Electronic Transactions on Numerical Analysis.
Volume 5, pp. 29-47, June 1997.
Copyright © 1997, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
etna@mcs.kent.edu

A NEW ALGORITHM FOR THE SVD OF A LONG PRODUCT OF MATRICES AND THE STABILITY OF PRODUCTS[*]

DAVID E. STEWART[†]

Key words. SVD, products of matrices, Lyapunov exponents.

AMS subject classifications. 65F15, 34D08.

Abstract. Lyapunov exponents can be estimated by accurately computing the singular values of long products of matrices, with perhaps 1000 or more factor matrices. These products have extremely large ratios between the largest and smallest eigenvalues. A variant of Rutishauser's Cholesky LR algorithm for computing eigenvalues of symmetric matrices is used to obtain a new algorithm for computing the singular values and vectors of long products of matrices with small backward error in the factor matrices. The basic product SVD algorithm can also be accelerated using hyperbolic Givens' rotations. The method is competitive with Jacobi-based methods for certain problems as numerical results indicate.

Some properties of the product SVD factorization are also discussed, including uniqueness and stability. The concept of a *stable product* is introduced; for such products, all singular values can be computed to high relative accuracy.

1. Introduction.

Algorithms have been developed to compute the singular values of products of matrices without explicitly computing the products. Some of these algorithms can only computed the SVD of a product of two matrices, such as the algorithms in Heath, Laub, Paige and Ward [14], Fernando and Hammarling [11], or for a product of three matrices (the HK SVD) by Ewerbring and Luk [9], [10]. Algorithms for computing the SVD of longer products are discussed in Bojanczyk, Ewerbring, Luk and Van Dooren [3], and Abarbanel, Brown and Kennel [1]. All of the above methods, except the last, use Jacobi methods for computing the SVD.

Another approach is to use product versions of rank-revealing factorizations. This is pursued by G.W. Stewart in [23], where a generalization of QR factorizations with column pivoting is developed.

The above algorithms use only orthogonal transformations. In contrast, there have also been some other forms of product SVD's (in fact, a whole tree of such decompositions) described by De Moor [4], H. Zha [25], and De Moor and Zha [5]. For each factor in the product SVD's in De Moor's and Zha's family of decompositions, they assign a "P" or a "Q" depending on whether the product involves the matrix as a factor directly, or the inverse adjoint of the matrix as a factor. That is, while a PP-SVD for factors A and B gives the SVD of AB, the PQ-SVD gives the SVD of $A(B^{-1})^*$.

These decompositions of De Moor and Zha use non-orthogonal transformations in the same spirit as the generalized SVD as presented in Golub and Van Loan [13], §8.7.3. However, here, as in the paper by Bojanczyk *et al.*, the product SVD is a factorization of $A = A^{(1)} \cdot A^{(2)} \cdots A^{(p)}$ of the form

$$A = U^{(1)} R^{(1)} \cdots R^{(p)} (U^{(p+1)})^T,$$

where $R^{(k)} = (U^{(k)})^T A^{(k)} U^{(k+1)}$ is upper triangular, and the matrices $U^{(k)}$ are orthogonal, and the product $R^{(1)} \cdots R^{(p)}$ is diagonal.

* Received September 27, 1996. Accepted for publication June 20, 1997. Communicated by L. Reichel.

† Mathematics Department, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0123, USA. Email: `stewart@math.vt.edu`. This work was supported in part through CODAC by AFOSR grant F49620-93-1-0280.

Here the emphasis is on obtaining backward error analysis in terms of the *factors* of the product rather than the product itself. As has often been noted, backward error analysis gives the best possible result for any numerical computation, since the data is invariably corrupted by noise with magnitude of the order of machine epsilon or unit roundoff. Also, the data for the product SVD problem consists of the sequence of factor matrices, not the product itself.

An aim of this work is to accurately compute Lyapunov exponents. To do this directly in terms of singular value computations requires high *relative* accuracy rather than high *absolute* accuracy. This is only sometimes possible, as a counterexample given in this paper shows. In this paper we introduce the concept of a *stable product*, for which singular values can be computed to high relative accuracy.

One application of these algorithms is to accurately compute Lyapunov exponents for dynamical systems [2]. This involves computing the SVD of a matrix defined either through differential equations

$$\Phi'(t) = A(t)\Phi(t), \qquad \Phi(0) = I, \tag{1.1}$$

or through difference equations

$$\Phi_{k+1} = A_k \Phi_k, \qquad \Phi_0 = I. \tag{1.2}$$

The ith Lyapunov exponent is related to the ith singular values of $\Phi(t)$ or Φ_k by the relation

$$\lambda_i = \lim_{t\to\infty} \frac{\log(\sigma_i(\Phi(t)))}{t} \quad \text{or } \lambda_i = \lim_{k\to\infty} \frac{\log(\sigma_i(\Phi_k))}{k}, \tag{1.3}$$

depending on whether the system is a differential or difference equation. Unfortunately from the numerical point of view, these matrices become increasingly ill-conditioned; distinct Lyapunov exponents imply exponentially diverging singular values. High *relative* accuracy of the singular values rather than high absolute accuracy is desired for obtaining good absolute accuracy in the estimates for the λ_i, even if the ratio of largest to smallest singular values is much greater than $1/\mathbf{u}$ where $\mathbf{u}$ is the unit roundoff. High accuracy relative to $||\Phi_k||$ is not sufficient as the smaller singular values can be easily swamped by noise from the larger singular values. With exponentially large and exponentially small singular values, there are also dangers of over- and under-flow. For example, in Lyapunov exponent calculations, the singular values of products of hundreds or thousands of matrices are wanted, and the singular values can easily exceed the range $10^{\pm 1000}$.

While there are already methods to estimate the Lyapunov exponents of a dynamical system by forming the QR factorization of a product of matrices, with a modest multiple of the work needed to compute this product QR factorization, the entire singular value decomposition can be computed. As the singular values are more directly related to the definition of the Lyapunov exponents, and the QR factorization only gives results that are *asymptotically* correct, it might be expected that this will give more quickly converging estimates of these exponents.

