}| < \epsilon \end{aligned} \tag{10}$$

with *comp* denoting a floating point operator. For the calculation of e_i there are 3 flops per step j and hence a total of $3K$ roundings which have to be superimposed on the $t_{j_{i-1,i}}$ elements. Fortunately, e_j is a sum of K terms which contain each a *different* element $t_{j_{i-1,i}}$ as a factor. We illustrate this for $K = 4$ and $s_j = 1$, highlighting the relevant elements:

$$\begin{aligned} e_i = \ comp(&\mathbf{t}_{4_{i-1,i}} \cdot t_{3_{i,i}} \cdot t_{2_{i,i}} \cdot t_{1_{i,i}} \\ &+ t_{4_{i-1,i-1}} \cdot \mathbf{t}_{3_{i-1,i}} \cdot t_{2_{i,i}} \cdot t_{1_{i,i}} \\ &+ t_{4_{i-1,i-1}} \cdot t_{3_{i-1,i-1}} \cdot \mathbf{t}_{2_{i-1,i}} \cdot t_{1_{i,i}} \\ &+ t_{4_{i-1,i-1}} \cdot t_{3_{i-1,i-1}} \cdot t_{2_{i-1,i-1}} \cdot \mathbf{t}_{1_{i-1,i}}). \end{aligned} \tag{11}$$

The $3K$ rounding errors can thus easily be superimposed on these different elements $t_{j_{i-1,i}}$, $j = 1, \ldots, K$. But since we have already superimposed errors on the all-diagonal elements $t_{j_{i,i}}$ we have to add these perturbations here as well. For $s_j = 1$ we thus have

$$\begin{aligned} e_i = \ &\bar{t}_{4_{i-1,i}} \cdot \bar{t}_{3_{i,i}} \cdot \bar{t}_{2_{i,i}} \cdot \bar{t}_{1_{i,i}} \\ &+ \bar{t}_{4_{i-1,i-1}} \cdot \bar{t}_{3_{i-1,i}} \cdot \bar{t}_{2_{i,i}} \cdot \bar{t}_{1_{i,i}} \\ &+ \bar{t}_{4_{i-1,i-1}} \cdot \bar{t}_{3_{i-1,i-1}} \cdot \bar{t}_{2_{i-1,i}} \cdot \bar{t}_{1_{i,i}} \\ &+ \bar{t}_{4_{i-1,i-1}} \cdot \bar{t}_{3_{i-1,i-1}} \cdot \bar{t}_{2_{i-1,i-1}} \cdot \bar{t}_{1_{i-1,i}}, \end{aligned} \tag{12}$$

where $(K-1)$ additional roundings are induced for each factor. Therefore, we have $\bar{t}_{j_{i-1,i}} = t_{j_{i-1,i}}(1 + \eta_{i,j})$, $|\eta_{i,j}| < (4K-1)\epsilon/(1-(4K-1)\epsilon)$. When some of the $s_j = -1$ the above expression is similar: the $t_{j_{i,i}}$ then appear as inverses, some $+$ signs change to $-$ signs, and an additional factor $1/(t_{j_{i-1,i-1}} t_{j_{i,i}})$ appears in the jth term if $s_j = -1$. So in the worst case $(K+1)$ additional roundings are introduced for each factor and the obtained bound is then $|\eta_{i,j}| < (4K+1)\epsilon/(1-(4K+1)\epsilon)$. In the worst case the errors yield a backward perturbation $\|\delta T_j\|$ which is thus bounded by $5K\epsilon\|T_j\|$ and hence much smaller than the errors δA_j incurred in the triangularization process. The perturbation effect of computing the elements q_i and e_i is thus negligible compared to that of the triangularization. We thus showed that the computed bidiagonal corresponds *exactly* to the bidiagonal of the product of slightly perturbed triangular matrices T_j, that in turn satisfy the bounds (6)–(7). Unfortunately, nothing of the kind can be guaranteed for the elements $o_{i,j}$ in (8), which are supposed to be zero in exact arithmetic. Notice that the element e_{i+1} is obtained as the norm of the vector on which a Householder transformation is applied:

$$|e_{i+1}| = \|T_K^{s_K}(i,i:n)T_{K-1}^{s_{K-1}}(i:n,i:n)\cdots T_1^{s_1}(i:n,i+1:n)\|,$$

where we used the MATLAB notation for subarrays: $T_1^{s_1}(i:n,i+1:n)$ is thus the submatrix of $T_1^{s_1}$ with row indices i to n and column indices $i+1$ to n. If all $s_i = 1$ we can obtain by straightforward perturbation results of matrix vector products, a bound of the type

$$|o_{i,j}| \leq \epsilon c_n \|T_K(i,i:n)\| \cdot \|T_{K-1}(i:n,i:n)\| \cdots \|T_1(i:n,i:n)\|.$$

If not all $s_i = 1$ we need to also use perturbation results of solutions of systems of equations, since we need to evaluate the last $n-i$ components of the vector $e_i^* T_K^{s_K}(i:n,i:n)T_{K-1}^{s_{K-1}}(i:n,i:n)\cdots T_1^{s_1}(i:n,i:n)$ and this requires a solution of a triangular system of equations each time a power $s_j = -1$ is encountered. In this case the bound would become

$$|o_{i,j}| \leq \epsilon c_n \|T_K^{s_K}(i,i:n)\| \cdot \|T_{K-1}^{s_{K-1}}(i:n,i:n)\| \cdots \|T_1^{s_1}(i:n,i:n)\|\kappa,$$

where κ is the product of all condition numbers of the inverted triangular systems (and hence much larger than 1). These are much weaker bounds than asking the off-diagonal elements of A to be ϵ smaller than the ones on the bidiagonal. This would be the case, e.g., if instead we had

$$|o_{i,j}| \leq \epsilon c_n \|T_K^{s_K}(i,i:n)T_{K-1}^{s_{K-1}}(i:n,i:n)\cdots T_1^{s_1}(i:n,i+1:n)\| = \epsilon c_n |e_{i+1}|.$$

