A QR-LIKE SVD ALGORITHM FOR A PRODUCT/QUOTIENT OF SEVERAL MATRICES

GENE GOLUB
Computer Science Department
Stanford University
Stanford, CA USA
golub@sccm.stanford.edu

KNUT SOLNA
Computer Science Department
Stanford University
Stanford, CA USA
solna@sccm.stanford.edu

PAUL VAN DOOREN
Cesame
Université Catholique de Louvain
Louvain-la-Neuve
Belgium
vandooren@anma.ucl.ac.be

ABSTRACT. In this paper we derive a new algorithm for constructing 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 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.

KEYWORDS. Numerical methods, generalized singular values, products of matrices, quotients of matrices.

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 $A = U\Sigma V^*$ – where U and V are unitary and Σ is diagonal – (for the latter A does not need to be square). It is interesting to notice that both these forms are computed by a QR-like iteration [4]. The SVD algorithm of Golub-Kahan [3] 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 [6] 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.

This is not the case, though, when considering the generalized singular value decomposition of two matrices, appearing as a quotient $B^{-1}A$ or a product BA. In this case the currently used algorithm is *not* of QR type but of a Jacobi type. The reason for this choice is that Jacobi methods extend to products and quotient without too much problems. The bad news is that 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.

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 singular value decomposition

Consider the problem of computing the singular value decomposition of a matrix A that is an expression of the following type :

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

where $s_i = \pm 1$, i.e. a sequence of products of quotients of matrices. For simplicity we assume that the A_i matrices are square $n \times n$ and invertible, but as was pointed out in [2], this does not affect the generality of what follows. While it is clear that one has to perform left and right transformations on A to get $U^*AV = \Sigma$, these transformations will only affect A_K and A_1. Yet, one can insert an expression $Q_i^* Q_i = I_n$ in 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 :

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

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. Notice 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 :

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

From the construction of a normal QR decompostion, 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 matriz A as well. Since meanwhile we computed the singular values of (1), it is clear that such a result can only be obtained by an *iterative procedure*. On the other hand, one intermediate form that is used in the Golub-Kahan SVD algorithm [3] 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 only have a product of matrices

A_i and in order to illustrate the procedure we show its evolution operating on a product of 3 matrices only, i.e. $A_3A_2A_1$. Below is a sequence of "snapshots" of the evolution of the bidiagonal reduction. Each snapshot 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 :

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

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

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

At this stage we are interested in the *first row* of this product (indicated in boldface 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 $O(Kn^2)$ 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 :

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

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

$$Q_K^{(1)*}AQ_0^{(1)} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{bmatrix}.$$

Now perform a Householder transformation $Q_1^{(2)} \in \mathcal{H}(2,n)$ on the rows of A_1 and the columns of A_2. Choose $Q_1^{(2)}$ to annihilate all but two elements in the second column of A_1

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

Then perform a Householder transformation $Q_2^{(2)} \in \mathcal{H}(2,n)$ on the rows of A_2 and the columns of A_3. Choose $Q_2^{(2)}$ to annihilate all but two elements in the second column of A_2 :

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

Then perform a Householder transformation $Q_3^{(2)} \in \mathcal{H}(2,n)$ on the rows of A_3 and choose it to annihilate all but two elements in the second column of A_3 :

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

For the product we know that :

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

At this stage we are interested in the *second row* of this product (indicated in boldface above). This row can be constructed as the product of the second row of A_3 with the matrices to the right of it, and this again requires only $\mathrm{O}(K(n-1)^2)$ flops. Once constructed we can find a Householder transformation $Q_0^{(2)} \in \mathcal{H}(3,n)$ operating on the last $(n-2)$ elements which annihilates all but two elements :

$$A_3(2,:)A_2A_1Q_0^{(2)} = \begin{bmatrix} 0 & x & x & 0 & 0 \end{bmatrix}. \tag{5}$$

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

$$Q_K^{(2)*}Q_K^{(1)*}AQ_0^{(1)}Q_0^{(2)} = \begin{bmatrix} x & x & 0 & 0 & 0 \\ 0 & x & x & 0 & 0 \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \end{bmatrix}.$$

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

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

Notice that we never construct the whole product $A = A_3A_2A_1$, 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 post multiplied with essentially n Householder transformations of decreasing range. For updating all A_i we therefore need

$5Kn^3/3$ flops, and for updating Q_K and Q_0 we need $2n^3$ flops. For constructing the required row vectors of A we need $(K-1)n^3/3$ flops. Overall we thus need $2Kn^3$ flops for the construction of the triangular T_i and $2n^3$ for the outer transformations Q_K and Q_0. Essentially this is $2n^3$ flops per updated matrix.