The aim is, therefore, to compute the singular values of a product of matrices

$$A = A^{(1)} \cdot A^{(2)} \cdots A^{(p)} \tag{1.4}$$

while avoiding forming the product (at least in the conventional way), or making the $A^{(k)}$ matrices ill-conditioned. The algorithm developed here is based on the LRCH algorithm of Rutishauser and Schwartz [21] for finding the eigenvalues of $A^T A$ by using *treppeniteration*, without forming either A or $A^T A$. This method is guaranteed to converge [21], but may do so rather slowly if some eigenvalues are close. In practice, for problems of the sort that come from dynamical systems, convergence is usually very rapid. Where convergence is slow, a shifting method based on hyperbolic rotations can accelerate convergence, as is described in section 8.

This approach based on the LRCH algorithm is in fact equivalent in exact arithmetic to implementing the QR algorithm for the product $A^T A$ as implemented by Abarbanel, Brown and Kennel [1], by a result of Wilkinson [24, p. 535]. However, Abarbanel, Brown and Kennel [1] do not perform any error analysis for their method under roundoff error, and it does not appear possible to perform backward error analysis on their method. Backward error results are proven in this paper for the LRCH method developed here.

Most previous product SVD algorithms, such as in [3], [8], and [14], are essentially Jacobi methods.

In this paper only real matrices are considered. The complex case is an easy extension of the real case.

The remainder of the paper is divided as follows. In section 2 the LRCH algorithm as applied to computing the singular values of a product of matrices is described; section 3 describes the convergence and deflation theory for the product LRCH algorithm; section 4 gives the backward error analysis for the algorithm, with some discussion of the stability properties of the problem; section 5 describes how non-square factors can be made square in the algorithm (which simplifies the analysis for testing rank deficiency); section 6 discusses the stability of the product SVD problem, and described *stable products*; section 7 discusses numerical rank deficiency in the context of *products* of matrices; section 8 describes the use of hyperbolic rotations to accelerate the convergence of the main algorithm; and section 9 gives some numerical results to indicate the competitiveness of the algorithm, at least for long products.

2. The LRCH algorithm for computing singular values of products.

The LRCH algorithm for computing the singular value decomposition of a long product of matrices is

```
(2.1)
$U_1 \leftarrow I$; $V_1 \leftarrow I$
for $i = 1, 2, \ldots$
    $Q_i^{(p+1)} \leftarrow I$
    for $k = p, p-1, \ldots, 1$
        $C_i^{(k)} \leftarrow A_i^{(k)} Q_i^{(p+1)}$
        $C_i^{(k)} = Q_i^{(k)} R_i^{(k)}$        (QR factorization)
    if $i$ odd then
        $U_{i+1} \leftarrow U_i Q_i^{(1)}$
    else
        $V_{i+1} \leftarrow V_i Q_i^{(1)}$
    for $k = 1, 2, \ldots, p$
        $A_{i+1}^{(k)} \leftarrow (R_i^{(p-k+1)})^T$
```

Note that i is the iteration counter.

This algorithm is based on Rutishauser's LRCH algorithm [21] for computing the eigenvalues of a symmetric positive semi-definite matrix. The basic LRCH algorithm without shifts is [21]:

```
(2.2)
for $i = 1, 2, \ldots$
    $B_i = L_i L_i^T$        (Cholesky factorization)
    $B_{i+1} \leftarrow L_i^T L_i$.
```

Computing the singular values of A is equivalent to computing the square roots of the eigenvalues of the matrix $B = A^T A$. Note that forming the Cholesky factorization of $B_i = A_i^T A_i = L_i L_i^T$ is equivalent to forming the QR factorization of A_i: $A_i = Q_i R_i$ implies that $R_i = D_i L_i^T$ where D_i is a diagonal matrix with ± 1's on the diagonal. The basic LRCH iteration for computing singular values is thus

(2.3)
```
for $i = 1, 2, \ldots$
    $A_i = Q_i R_i$    (QR factorization)
    $A_{i+1} \leftarrow R_i^T$.
```

(The updates for U_i and V_i have been dropped here.) It should also be noted that if A_i is rank deficient, there is still a factorization $A_i^T A_i = L_i L_i^T$, though this may not be computable by the standard Cholesky algorithm.

The QR factorization of $A_i = A_i^{(1)} \cdot A_i^{(2)} \cdots A_i^{(p)}$ can be computed without forming A_i by using a variant of *treppeniteration* as follows:

(2.4)
```
$Q_i^{(p+1)} \leftarrow I$
for $k = p, p-1, \ldots, 1$
    $C_i^{(k)} \leftarrow A_i^{(k)} Q_i^{(k+1)}$
    $C_i^{(k)} = Q_i^{(k)} R_i^{(k)}$    (QR factorization).
```

Then $A_i = Q_i^{(1)} R_i^{(1)} R_i^{(2)} \cdots R_i^{(p)} = Q_i^{(1)} R_i$, where each factor $R_i^{(k)}$, and therefore R_i, is upper triangular. This version of *treppeniteration* is used by Eckmann and Ruelle [7] for estimating Lyapunov exponents. It is also discussed as a method of computing product QR factorizations by Geist, Parlitz and Lautersborn [12] in their survey of different methods for computing Lyapunov exponents.

If each of the $A^{(k)}$ are $n \times n$, then one iteration of *treppeniteration* would take about $4n^3 p/3$ flops for the QR factorizations using Householder transformations [13, p. 212], and $2n^3(p-1)$ flops for the matrix multiplications (which could be reduced to about $n^3(p-1)$ flops by using the fact that the $R^{(k)}$ are triangular). The total cost of an iteration of *treppeniteration* is then about $n^3(7p/3 - 2)$ flops.