Such a bound would guarantee high relative accuracy in the singular values computed from the bidiagonal only [4]. Hence, this is the kind of result one would hope for. These two bounds can in fact be very different when significant cancellations occur between the individual matrices, e.g., if

$$\|A\| << \|A_K^{s_K}\| \cdots \|A_2^{s_2}\| \cdot \|A_1^{s_1}\|.$$

One could observe that the bidiagonalization procedure is in fact a Lanczos procedure [6]. Therefore, there is a tendency to find first the *dominant* directions of the expression $A_K^{s_K} \cdots A_1^{s_1}$ and hence also those directions where there is less cancellation between the different factors. We will see in the examples below that such a phenomenon indeed occurs which is a plausible explanation for the good accuracy obtained. One way to test the performance of the algorithm in cases with very small singular values is to generate powers of a symmetric matrix $A = S^K$. The singular values will be the powers of the absolute values of the eigenvalues of S:

$$\sigma_i(A) = |\lambda_i(S)|^K,$$

and hence will have a large dynamic range. The same should be true for the bidiagonal of A and the size of the $o_{i,j}$ will then become critical for the accuracy of the singular values when computed from the bidiagonal elements $q_i,\ e_i$. This and several other examples are discussed in the next section. There we observe a very high relative accuracy even for the smallest singular values. The only explanation we can give for this is that as the bidiagonalization proceeds, it progressively finds the largest singular values first and creates submatrices that are of smaller norm. These then do not really have cancellation between them, but instead the decreasing size of the bidiagonal elements is the result of decreasing elements in each transformed matrix A_i. In other words, a grading is created in each of the transformed matrices. We believe this could be explained by the fact that the bidiagonalization is a Lanczos procedure and that such grading is often observed there when the matrix has a large dynamic range of eigenvalues. In practice it is of course always possible to evaluate bounds for the elements $|o_{i,j}|$ and thereby obtain estimates of the accuracy of the computed singular values.

The consequence of the above is that the singular values of such sequences can be computed (or better, "estimated") at high relative accuracy from the bidiagonal only! Notice that the bidiagonalization requires 4 to $6Kn^3$ flops but that the subsequent SVD of the bidiagonal is essentially free since it is $\mathrm{O}(n^2)$.

4. Singular vectors and iterative refinement. If one wants the singular vectors as well as the singular values at a guaranteed accuracy, one can start from the bidiagonal B as follows. First compute the bidiagonal,

$$B = Q_K^* A Q_0 = T_K^{s_K} \cdots T_2^{s_2} \cdot T_1^{s_1},$$

and then the SVD of B,

$$B = U \Sigma V^*,$$

where we choose the diagonal elements of Σ to be ordered in decreasing order. We then proceed by propagating the transformation U (or V) and updating each T_i so that they remain upper triangular. Since the neglected elements $o_{i,j}$ were small, the new form

$$\hat{Q}_K^* A \hat{Q}_0 = \hat{T}_K^{s_K} \cdots \hat{T}_2^{s_2} \cdot \hat{T}_1^{s_1}$$

will be upper triangular, and nearly diagonal. This is the ideal situation to apply one sweep of Kogbetliantz's algorithm. Since this algorithm is quadratically convergent when the diagonal is ordered [2], one sweep should be enough to obtain ϵ-small off-diagonal elements.

The complexity of this procedure is as follows. If we use only Givens transformations, we can keep all matrices upper triangular by a subsequent Givens correction. Such a pair takes $6n$ flops per matrix and we need to propagate $n^2/2$ of those. That means $3n^3$ per matrix. The cost of one Kogbetliantz sweep is exactly the same since we propagate the same amount of Givens rotations. We therefore arrive at the following total count for our algorithm:

4 to $6Kn^3$ for triangularizing $A_i \to T_i$,
4 to $5n^3$ for constructing Q_K and Q_0,
$8n^3$ for computing U and V,
$3Kn^3$ for updating $T_i \to \hat{T}_i$,
$3Kn^3$ for one last Kogbetliantz sweep.

The total amount of flops after the bidiagonalization is thus comparable to applying 2 Kogbetliantz sweeps, whereas the Jacobi-like methods typically require 5 to 10 sweeps!

Moreover, our method allows us to select a few singular values and only compute the corresponding singular vectors. The matrices Q_K and Q_0 can be stored in factored form and inverse iteration performed on B to find its selected singular vector pairs and then transformed back to pairs of A using Q_K and Q_0.

5. Numerical examples. Computing the SVD of a general product/quotient of matrices is, as suggested above, a delicate numerical problem. In this section we analyze the accuracy of the QR-like algorithm described in this paper using several examples of varying degree of difficulty. The examples are chosen in order to illustrate the following points already discussed in the paper.

(a) Implicit methods are more reliable than explicit methods. This is of course well known, but we illustrate it with some striking examples.
(b) The bidiagonal computed by the QR-like method yields singular values computed to high relative accuracy even when their dynamical range is very large.
(c) The bidiagonal has a typical "graded" structure when the singular values have a wide dynamical range and its "off-bidiagonal" elements are negligible with respect to the bidiagonal elements in the same row. This is due to its connection to the Lanczos procedure as discussed earlier.
(d) The connection with the Lanczos procedure also allows us to terminate the bidiagonalization early and yet has a good estimate of the dominant singular values.

Points (a)–(d) illustrate the good (relative) accuracy that can be obtained from this procedure even without using iterative refinement based on Kogbetliantz's algorithm. The following points now compare the QR-like and Kogbetliantz approaches.

(e) The bidiagonalization and Kogbetliantz methods have comparable accuracy in "difficult" examples with strong cancellation in the product.
(f) The typical number of Kogbetliantz steps (6 to 10) needed for convergence yields a much slower method than mere bidiagonalization. Moreover, the results are comparable, even when the Kogbetliantz iteration is continued further.
(g) The accuracy obtained from the bidiagonal only is already better on average than that of Kogbetliantz.