If we now have some of the $s_i = -1$ we can not use Householder transformations anymore. Indeed in order to construct the rows of A when needed, the matrices A_i^{-1} have to be triangularized first, say with a QR factorization. The QR factorization is performed in an initial step. From there on the same procedure is followed, but using Givens rotations instead of Householder transformations. The use of Givens rotations allow us to update the triangularized matrices A_i^{-1} while keeping them upper triangular. Each time a Givens rotation detroys this triangular form, another Givens rotation is applied to the other side of that matrix in order to restore its triangular form. The same technique is e.g. used in keeping the B matrix upper triangular in the QZ algorithm applied to $B^{-1}A$. The bookkeeping of this algorithm is a little more involved and so are the operation counts, which is why we do not develop this here. One shows that when there are inverses involved, the complexity of the bidiagonalization step amounts to less than $4n^3$ flops per updated matrix.

3 Computing the singular values

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

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

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 [7], [4]).

Things are different with the elements of A since they are not stored 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 :

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

The diagonal elements q_i are indeed just a product of the corresponding diagonal elements of the T_j matrices, possibly inverted :

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

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 superposed on the diagonal elements $t_{j_{i,i}}$ of the matrices T_j. For the errors performed when computing the e_i one needs a more detailed analysis. We show below that the backward errors can be superposed 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
$q_i := q;\ e_i := e;$

Notice that for $i = 1$ the same recurrence holds without the expressions involving e since e_1 does not exist. From these recurrences it is clear that the calculation of q_i involves 1 flop per step j and hence a total of K rounding errors which can be superposed on the diagonal elements $t_{j_{i,i}}$:

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

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

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

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

$$
\begin{aligned}
e_i &= \overline{t}_{4_{i-1,i}}.\overline{t}_{3_{i,i}}.\cdots.\overline{t}_{2_{i,i}}.\overline{t}_{1_{i,i}}+ \\
&\quad \overline{t}_{4_{i-1,i-1}}.\overline{t}_{3_{i-1,i}}.\cdots.\overline{t}_{2_{i,i}}.\overline{t}_{1_{i,i}}+ \\
&\quad \vdots \\
&\quad \overline{t}_{4_{i-1,i-1}}.\overline{t}_{3_{i-1,i-1}}.\cdots.\overline{t}_{2_{i-1,i}}.\overline{t}_{1_{i,i}}+ \\
&\quad \overline{t}_{4_{i-1,i-1}}.\overline{t}_{3_{i-1,i-1}}.\cdots.\overline{t}_{2_{i-1,i-1}}.\overline{t}_{1_{i-1,i}}
\end{aligned} \tag{12}
$$

where $(K-1)$ additional rounding are induced for each factor. Therefore we have $\overline{t}_{j_{i-1,i}} = t_{j_{i-1,i}}(1+\eta_{i,j}),\ \ |\eta_{i,j}| < 4(K-1)\epsilon/(1-4(K-1)\epsilon)$. When some of the $s_j = -1$ the above expression is still 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 j-th term if $s_j = -1$. The obtained bound is then $|\eta_{i,j}| < 4(K+1)\epsilon/(1-4(K+1)\epsilon)$. 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 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, who in turn satisfy the bounds (6,7). Unfortunately, nothing of the kind can be quaranteed for the elements

$o_{i,j}$ in (8), who are supposed to be zero in exact arithmetic. The best bound we can obtain from the construction of the rows (4,5) is that :

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

which is a much weaker bound than asking the off diagonal elements of A to be ϵ smaller that the ones on the bidiagonal. This would be the case e.g. if instead we had :