Since in (2.2) B_i converges to a diagonal matrix as $i \to \infty$, $R_i^T R_i$ approaches a diagonal matrix of eigenvalues of $A^T A$, and so R_i must converge to a diagonal matrix containing the singular values of A. Let $Q_i = Q_i^{(1)}$. Note that (2.3) can also give the singular vectors as shown below. One cycle of (2.3) gives $A_{i+1} = (Q_i^T A_i)^T = A_i^T Q_i$. Hence $A_{i+2} = (A_i^T Q_i)^T Q_{i+1} = Q_i^T A_i Q_{i+1}$, and so

$$
\begin{aligned}
A_{2i+1} &= Q_{2i-1}^T \cdots Q_3^T Q_1^T A_1 Q_2 Q_4 \cdots Q_{2i} \\
&= (Q_1 Q_3 \cdots Q_{2i-1})^T A (Q_2 Q_4 \cdots Q_{2i}) \\
&= U_i^T A V_i .
\end{aligned} \tag{2.5}
$$

Note that $A_i^{(k)} Q_i^{(k+1)} = Q_i^{(k)} R_i^{(k)}$ and so the singular values of $A_i^{(k)}$ and $R_i^{(k)}$ are identical. This means that $R_i^{(k)}$ is as well conditioned as $A_i^{(k)}$ in the 2-norm.

It should also be noted that the "product QR" factorization that is produced by *treppeniteration* is unique up to the signs of the entries if only real arithmetic is used. (If complex arithmetic is used, then the entries are unique up to unit factors $e^{i\theta}$ for some real θ.) This is shown more precisely in the following lemma.

LEMMA 1. *Suppose that*

$$A = Q_1^{(1)} R_1^{(1)} \cdots R_1^{(p)} (Q_1^{(p+1)})^T = Q_2^{(1)} R_2^{(1)} \cdots R_2^{(p)} (Q_2^{(p+1)})^T$$

are two product QR factorizations of $A = A^{(1)} \cdots A^{(p)}$ *with* $A^{(k)} = Q_i^{(k)} R_i^{(k)} (Q_i^{(k+1)})^T$ *for* $k = 1, 2, \ldots, p$, $i = 1, 2$. *If* $Q_1^{(p+1)} = Q_2^{(p+1)}$ *then there are diagonal matrices* $D^{(k)}$ *of the form diag* $(\pm 1, \ldots, \pm 1)$ *where* $Q_1^{(k)} = Q_2^{(k)} D^{(k)}$ *and* $R_1^{(k)} = D^{(k)} R_2^{(k)} D^{(k+1)}$ *for* $k = 1, 2, \ldots, p$.

Proof. Since $Q_1^{(p+1)} = Q_2^{(p+1)}$ we can take $D^{(p+1)} = I$.

Now, we proceed by induction backwards from $k = p + 1$ down to $k = 1$. Suppose that the induction hypothesis has been established for $k + 1 > 1$. Since

$$A^{(k)} = Q_1^{(k)} R_1^{(k)} (Q_1^{(k+1)})^T = Q_2^{(k)} R_2^{(k)} (Q_2^{(k+1)})^T,$$

by the induction hypothesis,

$$Q_1^{(k)} R_1^{(k)} D^{(k+1)} (Q_2^{(k+1)})^T = Q_2^{(k)} R_2^{(k)} (Q_2^{(k+1)})^T.$$

Premultiplying by $(Q_1^{(k)})^T$ and postmultiplying by $Q_2^{(k+1)} (R_2^{(k)})^{-1}$ gives

$$R_1^{(k)} D^{(k+1)} (R_2^{(k)})^{-1} = (Q_1^{(k)})^T Q_2^{(k)}.$$

The left hand side is an upper triangular matrix, while the right hand side is a real orthogonal matrix. Thus they must both be diagonal matrices with entries ± 1 on the diagonal. Let $D^{(k)} = ((Q_1^{(k)})^T Q_2^{(k)})^T = (Q_2^{(k)})^T Q_1^{(k)}$. Premultiplying by $Q_2^{(k)}$ gives $Q_1^{(k)} = Q_2^{(k)} D^{(k)}$. Then $R_1^{(k)} = D^{(k)} R_2^{(k)} D^{(k+1)}$ follows directly, and the induction hypothesis holds for k.

Thus by induction the result follows. □

Note that if the factorization is performed so as to ensure that the diagonal entries of the $R^{(k)}$ matrices are positive, then it can also be established that $D^{(k)} = I$ for all k, and the product QR factorization is unique.

This uniqueness result gives an efficient method for computing the "graded QR decomposition" of Stewart [23]: First form the QR factorization with column pivoting $A^{(p)} = Q^{(p)} R^{(p)} \Pi^T$ of $A^{(p)}$. Then form the product QR factorization of $A\Pi$ by treppeniteration. Since G.W. Stewart's graded QR factorization is a product QR factorization of $A\Pi$, this is the the result of the above procedure up to signs.

3. Convergence and deflation.

It is easy to show that with suitable choice of signs in $D_i = \operatorname{diag}(\pm 1, \pm 1, \ldots, \pm 1)$, $D_i R_i$ converges to the diagonal matrix of singular values of A as $i \to \infty$ [21]. However, this does not imply that $R_i^{(k)}$ converges as $i \to \infty$, for any k. What can be said is that, using $|A|$ to denote the matrix with entries $|a_{ij}|$,

$$|\operatorname{diag} R_i| = |\operatorname{diag}(R_i^{(1)} \cdots R_i^{(p)})| = |\operatorname{diag}(R_i^{(1)})| \cdots |\operatorname{diag}(R_i^{(p)})| \tag{3.1}$$

converges to the diagonal matrix of singular values as $i \to \infty$. The strictly upper triangular parts of $R_i^{(k)}$ generally do not converge to zero. (Counterexamples using a pair of 2×2 matrices can be easily found.)