Finally, we illustrate that good (relative) accuracy is obtained also for a matrix *quotient.*

(h) The accuracy obtained by bidiagonalization of a matrix quotient is high, even when compared to the accuracy obtained if the inverted factors are explicitly known.

These points illustrate the power of this QR-like method. Note that in all examples we use only the basic part of the algorithm without the iterative refinement step. All calculations were carried out in MATLAB on a Silicon Graphics Indigo workstation with IEEE floating point standard. For computing the singular values of the computed bidiagonal we use the method due to Fernando and Parlett [5].

(a) *Implicit versus explicit.* Let us consider the following products:

$$A_1[n,m] = T_n^m,$$

where T_n is a $n \times n$ symmetric Toeplitz matrix whose first column is $[2, -1, 0, 0, \dots, 0]$ (singular values and singular vectors of such matrices are known [8]). Since it contains only integers we can form these powers of T_n without any rounding errors, which is important for our comparison. The accuracy obtained by computing the SVD of this explicitly formed product is displayed in Figure 1. The interpretation of the figure is

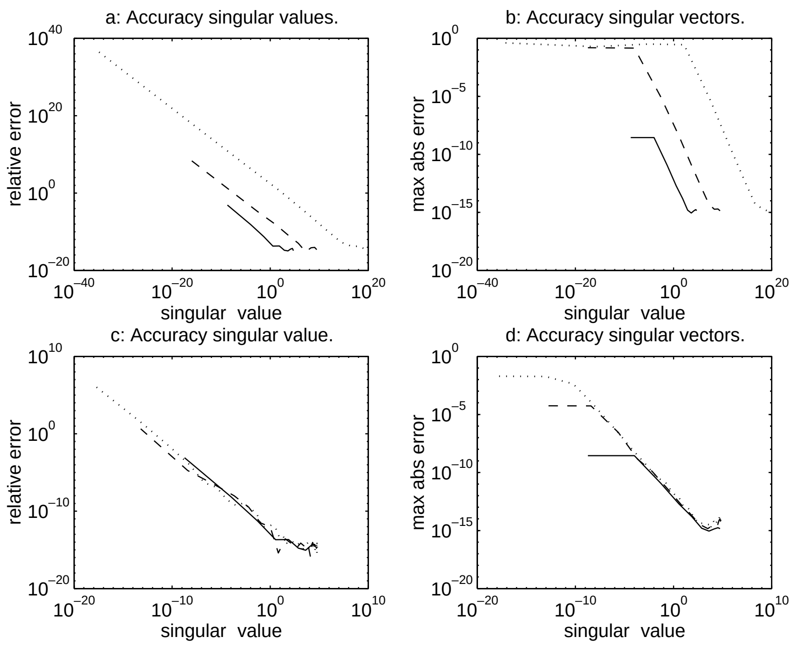

FIG. 1. *The relative accuracy obtained by computing the SVD for the explicitly formed product of Toeplitz matrices. In* (a) *and* (b) *solid, dashed, and dotted lines correspond to* $A_1[n, m]$ *for* $n = 10$ *and* $m \in \{8, 16, 32\}$*; in* (c) *and* (d)*,* $n \in \{10, 20, 40\}$ *and* $m = 8$*. The lines interpolate the relative accuracies for the different singular values.*

as follows. Figures 1(a) and 1(b) correspond to $n = 10$ and $m \in \{8, 16, 32\}$, whereas Figures 1(c) and 1(d) correspond to $n \in \{10, 20, 40\}$ and $m = 8$. The associated results are indicated by the solid, dashed, and dotted lines, respectively. Notice that each line corresponds to a different range of singular values as expected for matrices T_n^m with different values of m and n. In Figures 1(a) and 1(c) we plot the relative accuracy of the singular values as a function of the actual magnitude of the singular value. The lines interpolate the observed relative accuracies, that is, the values $|\sigma_i - \hat{\sigma}_i|/\sigma_i$. In Figures 1(b) and 1(d) we plot the maximum absolute error in the left singular vector elements, that is, $\max_j[|u_{ji} - \hat{u}_{ji}|]$, with u_{ji} being the elements of the ith singular vector, also as a function of the magnitude of the corresponding singular value. From the figure it is clear that the relative accuracy of the computed decomposition is quickly lost as we form powers of the matrix. Moreover, the situation is aggravated as we increase the dimension, and hence the condition number, of the matrix. The explanation lies of course in the fact that roundoff errors in a matrix A are typically proportional to $\epsilon||A||$. For the product $A_1[n, m]$ this tends to have a catastrophic effect on the accuracy of the smallest singular values since they are smaller than $\epsilon||A_1||$.

Let us now use the QR-like SVD algorithm. The result is shown in Figure 2 and illustrates that we have obtained a relative accuracy which is essentially independent of the magnitude of the associated singular value. This shows the need for implicit methods.

(b) *Relative accuracy of implicit methods.* A nonsymmetric example along the same vein is given in Figure 3. The interpretation of the figure is as for Figure 2, but now we consider the product