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

Yet, this is the kind of result one would hope for if some singular values of A are small and still have to be computed to a high relative accuracy. These two bounds can in fact be very different when significant cancellations occur between the individual matrices, e.g. if

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

One could observe that the bidiagonalization procedure is in fact a Lanczos procedure [3]. Therefore there is a tendency to find first the *dominant* directions of the expression $A_K^{s_K} \cdot \ldots \cdot A_1^{s_1}$ and hence also those directions where there is less cancellations 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 fromnm this decomposition. In practice it is ofcourse always possible to evaluate bounds for the elements $|o_{i,j}|$ and thereby yield estimates of the accuracy of the computed singular values.

One way to test the performance of this 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. We ran several tests with matrices S of which we know the exact eigenvalues and observed 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.

The consequence of all this 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 $2Kn^3$ flops but that the subsequent SVD of the bidiagonal is essentially free since it is $O(n^2)$.

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 :

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

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 :

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

will be upper triangular, and nearly diagonal. This is the ideal situation to apply one sweep of Kogbetliantz's algorithm. Since this algorithm is qudratically convergent when the diagonal is ordered [5], 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 $4n$ flops per matrix and we need to propagate $n^2/2$ of those. That means $2n^3$ per matrix. The cost of one Kogbetliantz sweep is exactly the same since we propagate the same amount

of Givens rotations. We arrive thus at the following total count for our algorithm :

$2Kn^3$ for triangularizing $A_i \rightarrow T_i$
$2n^3$ for constructing Q_K and Q_0
$8n^3$ for computing U and V
$2Kn^3$ for updating $T_i \rightarrow \hat{T}_i$
$2Kn^3$ for one last Kogbetliantz sweep.

The total amount of flops after the bidiagonalization is thus comparable to 2 Kogbetliantz sweeps, whereas the Jacobi like methods typically require 5 to 10 of those sweeps ! Moreover, this method allows to select a few singular values and only compute the corresponding singular vectors. The matrices Q_K and Q_0 can e.g. be stored in factored form and inverse iteration can be 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 Concluding remarks

The algorithm presented in this paper nicely complements the unitary decompositions for sequences of matrices defined for the generalized QR [2] and Schur decompositions [1]. These decompositions find applications in sequences of matrices defined from discretizations of ordinary differential equations occurring e.g. in two point boundary value problems [8] or control problems [1]. We expect they will lead to powerful tools for analyzing as well as solving problems in these application areas.

Acknowledgements

G. Golub was partially supported by the National Science Foundation under Grants DMS-9105192 and DMS-9403899. K. Solna was partially supported by the Norwegian Council for Research. P. Van Dooren was partially supported by the National Science Foundation under Grant CCR-9209349.

References

[1] A. Bojanczyk, G. Golub and P. Van Dooren, The periodic Schur form. Algorithms and Applications, Proceedings SPIE Conference, pp. 31-42, San Diego, July 1992.

[2] B. De Moor and P. Van Dooren, Generalizations of the singular value and QR decomposition, *SIAM Matr. Anal. & Applic.* **13**, pp 993-1014, 1992.

[3] G. Golub and V. Kahan, Calculating the singular values and pseudo-inverse of a matrix. *SIAM Numer. Anal.* **2**, pp 205-224, 1965.

[4] G. Golub and C. Van Loan, *Matrix Computations* 2nd edition, The Johns Hopkins University Press, Baltimore, Maryland, 1989.

[5] J.P. Charlier and P. Van Dooren, On Kogbetliantz's SVD algorithm in the presence of clusters. *Linear Algebra & Applications* **95**, pp 135-160, 1987.

[6] C. Moler and G. Stewart, An algorithm for the generalized matrix eigenvalue problem, *SIAM Numer. Anal.* **10**, pp 241-256, 1973.

[7] J. H. Wilkinson, *The algebraic eigenvalue problem*, Clarendon press, Oxford, 1965.

[8] R. Mattheij and S. Wright, Parallel stable compactification for ODE with parameters and multipoint conditions, Int. Rept. Argonne Nat. Lab., IL, 1994.