To test for convergence, write

$$R_i = \begin{bmatrix} \overline{R}_i & \tilde{r}_i \\ 0 & r_i^* \end{bmatrix}, \qquad R_i^{(k)} = \begin{bmatrix} \overline{R}_i^{(k)} & \tilde{r}_i^{(k)} \\ 0 & r_i^{(k)*} \end{bmatrix}.$$

Then some arithmetic using $R_i R_i^T = R_{i+1}^T R_{i+1}$ gives

$$(r_{i+1}^*)^2 + \|\tilde{r}_{i+1}\|_2^2 = (r_i^*)^2, \tag{3.2}$$

ETNA
Kent State University
etna@mcs.kent.edu

so $||\tilde{r}_i||_2 \longrightarrow 0$ as $i \longrightarrow \infty$.

The convergence rate with which $R_i \longrightarrow$ diag can be estimated in the case of distinct singular values to be geometric with rate at least $\max_k(\sigma_{k+1}/\sigma_k)^2$. The above analysis, however, shows that there must still be convergence even if $\sigma_{k+1} = \sigma_k$ in the sense that $\tilde{r}_i \longrightarrow 0$. Once $\tilde{r}_i$ is sufficiently small, the matrix can be deflated by setting $\tilde{r}_i \longleftarrow 0$, and the argument can be recursively applied to the principal sub-matrix $\overline{R}_i$.

For many applications with long products, the singular values diverge exponentially quickly, so the ratios σ_{k+1}/σ_k are very small. If these ratios are all significantly smaller than $\mathbf{u}$, then only a small number of iterations are needed to obtain convergence. It has been found that for most Lyapunov exponent calculations with hundreds or thousands of factors, that only two iterations are needed.

Rank deficient matrices. Rank deficiency is no barrier to fast convergence. If the rank of R_i is r, then $\sigma_{r+1} = 0 < \sigma_r$, and so the convergence rate for the separation of the rank deficient part of R_i is faster than geometric convergence.

For R_i to be rank deficient, it must have a zero on the diagonal. This means that one of the factor matrices $R_i^{(k)}$ must also have a zero on its diagonal. For there to be a zero on the diagonal in that position at the end of the next iteration, it can be shown that the row corresponding to that diagonal entry must also be zero. If that is not so, then the zero entry will be pushed further down the diagonal. For there to be a zero entry in that diagonal position after two iterations, not only does the corresponding row need to be zero, but also the corresponding column will be zero after the iterations. Once this is done, standard QR factorization techniques (such as Householder transformations and Givens' rotations) will preserve the zeros in this row and this column.

Deflation. Deflation can be carried out when $|r^*_{i+1}/r^*_i|$ is sufficiently close to one, as $||\tilde{r}_{i+1}||_2 = r^*_{i+1}\sqrt{(r^*_i/r^*_{i+1})^2 - 1}$. This may give rise to numerical difficulties as $r^*_i/r^*_{i+1} = 1 + O(\mathbf{u})$ implies that $||\tilde{r}_{i+1}||_2 = O(\sqrt{\mathbf{u}})$. Fortunately, there is a better test which relates directly to the backward error.

Essentially deflation is just a matter of zeroing the $\tilde{r}_i$ vector. This can be done by zeroing the nth column of each $R^{(i)}$ matrix above the diagonal. This gives an exact decoupling of the last singular value, which will not be destroyed by later computations.

The deflation also results in small backward error if $||(\overline{R}_i)^{-1}\tilde{r}_i||_2$ is small. Provided $(R_i)_{kk}$ is not small relative to the magnitude of the kth row for each k, the quantity $||(\overline{R}_i)^{-1}\tilde{r}_i||_2$ can be computed to high absolute accuracy. This is because *rows* of R_i can be computed to high accuracy relative to the size of that row. Further, as $R_i \longrightarrow$ diag the diagonal entries will come to dominate the other entries in each row, so the assumption that "$(R_i)_{kk}$ *is not small relative to the magnitude of the* k*th row for each* k" will become true after some iterations.

Once $||(\overline{R}_i)^{-1}\tilde{r}_i||_2$ is of the same order as $\mathbf{u}$, deflation can proceed since for $\eta = (\overline{R}_i)^{-1}\tilde{r}_i$,

$$\begin{bmatrix} \overline{R}_i & \tilde{r}_i \\ 0 & r^*_i \end{bmatrix} \begin{bmatrix} I & -\eta \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \overline{R}_i & 0 \\ 0 & r^*_i \end{bmatrix}.$$

Thus zeroing the last column above the diagonal gives a multiplicative backward error, and hence a relative error in the singular values of size $||\eta||_2$, by the results in §6. Note that $(\overline{R}_i)^{-1}\tilde{r}_i$ can be computed to high absolute accuracy by using diagonal scaling.

4. Nonsquare factors.

Nonsquare matrices can make the calculations seem somewhat awkward. However, the product can be reduced, in a finite number of steps, to a product of square matrices. Consider an adjacent pair of factors $A^{(k)}A^{(k+1)}$. The first pass of the above algorithm computes

$$Q^{(k)}R^{(k)}R^{(k+1)} = A^{(k)}A^{(k+1)}Q^{(k+2)},$$

where both $R^{(k)}$ and $R^{(k+1)}$ are upper triangular. Suppose that $A^{(k)}$ is $m^{(k)} \times n^{(k)}$ and $A^{(k+1)}$ is $m^{(k+1)} \times n^{(k+1)}$. For the product to be meaningful, $n^{(k)} = m^{(k+1)}$.

If $m^{(k+1)} > n^{(k)} = m^{(k+2)}$, then as $R^{(k+1)}$ is upper triangular, the bottom $m^{(k+1)} - n^{(k+1)}$ rows of $R^{(k+1)}$ are zero. Thus, in the product $R^{(k)}R^{(k+1)}$, the right-most $m^{(k+1)} - n^{(k+1)}$ columns of $R^{(k)}$ are multiplied by a zero submatrix. Thus the product $R^{(k)}R^{(k+1)}$ is unchanged if the last $m^{(k+1)} - n^{(k+1)}$ rows of $R^{(k+1)}$ and the last $m^{(k+1)} - n^{(k+1)}$ columns of $R^{(k)}$ are deleted. Also note that if $m^{(1)} > n^{(1)}$, then the bottom $m^{(1)} - n^{(1)}$ rows of $R^{(1)}$ can be deleted without changing the singular values or rank of the product.