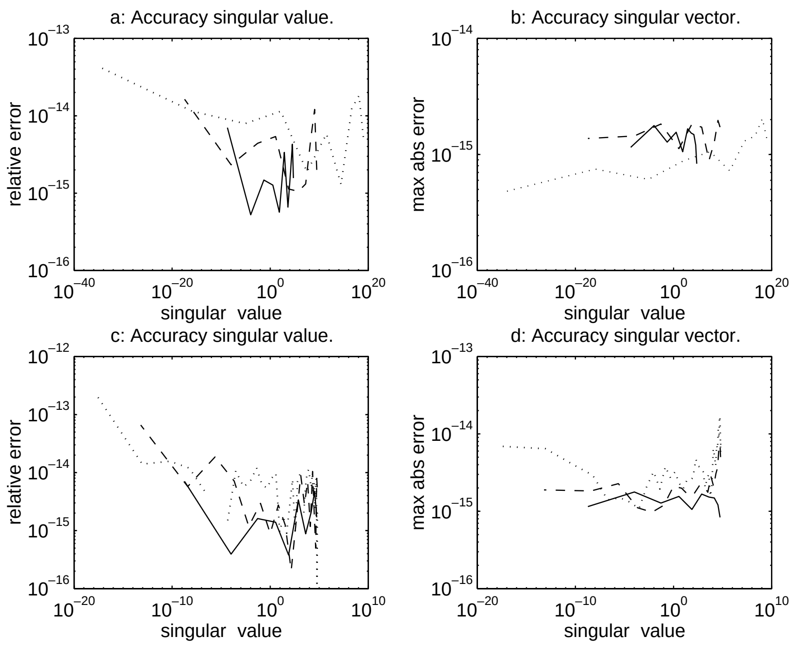

FIG. 2. *Relative accuracy of the SVD estimate obtained by the QR-like algorithm for the product of Toeplitz matrices. In* (a) *and* (b) *solid, dashed, and dotted lines correspond to* $A_1[n,m]$ *for* $n = 10$ *and* $m \in \{8, 16, 32\}$*; in* (c) *and* (d)*,* $n \in \{10, 20, 40\}$ *and* $m = 8$*. Note that also the smallest singular values are computed with high relative accuracy.*

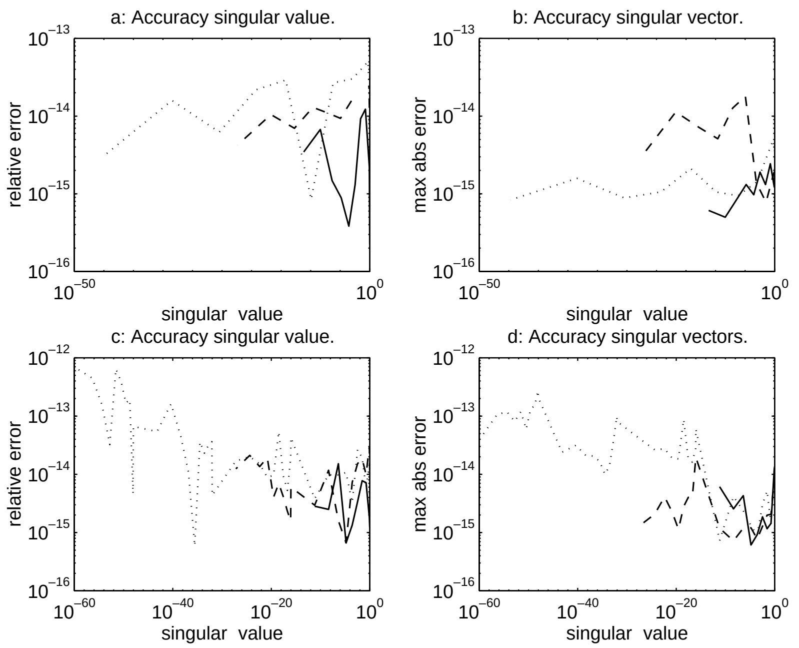

FIG. 3. *Relative accuracy of the SVD estimate obtained by the QR-like algorithm for the product of nonsymmetric matrices. In* (a) *and* (b) *solid, dashed, and dotted lines correspond to* $A_2[n,m]$ *for* $n = 10$ *and* $m \in \{8, 16, 32\}$*; in* (c) *and* (d)*,* $n \in \{10, 20, 40\}$ *and* $m = 8$*.*

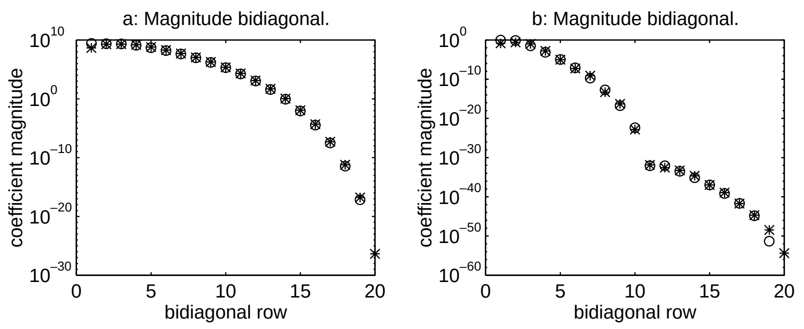

FIG. 4. *Grading in the bidiagonal decomposition computed by the QR-like algorithm. Figure* (a) *corresponds to* $A_1[20,16]$ *and* (b) *to* $A_2[20,16]$. *The* $*$*'s show the magnitude of the diagonal coefficients, the o's the magnitude of the upper bidiagonal coefficients.*

$$A_2[n,m] = (D_n\ D_n^*)^m.$$

Thus, there are $2m$ matrices in the matrix product. The matrix D_n is obtained by explicitly forming

$$D_n \equiv U_n\ \Sigma_n\ V_n^*, \tag{13}$$

where the matrices U_n and V_n are randomly chosen orthogonal matrices. They are defined by the singular vectors of a matrix with independent mean zero unit variance Gaussian entries. Furthermore, Σ_n is the leading part of a $10k \times 10k$ diagonal matrix with diagonal equal to the Kronecker product:

$$[10, 9.9, 9, 8, 7, 6, 5, 4, 3, 2] \otimes [10^{-1}, 10^{-2}, \ldots, 10^{-k}].$$

The motivation for choosing the product in this way is that we obtain an example in which the matrices involved are nonsymmetric and for which we "know" the actual singular values and can examine the obtained relative accuracy. The result is much as above. Using the implicit procedure for computing the singular values returns singular values whose relative accuracies are rather insensitive to the actual magnitude of the corresponding singular value.

(c) *Graded bidiagonal.* That the merits of the algorithm can be understood in terms of the Lanczos connection is confirmed by the next example explained in Figure 4. Here we have plotted the magnitude of the coefficients of the computed bidiagonal for the products $A_1[20,16]$ and $A_2[20,16]$, respectively, in Figures 4(a) and 4(b). The $*$'s show the absolute values of the diagonal coefficients and the o's the absolute values of the upper bidiagonal coefficients in the computed bidiagonal. We see that in both cases a grading has indeed been obtained. The algorithm picks out the dominant directions first, leading to a grading in the computed decomposition. The "effective condition number" of remaining subproblems are therefore successively reduced, and the associated singular values can apparently be obtained with high relative accuracy.

The high accuracy obtained above suggests that the "off-bidiagonal" elements in the transformed product are indeed small relative to the bidiagonal. This is confirmed by the next figure. In Figure 5 we plot, indicated by $*$, the norm of the off-bidiagonal elements normalized by the norm of the bidiagonal elements. That is, after the transformation to upper triangular form we explicitly form the product of the matrices in the product and compute for each row j, $||o_{j,(j+2):n}||/||o_{j,j:(j+1)}||$, with $o_{i,j}$ being the

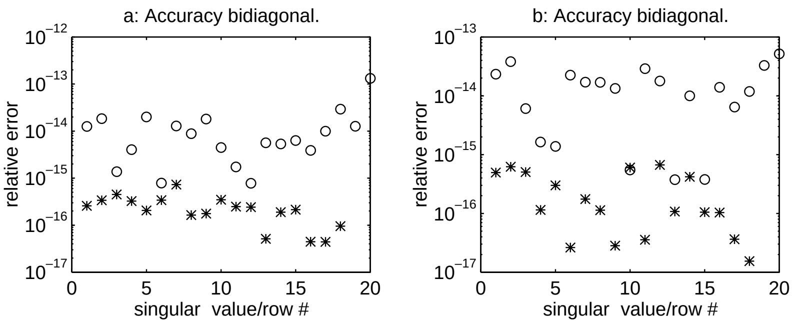

FIG. 5. *Relative accuracies of the bidiagonal elements* ($*$) *and of the computed singular values* (o). *Figure* (a) *corresponds to* $A_1[20,16]$ *and* (b) *to* $A_2[20,16]$.

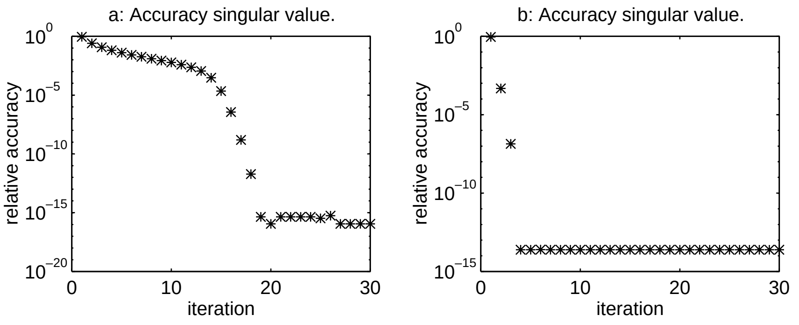

FIG. 6. *Accuracy of estimate of dominant singular value obtained from leading part of computed bidiagonal. The accuracy is plotted as a function of the dimension of the leading submatrix. Figure* (a) *corresponds to* $A_1[80,16]$ *and* (b) *to* $A_2[80,16]$.

elements in the computed product. In exact arithmetic this quantity should be zero. The o's in the figure are the relative accuracies in the computed singular values. Note that the grading and relative smallness of the off-bidiagonal elements make it possible to compute even the smallest singular values with high relative accuracy.

(d) *Dominant singular value.* A consequence of the Lanczos connection is furthermore that we can obtain good estimates for the dominant singular values of the product without computing the full bidiagonal. This is illustrated in Figure 6. Here we plot the estimate of the dominant singular value we obtain by computing the corresponding singular value for the leading parts of the computed bidiagonal, $\hat{B}(1:i,1:i)$. We plot the relative accuracy of this estimate as a function of i in Figures 6(a) and 6(b), corresponding to the products $A_1[80,16]$ and $A_2[80,16]$, respectively. The plots show that we need compute only a part of the bidiagonal in order to obtain a good estimate of the dominant singular value.

(e) *Examples with strong cancellation.* Here we consider examples with a significant cancellation in the product. That is, a subsequence of the matrices in the product is associated with a large dynamic range relative to that of the product whose associated singular values might be only mildly, or not at all, graded. The following example illustrates this:

$$A_3[10,m] = D_{10}^{m}\ D_{10}^{-m},$$
$$A_4[10,m] = (D_{10}\ D_{10}^{-1})^{m},$$