If we say that after i passes, $R_i^{(k)}$ is an $m_i^{(k)} \times n_i^{(k)}$ matrix after deleting irrelevant rows and columns, then

$$m_i^{(1)} \leq n_i^{(1)} = m_i^{(2)} \leq n_i^{(2)} = m_i^{(3)} \leq \cdots \leq n_i^{(p)}. \tag{4.1}$$

Since the second step involves a transposition that reverses the order of the factors, and transposes each factor,

$$m_{i+1}^{(k)} \leq n_i^{(p-k+1)}, \qquad n_{i+1}^{(k)} \leq m_i^{(p-k+1)}.$$

Thus $m_{i+2}^{(k)} \leq m_i^{(k)}$ and $n_{i+2}^{(k)} \leq n_i^{(k)}$. Since these quantities are non-negative integers, they must eventually be constant for even i, with $m_{i+1}^{(k)} = n_i^{(p-k+1)}$ and $n_{i+1}^{(k)} = m_i^{(p-k+1)}$. Using this in (4.1) for sufficiently large j gives the reversed inequalities:

$$n_{j+1}^{(p)} \leq m_{j+1}^{(p)} = n_{j+1}^{(p-1)} \leq m_{j+1}^{(p-1)} = n_{j+1}^{(p-2)} \leq \cdots \leq m_{j+1}^{(1)}. \tag{4.2}$$

Combining (4.1) and (4.2) shows that the factor matrices will eventually become square under this process of deleting irrelevant rows.

5. Error analysis.

The main result of this section is that the computed singular values of $A = A^{(1)} \cdots A^{(p)}$ are exact singular values for a perturbed matrix

$$A + E = (A^{(1)} + E^{(1)}) \cdots (A^{(p)} + E^{(p)}).$$

In terms of the number of iterations of LRCH used (N), the maximum number of rows or columns in a factor matrix (n), and machine roundoff ($\mathbf{u}$), these perturbations $\|E^{(k)}\|$ can be bounded by quantities of order $\mathbf{u} n^{3/2} N \|A^{(k)}\|$ if no deflation is used. The backward error that occurs in deflating can be made quite small as is shown below, and can be bounded by easily computable quantities. In what follows we use matrices with carets ($\widehat{\ }$) to indicate the corresponding quantities computed using finite precision arithmetic, as opposed to exact arithmetic. The function $fl(\cdots)$ will be used to denote the results of a computation. The error analysis is based on that given in Wilkinson [24], and Golub and Van Loan [13]. It is assumed throughout that Householder operations are used to computed the QR factorizations, and that a standard matrix multiplication procedure is used (and not, for example, Strassen's method), and that $n^{3/2}\mathbf{u} \leq 0.01$. Larger constants than 0.01 can be used here, provided the values of the constants K_i in the proof below are increased accordingly. The $\|\cdot\|_F$ norm is the Frobenius norm: $\|A\|_F = \sqrt{\text{trace}(A^T A)}$.

THEOREM 1. *If $\widehat{\Sigma} = \widehat{R}^{(1)} \cdots \widehat{R}^{(p)}$ is the matrix of computed singular values in N steps without deflation, then $\widehat{\Sigma}$ contains the exact singular values of a matrix product*

$$(A^{(1)} + E^{(1)})(A^{(2)} + E^{(2)}) \cdots (A^{(p)} + E^{(p)}),$$

where $||E^{(i)}||_F = O(Nn^{3/2}\mathbf{u}||A^{(i)}||_F)$ *provided* $Nn^{3/2}\mathbf{u}\max_i ||A^{(i)}||_F \leq 1$.

Proof. In what follows, K_i will denote an appropriate absolute quantity (for example, $K_0 = 1.01$ and $K_1 = 12.36$). The precise value of these constants is not important, and all are of modest size.

The lowest level operations in (2.4) are matrix multiplications and QR factorizations. From Golub and Van Loan [13, §2.4.8, p. 66],

$$fl(AB) = AB + E, \qquad \text{where} \quad |E| \leq K_0 n\mathbf{u}|A|\,|B|.$$

From this it is evident that, using Frobenius norms,

$$||fl(AB) - AB||_F \leq K_0 n\mathbf{u}||A||_F||B||_F.$$

The matrix $\widehat{Q}$ computed by the QR factorization of a matrix A with $\widehat{A} = \widehat{Q}\widehat{R}$, is nearly orthogonal in that there is an exactly orthogonal matrix Q, where

$$||\widehat{Q} - Q||_F \leq (n-1)(1 + K_1\mathbf{u})^{n-2}K_1\mathbf{u}||A||_F \leq K_2 n^{3/2}\mathbf{u},$$

and

$$||\widehat{R} - Q^T A||_F \leq (n-1)(1 + K_1\mathbf{u})^{n-2}K_1\mathbf{u}||A||_F \leq K_2 n\mathbf{u}||A||_F.$$

Now,

$$\widehat{C}_i^{(k)} = fl(\widehat{A}_i^{(k)}\widehat{Q}_i^{(k+1)}) = \widehat{A}_i^{(k)}\widehat{Q}_i^{(k+1)} + E_{0,i}^{(k)},$$

where $||E_{0,i}^{(k)}||_F \leq K_0 n\mathbf{u}||\widehat{A}_i^{(k)}||_F||\widehat{Q}_i^{(k+1)}||_F$. Since

$$\begin{aligned} ||\widehat{Q}_i^{(k+1)}||_F &\leq ||Q_i^{(k+1)}||_F + ||\widehat{Q}_i^{(k+1)} - Q_i^{(k+1)}||_F \\ &\leq \sqrt{n} + K_2 n^{3/2}\mathbf{u} \leq (1 + K_2\mathbf{u}n)\sqrt{n}, \end{aligned}$$

this gives $||E_{0,i}^{(k)}||_F \leq K_3 n^{3/2}\mathbf{u}||\widehat{A}_i^{(k)}||_F$.

Now write

$$\widehat{C}_i^{(k)} = \widehat{A}_i^{(k)}\widehat{Q}_i^{(k+1)} + E_{0,i}^{(k)} = \widehat{A}_i^{(k)}Q_i^{(k+1)} + E_{1,i}^{(k)},$$

where $E_{1,i}^{(k)} = \widehat{A}_i^{(k)}(\widehat{Q}_i^{(k+1)} - Q_i^{(k+1)}) + E_{0,i}^{(k)}$. Then

$$||E_{1,i}^{(k)}||_F \leq ||\widehat{A}_i^{(k)}||_F(K_2 n^{3/2}\mathbf{u} + K_3 n^{3/2}\mathbf{u}) = K_4 n^{3/2}\mathbf{u}||\widehat{A}_i^{(k)}||_F.$$

Thus,

$$\widehat{C}_i^{(k)} = (\widehat{A}_i^{(k)} + E_{2,i}^{(k)})Q_i^{(k+1)}$$

with $||E_{2,i}^{(k)}||_F \leq K_4 n^{3/2}\mathbf{u}||\widehat{A}_i^{(k)}||_F$ since $E_{2,i}^{(k)} = E_{1,i}^{(k)}(Q_i^{(k+1)})^T$.

The QR factorization step in (2.4) computes the QR factorization of $\widehat{C}_i^{(k)}$. Then

$$\begin{aligned} ||\widehat{C}_i^{(k)} - Q_i^{(k)}\widehat{R}_i^{(k)}||_F &\leq K_2 n\mathbf{u}||\widehat{C}_i^{(k)}||_F \\ &\leq K_2 n\mathbf{u}(1 + K_4 n^{3/2}\mathbf{u})||\widehat{A}_i^{(k)}||_F \\ &\leq K_5 n\mathbf{u}||\widehat{A}_i^{(k)}||_F, \end{aligned} \tag{5.1}$$

and $\|Q_i^{(k)} - \widehat{Q}_i^{(k)}\|_F \le K_2 n^{3/2}\mathbf{u}$. So $Q_i^{(k)}\widehat{R}_i^{(k)} = \widehat{C}_i^{(k)} + E_{3,i}^{(k)}$ where

$$\|E_{3,i}^{(k)}\|_F \le K_2 n\mathbf{u}(1 + K_4 n^{3/2}\mathbf{u})\|\widehat{A}_i^{(k)}\|_F.$$

Hence,

$$\widehat{R}_i^{(k)} = Q_i^{(k)T}((\widehat{A}_i^{(k)} + E_{2,i}^{(k)})Q_i^{(k+1)} + E_{3,i}^{(k)}) = Q_i^{(k)T}(\widehat{A}_i^{(k)} + E_{4,i}^{(k)})Q_i^{(k+1)},$$

where $Q_i^{(k)}$ and $Q_i^{(k+1)}$ are exactly orthogonal, and

$$\|E_{4,i}^{(k)}\|_F \le \|E_{2,i}^{(k)}\|_F + \|E_{3,i}^{(k)}\|_F \le (K_5 + K_4\sqrt{n})n\mathbf{u}\|\widehat{A}_i^{(k)}\|_F.$$

This is the first step of (2.3); the second step of (2.3) simply involves a transpose, which involves transposing and reordering the $R_i^{(k)}$ matrices. Neither of these operations in the second step of (2.3) involves roundoff error.

Thus, $\widehat{A}_{i+1}^{(k)} = \widehat{R}_i^{(p-k+1)T} = Q_i^{(p-k)T}(\widehat{A}_i^{(p-k+1)} + E_{4,i}^{(p-k+1)})^T Q_i^{(p-k+1)}$ with

$$\|E_{4,i}^{(p-k+1)}\|_F \le (K_5 + K_4\sqrt{n})n\mathbf{u}\,\|\widehat{A}_i^{(p-k+1)}\|_F. \tag{5.2}$$

This implies that

$$\|\widehat{A}_{i+1}^{(k)}\|_F \le (1 + K_6\, n^{3/2}\mathbf{u})\|\widehat{A}_i^{(p-k+1)}\|_F \le K_7\|\widehat{A}_i^{(p-k+1)}\|_F.$$

So two applications of LRCH gives

$$\widehat{A}_{i+2}^{(k)} = \widehat{R}_i^{(p-k+1)T} = Q_{i+1}^{(p-k)T}Q_i^{(k)T}(\widehat{A}_i^{(k)} + E_{5,i}^{(k)})Q_i^{(k+1)}Q_{i+1}^{(p-k+1)}, \tag{5.3}$$

where

$$\|E_{5,i}^{(k)}\|_F \le (K_9 + K_8\sqrt{n})n\mathbf{u}\|\widehat{A}_i^{(k)}\|_F \le (K_8 + K_9)n^{3/2}\mathbf{u}\|\widehat{A}_i^{(k)}\|_F. \tag{5.4}$$

If no deflation is used then it is straightforward to see that N applications of LRCH gives the same singular values as a perturbation to each factor of A of size
$O(n^{3/2}\mathbf{u}N\|A^{(k)}\|_F)$ in the Frobenius norm. □

6. Stability. The question of stability of the problem cannot be answered completely here. However, a partial indication can be given by the following result.

THEOREM 2. *If* $A \in \mathbf{R}^{m\times n}$ *and* $E \in \mathbf{R}^{m\times n}$, *then* $|\sigma_i((I+E)A)/\sigma_i(A) - 1| \le \|E\|_2$ *for* $i = 1, \ldots,$ *rank*(A).

This follows from a standard result for multiplicative bounds of singular values: $\sigma_{i+j-1}(AB) \le \sigma_i(A)\sigma_j(B)$. See [15, p. 423, Ex. 18], and [16, p. 178, Thm. 3.3.16(d)]. This is an example of an outer multiplicative perturbation. Note that the deflation operations are considered in section 3.