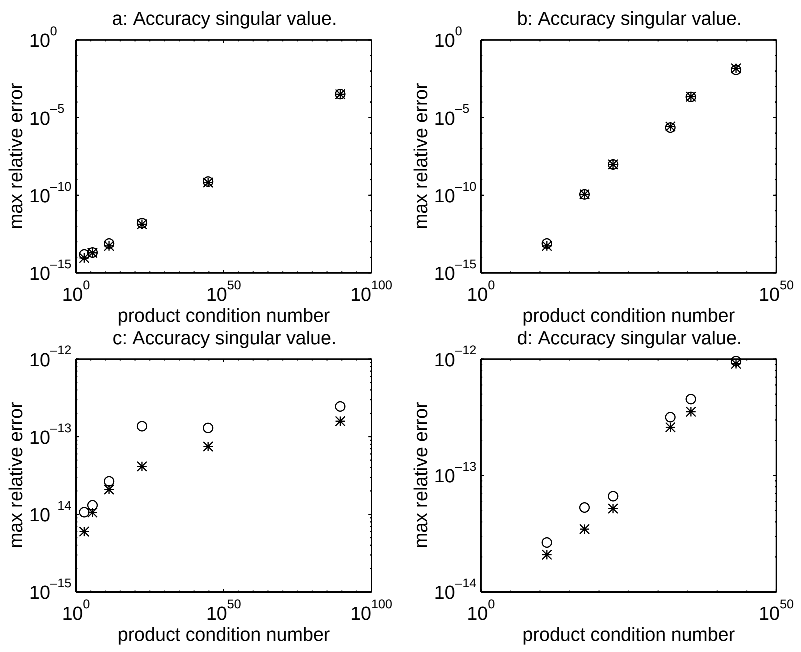

FIG. 7. *The figure compares the accuracy of the computed SVD using, respectively, the QR-like* $(*)$ *and the Kogbetliantz* (o) *algorithms. The considered matrix products exhibit strong cancellation. Figures* (a) *and* (b) *correspond to* $A_3[n,m]$ *and* (c) *and* (d) *to* $A_4[n,m]$*. In* (a) *and* (c), $n = 10$ *and* $m \in \{2,4,8,16,32,64\}$*; in* (b) *and* (d), $n \in \{10,14,18,22,26,30\}$ *and* $m = 8$*. The QR-like algorithm provides accurate singular value estimates at a lower computational cost than the Kogbetliantz algorithm.*

with D_{10} defined as in (13). Note that in this example we compute D_{10}^{-1} explicitly and use the product form of the algorithm. In Figure 7 subplots (a) and (b) correspond to A_3 and (c) and (d) to A_4. For the various product sizes we plot the maximum relative error over the computed singular values. We do so as a function of the "product condition number," defined as the product of the condition numbers of the matrices involved, in this case κ_D^{2m}. In Figures 7(a) and 7(c) we let $n = 10$ and $m \in \{2,4,8,16,32,64\}$, whereas in Figures 7(b) and 7(d) we let $n \in \{10,14,18,22,26,30\}$ and $m = 8$. Note that for both of the above matrix products the product condition number is much larger than its actual condition number. Figures 7(a) and 7(b) show that for the matrix products which are associated with a significant cancellation there is a loss in relative accuracy. The o's in the plot correspond to computing the decomposition by the Kogbetliantz algorithm, fixing the number of sweeps to 12 to avoid issues of convergence tests. Note that the accuracy obtained thereby is not much better than that obtained by the bidiagonalization part of the QR-like algorithm, that is, without iterative refinement.

(f) *Convergence and complexity.* We next turn to the special but important case when $m = 2$ and compare the performance of the algorithm with that of the Kogbetliantz algorithm. In Figures 8(a) and 8(b) we consider the product $A_2[40,1]$. The dashed lines correspond to the relative accuracy obtained by $2, 4, 6$, and 10 sweeps of the Kogbetliantz algorithm. The relatively slow convergence of some singular values corresponds to those being closely spaced. Note that even 10 sweeps of Kogbetliantz's algorithm do not return an approximation with accuracy beyond that obtained by the

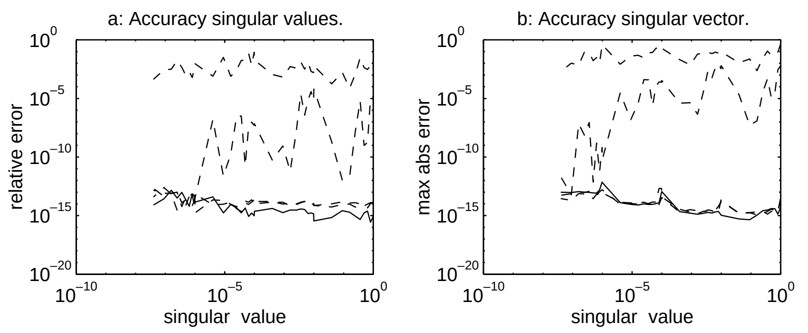

FIG. 8. *Convergence of Kogbetliantz algorithm when computing the SVD of the pair of matrices defined by $A_2[40,1]$. The accuracies after* 2, 4, 6, *and* 10 *sweeps are shown. The bottom solid line shows the accuracy obtained with the QR-like algorithm without iterative refinement.*

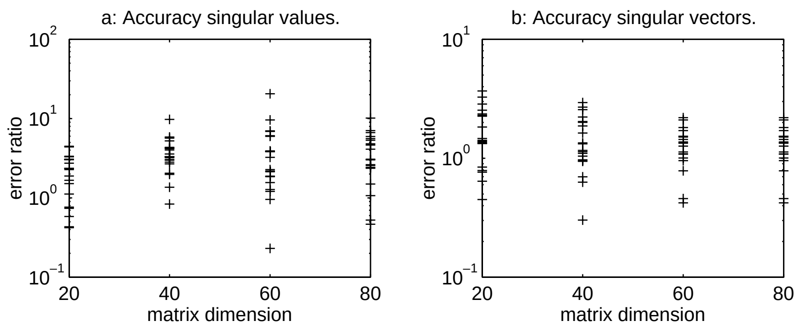

FIG. 9. *Comparison of accuracy of the computed SVD obtained, respectively, by the QR-like and the Kogbetliantz algorithms for the square of a collection of random matrices. For each matrix realization the cross is the maximum relative error with the Kogbetliantz algorithm over the maximum relative error for the QR-like algorithm.*

QR-like SVD algorithm without iterative refinement as shown by the solid line.