The problem with trying to prove stability of the problem of computing the singular values to high relative precision is that of turning a perturbation of each factor into a *multiplicative* perturbation of the product. Note firstly that for a particular factor, $A + E = A(I + F)$ where $F = A^{-1}E$. If $\|E\|_2 = O(\mathbf{u}\|A\|_2)$, then $\|F\|_2 = O(\mathbf{u}\kappa_2(A))$. A similar result will give multiplicative perturbations of each factor on the left with the same bounds. Provided none of the *factors* is ill-conditioned, this gives small multiplicative backward errors for each factor.

The relative stability of the singular values to multiplicative perturbations does not hold in general as the following example shows. Suppose that $0 < \eta = 2^{-k} \ll \epsilon_1, \epsilon_2 \ll 1$ are chosen and $A = \operatorname{diag}(1/2, 1)$ and $B = \operatorname{diag}(1, 1/2)$. Then η is the repeated singular value of the product

$$A^k B^k = \begin{bmatrix} \eta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \eta \end{bmatrix} = \eta I.$$

Let $I + E = \begin{bmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 1 \end{bmatrix}$. Then

$$A^k (I + E) B^k = \begin{bmatrix} \eta & \epsilon_1 \eta^2 \\ \epsilon_2 & \eta \end{bmatrix} = \eta \begin{bmatrix} 1 & \epsilon_1 \eta \\ \epsilon_2/\eta & 1 \end{bmatrix}$$

, which has a singular value of size $\approx \epsilon_2$.

On the other hand, the singular values of some products are not changed excessively by perturbations of the factors. For example, consider the product of k factors $B^k = \operatorname{diag}(1, 1/2)^k$. Then for any $0 \leq l \leq k$,

$$B^l \begin{bmatrix} 1 & \epsilon_1 \\ \epsilon_2 & 1 \end{bmatrix} B^{k-l} = \begin{bmatrix} 1 & 0 \\ 2^{-l}\epsilon_2 & 1 \end{bmatrix} B^k \begin{bmatrix} 1 & 2^{-(k-l)}\epsilon_1 \\ 0 & 1 - \epsilon_1\epsilon_2 \end{bmatrix}.$$

Clearly the ill-conditioning of the product does not necessarily make the singular values sensitive to perturbations of the factors.

The product $B^k = \operatorname{diag}(1, 1/2)^k$ is an example of a *stable product*. The above result can be generalized to replace any collection of inner multiplicative perturbations

$$B(I + \widetilde{E}^{(1)})B(I + \widetilde{E}^{(2)})B \cdots (I + \widetilde{E}^{(p)})B$$

with a pair of outer multiplicative perturbations

$$(I + F_+)B^k(I + F_-),$$

where F_+ and F_- have a size comparable to $\max_i \|\widetilde{E}^{(i)}\|$. More formally,

DEFINITION 1. *An infinite product* $\cdots A^{(-1)}A^{(0)}A^{(1)}\cdots$ *is a* stable product *if there are constants* C *and* $\epsilon_0 > 0$ *such that for any* $i < j$ *and matrices* $E^{(k)}$ *with* $\|E^{(k)}\| \leq \epsilon_0$,

$$A^{(i)}(I + E^{(i)})A^{(i+1)} \cdots (I + E^{(j-1)})A^{(j)} = (I + F_-)A^{(i)}A^{(i+1)} \cdots A^{(j)}(I + F_+)$$

for some matrices F_-, F_+, *where* $\|F_-\|, \|F_+\| \leq C \max_{i \leq k \leq j} \|E^{(k)}\|$.

A finite product $A^{(1)}A^{(2)} \cdots A^{(p)}$ *is stable if the above holds for a* modest *constant* C *(which does not grow exponentially in* p*).*

If a product $A^{(1)}A^{(2)} \cdots A^{(p)}$ is a stable product in this sense, and each factor is well conditioned, then the perturbed product

$$(A^{(1)} + E^{(1)})(A^{(2)} + E^{(2)}) \cdots (A^{(p)} + E^{(p)})$$

can be represented in terms of inner and outer multiplicative perturbations

$$(I + E^{(1)}(A^{(1)})^{-1})A^{(1)}(I + E^{(2)}(A^{(2)})^{-1})A^{(2)} \cdots (I + E^{(p)}(A^{(p)})^{-1})A^{(p)},$$

which can then be represented in terms of outer multiplicative perturbations only:

$$(I + F_-)A^{(1)}A^{(2)} \cdots A^{(p)}(I + F_+),$$

where $\|F_+\|, \|F_-\| \le C\ \max_i \|E^{(i)}\|\ \|(A^{(i)})^{-1}\|$ for a modest constant C, and where the perturbations $E^{(i)}$ are all sufficiently small.

There are results showing the backward stability of Lyapunov exponents under *hyperbolicity* assumptions. See, for example, Dieci, Russell and Van Vleck [6] which shows stability of the Lyapunov exponents under the assumption that the linear system

$$\widehat{\Phi}_{k+1} = e^{-\gamma} A^{(k)} \widehat{\Phi}_k$$

is hyperbolic whenever γ is not a Lyapunov exponent. That is, there is a pair of (possibly non-orthogonal) projectors P_k^s and P_k^u for each k, modest constants $K > 0$ and $0 < \alpha < 1$ where $P_k^s + P_k^u = I$, $\|P_k^s\|$ and $\|P_k^u\|$ are bounded independently of k, range $P_{k-1}^s = A^{(k)}$range P_k^s and similarly for P_k^u, and for $l \ge k$,

$$e^{-(l-k)\gamma} \|P_{k-1}^s A^{(k)} A^{(k+1)} \cdots A^{(l-1)} A^{(l)} P_l^s\| \le K\alpha^{l-k},$$
$$e^{-(l-k)\gamma} \|(P_{k-1}^u A^{(k)} A^{(k+1)} \cdots A^{(l-1)} A^{(l)} P_l^u)^{-1}\| \le K\alpha^{l-k},$$

where the latter inverse is of the product considered as a linear map range $P_k^u \to$ range P_l^u. Whether a given product of matrices satisfies this hyperbolicity assumption is a difficult question to answer. It is, however, a question worth further investigation from the point of view of numerical analysis.