(g) *Comparison of accuracy.* The two plots in Figure 9 are obtained as follows. We consider the products defined by

$$A_5[n,2] = N_n\ N_n^*$$

with N_n being an $n \times n$ random matrix whose coefficients are normally distributed and with $n \in \{20, 40, 60, 80\}$. The SVD was first computed (via MATLAB) and we then reconstructed N_n from this SVD. Hence it is reasonable to assume that the singular values of N_n and $A_5[n,2]$ are known "exactly." Let $\hat{\sigma}_i$ and $\tilde{\sigma}_i$ represent the estimates of the singular values associated with, respectively, the QR-like and the Kogbetliantz algorithms. In the latter case we used 10 sweeps, whereas in the former we did not include iterative refinement. Similarly let $\hat{u}_{ij}$ and $\tilde{u}_{ij}$ represent the coefficients in the left singular vectors. We then plot the ratio $\max_i[|\sigma_i - \tilde{\sigma}_i|/\sigma_i]/\max_i[|\sigma_i - \hat{\sigma}_i|/\sigma_i]$ in Figure 9(a) and the ratio $\max_{ij}[|u_{ij} - \tilde{u}_{ij}|]/\max_{ij}[|u_{ij} - \hat{u}_{ij}|]$ in Figure 9(b). The +'s correspond to different realizations of the matrix N_n. We see that the QR-like algorithm typically yields a more accurate approximation despite its lower computational cost.

(h) *Example with matrix quotients.* In this last example we compute the SVD of matrix quotients involving inverted matrices. As described above, we then have

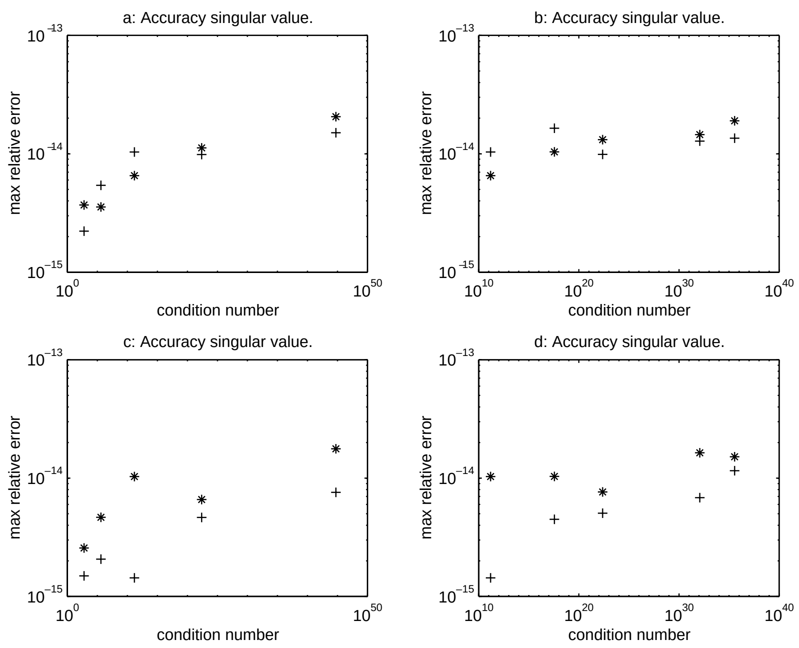

FIG. 10. *The figure compares the accuracy of the computed SVD when the QR-like algorithm based on, respectively, the quotient representation* (+) *and the product representation* ($*$) *of the matrix is being used. Figures* (a) *and* (b) *correspond to* $A_6[n,m]$ *and* (c) *and* (d) *to* $A_7[n,m]$*. In* (a) *and* (c)*,* $n = 10$ *and* $m \in \{2,4,8,16,32\}$ *and in* (b) *and* (d)*,* $n \in \{10,14,18,22,26\}$ *and* $m = 8$*. Note that the accuracy obtained from the quotient representation is similar to the accuracy obtained from the product representation.*

to carry out an initial step in the bidiagonalization where the QR factorizations of the inverted matrices are computed. We construct the matrix quotients such that we explicitly can compute the inverses and compare the accuracy of the product version of the algorithm, based on these explicitly computed inverses, with the quotient version.

First, define the matrix quotient A_6:

$$A_6[n,m] = A_1^{-1}A_2^{-1}\cdots A_m^{-1}A_{m+1}A_{m+2}\cdots A_{2m},$$

where

$$\begin{aligned} A_i &= Q_n^{(i)}\Sigma_n^{-1}Q_n^{(i-1)*}, \\ A_{m+i} &= Q_n^{(m+i-1)}\Sigma_n Q_n^{(m+i)*} \end{aligned}$$

for $1 \le i \le m$ and with Σ_n defined as in (13). Moreover, the $Q_n^{(i)}$ are independent random orthogonal $n \times n$ matrices. As above these are defined by the singular vectors of a matrix with independent mean zero unit variance Gaussian entries.

Second, define the matrix quotient A_7:

$$A_7[n,m] = A_1^{-1}A_2A_3^{-1}\cdots A_{2m-2}A_{2m-1}^{-1}A_{2m}$$

with

$$\begin{aligned} A_{2i-1} &= Q_n^{(2i-1)}\Sigma_n^{-1}Q_n^{(2i-2)*}, \\ A_{2i} &= Q_n^{(2i-1)}\Sigma_n Q_n^{(2i)*} \end{aligned}$$

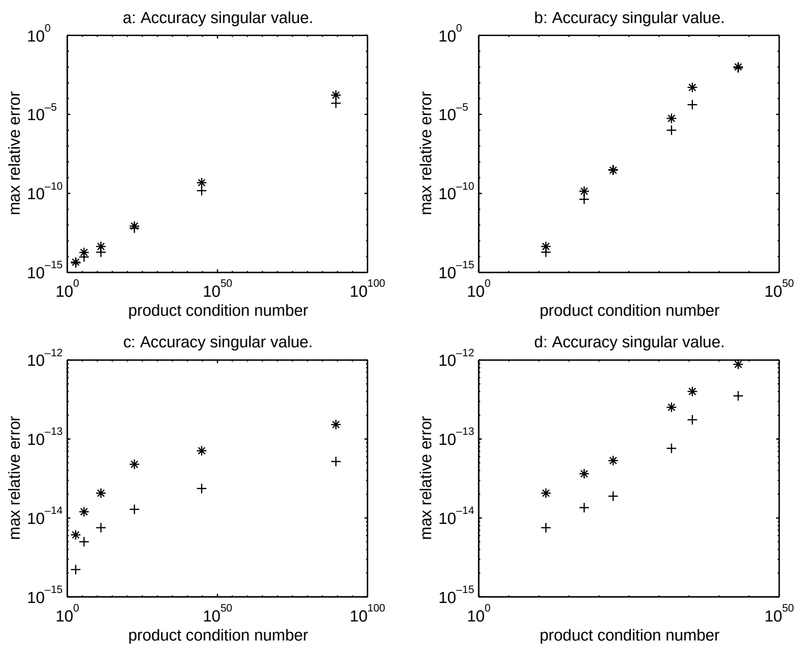

FIG. 11. *The figure compares the accuracy of the computed SVD when the QR-like algorithm based on, respectively, the quotient representation* $(+)$ *and the product representation* $(*)$ *of the matrix is being used. The considered matrix quotients exhibit strong cancellation. Figures* (a) *and* (b) *correspond to* $A_3[n,m]$ *and* (c) *and* (d) *to* $A_4[n,m]$. *In* (a) *and* (c), $n = 10$ *and* $m \in \{2,4,8,16,32,64\}$ *and in* (b) *and* (d), $n \in \{10,14,18,22,26,30\}$ *and* $m = 8$.

for $1 \le i \le m$ and with Σ_n and $Q_n^{(i)}$ defined as above. Note that these quotients are associated with a large dynamic range.

The resulting relative accuracy obtained when varying the matrix dimension and the number of matrices in the quotient is shown in Figure 10. Figures 10(a) and 10(b) correspond to the quotient A_6 and Figures 10(c) and 10(d) correspond to the quotient A_7. In Figures 10(a) and 10(c), $n = 10$ and $m \in \{2,4,8,16,32\}$, whereas in Figures 10(b) and 10(d), $n \in \{10,14,18,22,26\}$ and $m = 8$. The $+$'s show the accuracy obtained with the QR-like algorithm based on the *quotient* and *without* iterative refinement. The $*$'s show the accuracy obtained with the product form of the QR-like algorithm *without* iterative refinement; note that in this case the inverses are explicitly computed. We see that the relative accuracy obtained when we do not assume knowledge of the inverses is comparable to, or even somewhat better than, the accuracy obtained if these are known!

Finally, reconsider the matrices of example (e) that exhibit strong cancellation. We compute as above the decomposition based on both the product form and the quotient form. The result is shown in Figure 11. Figures 11(a) and 11(b) correspond to the quotient A_3 and Figures 11(c) and 11(d) correspond to the quotient A_4. In Figures 11(a) and 11(c), $n = 10$ and $m \in \{2,4,8,16,32,64\}$, whereas in Figures 11(b) and 11(d), $n \in \{10,14,18,22,26,30\}$ and $m = 8$. The figure shows that computation based on the quotient form gives a relative accuracy that is in general somewhat better than the accuracy based on the product form, at the cost of a slightly higher flop count.

6. Concluding remarks. The algorithm presented in this paper nicely complements the unitary decompositions for sequences of matrices defined for the generalized QR [3] and Schur decompositions [1]. These decompositions find applications in sequences of matrices defined from discretizations of ordinary differential equations occurring, for instance, in 2-point boundary value problems [9] or control problems [1]. We expect that they will lead to powerful tools for analyzing as well as solving problems in these application areas.

We want to stress here that in all examples it turned out to be sufficient to compute the bidiagonal B of the expression $A^{s_K} \cdots A^{s_1}$ and then the singular values of B, without any further iterative refinement. This is rather surprising. The bounds obtained on the accuracy of the bidiagonal are much worse than what was observed in the examples. This point and the connection to the Lanczos process need further analysis. That we get accurate approximations for the leading order bidiagonals might be useful when solving ill-posed or inverse problems.

The main advantage of the new method lies exactly in the fact that this bidiagonal is so accurate. If no iterative refinement is needed, then the method requires 5 to 10 times less flops than Kogbetliantz! If iterative refinement is needed, then the method should still be superior since the work then amounts essentially to the work of two Kogbetliantz steps.

Finally, we point out that there is recent work on computing singular values to high relative accuracy via the Kogbetliantz algorithm. This work is based on extracting particular scalings from the factors. So far this has been applied to problems involving three factors only. Extensions to several matrices and whether these methods perhaps could be combined with the bidiagonal approach in an advantageous way are still open problems. Those methods and the ideas developed in this paper are, we believe, related. In both methods grading in the factors, obtained either explicitly or implicitly, is important.

REFERENCES

[1] A. Bojanczyk, G. Golub, and P. Van Dooren, *The periodic Schur form. Algorithms and applications*, in Proceedings of the SPIE Conference, San Diego, CA, 1992, pp. 31–42.

[2] J. P. Charlier and P. Van Dooren, *On Kogbetliantz's SVD algorithm in the presence of clusters*, Linear Algebra Appl., 95 (1987), pp. 135–160.

[3] B. De Moor and P. Van Dooren, *Generalizations of the singular value and QR decomposition*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 993–1014.

[4] J. Demmel, M. Gu, S. Eisenstat, I. Slapnicar, and K. Veselic, *Computing the singular value decomposition with high relative accuracy*, Linear Algebra Appl., submitted.

[5] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229.

[6] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205–224.

[7] G. Golub and C. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[8] R. Gregory and D. Karney, *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, New York, 1969.

[9] R. M. Mattheij and S. J. Wright, *Parallel stabilized compactification for ODEs with parameters and multipoint conditions*, Appl. Numer. Math., 13 (1993), pp. 305–333.

[10] C. B. Moler and G. W. Stewart, *An algorithm for the generalized matrix eigenvalue problem*, SIAM J. Numer. Anal., 10 (1973), pp. 241–256.

[11] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.