7. Rank deficient products. A *product* of matrices $A = A^{(1)} A^{(2)} \cdots A^{(p)}$ is defined here to be *numerically rank deficient* if there are matrices $E^{(i)}$, $i = 1, \ldots, p$ where $\|E^{(i)}\|_2 \le C\mathbf{u}\|A^{(i)}\|_2$ for some modest constant C, and the perturbed product

$$(A^{(1)} + E^{(1)})(A^{(2)} + E^{(2)}) \cdots (A^{(p)} + E^{(p)})$$

is exactly rank deficient. This "backward error" definition of rank deficiency means that simply looking at the (computed) singular values is not enough to determine if a product is rank deficient. For example, the product of $(k+1)$ factors

$$\begin{bmatrix} 1 & \\ & 1/2 \end{bmatrix}^k \begin{bmatrix} \mathbf{u} & \\ & 1 \end{bmatrix}$$

is rank deficient because of the singular value $\mathbf{u}$ even though the other singular value 2^{-k} might be far, far smaller. On the other hand

$$\begin{bmatrix} 1 & \\ & 1/2 \end{bmatrix}^k$$

is not a numerically rank deficient *product* for *any* k. Since the main algorithm presented here is backward stable with respect to the *factors* of a product, this definition of a numerically rank deficient product is appropriate.

Detecting numerical rank deficiency in products of square matrices is comparatively easy: all that is needed is to check that each factor is numerically nonsingular. This can be done by performing an SVD on each factor and looking at the ratio of the smallest to the largest singular values, and if this is less than $C\mathbf{u}$ for any factor, then the factor and the product is rank deficient.

This simple test cannot be applied directly to products of non-square matrices. However, if the technique described in §4 is applied, then the product is transformed to become a product of square matrices. However, the test needs to be modified to take the ratio of the smallest singular value of each computed square factor, to the largest singular value of the

ETNA
Kent State University
etna@mcs.kent.edu

corresponding *original* factor $A^{(k)}$. If this ratio is less than $C\mathbf{u}$ then the product is rank deficient.

Note, however, that determining the (numerical) rank deficiency of a product where more than one factor is rank deficient, is a more complex task, requiring the computation of null spaces of factors, and determining how these subspaces are transformed by the other factors.

8. Accelerating convergence. Convergence is accelerated in this algorithm by using a shifting strategy together with deflation. The LRCH algorithm can be shifted, and this is indeed part of the original algorithm in [21]. However, the shift must be smaller than the smallest singular value of $R = R^{(1)}R^{(2)}\cdots R^{(p)}$ as otherwise the Cholesky factorization of $R^TR - \sigma I$ does not exist.

The standard shifting strategy of setting s to be (smallest) eigenvalue of the bottom right-hand 2×2 submatrix of R^TR was modified in [21], and combined with a suitable mechanism for handling the occasional failure of this technique.

The shift cannot be done directly, as it is in LRCH, but must be done implicitly on the product. Note that the product R^TR does not need to be formed if we apply hyperbolic rotations to

$$S = \begin{bmatrix} R \\ \tau I \end{bmatrix} \in \mathbf{R}^{2n\times n}, \tag{8.1}$$

where $\tau^2 = \sigma$. Applying hyperbolic rotations to S gives

$$HS = \begin{bmatrix} R_+ \\ 0 \end{bmatrix} \in \mathbf{R}^{2n\times n}, \tag{8.2}$$

where H is the product of partitioned hyperbolic rotations.

Let J be an elementary partitioned hyperbolic rotation. Then

$$J = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & c & \dots & s & & \\ & & \vdots & & \vdots & & \\ & & s & \dots & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} \tag{8.3}$$

where $c^2 - s^2 = 1$. Then,

$$J^T \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} J = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}. \tag{8.4}$$

Thus, applying suitably partitioned hyperbolic rotations to $S = \begin{bmatrix} X \\ Y \end{bmatrix}$, leaves $X^TX - Y^TY$ invariant. Thus, $R_+^TR_+ = R^TR - \tau^2 I$. Noting that $||S||_F^2 = \text{trace}\,(S^TS)$, taking traces shows that hyperbolic rotations will leave $||X||_F^2 - ||Y||_F^2$ invariant. Hence $||R_+||_F^2 = ||R||_F^2 - n\tau^2$.

Hyperbolic rotations are not much used in numerical analysis as they can be ill-conditioned. In particular, $4c^2 \geq \kappa_2(J) = (c + |s|)^2 \geq c^2$. Care should therefore be taken to avoid excessively large values of c. This can be avoided if we choose τ so that $\tau/\sigma_n(R)$ is not too close to one. A combination of two strategies can be used to avoid problems: (1) the modification of the matrix can be aborted if the values of $|c|$ are excessive, and the original factors can be restored with only the original first factor saved; (2) choose τ so that $\tau/\widehat{\sigma}_n(R)$ is no more than, say, 0.95 where $\widehat{\sigma}_n(R)$ is the estimate of the smallest singular value.

The hyperbolic rotations are applied to zero successive rows of the bottom $n \times n$ submatrix of S, starting with the bottom row. After the $(2n-k)$th row of S is zeroed, $Y = \text{diag}\,(\tau, \ldots, \tau, 0, \ldots, 0)$ and $||Y||_F = \tau\sqrt{n-k-1}$. So $||X||_F^2 = ||R||_F^2 + (||Y||_F^2 - n\tau^2) < ||R||_F^2$. Thus, the size of X in the Frobenius norm is decreased by each row elimination. Note also that if $|c|$ is of modest size for the hyperbolic rotations then the 2-norms of the rows of Y are of the same order as $\tau < \sigma_n(X)$.

In order to determine the appropriate shift and hyperbolic rotations the actual values of R are needed. This can be done row by row, so that each computed row of R has an error that is small relative to its own 2-norm. By using a separate exponent this can also be done in a way that avoids overflow except for extremely long products of matrices or extremely large entries. Once the parameters of a hyperbolic rotation are computed they need to be applied to the rows of S. However, these hyperbolic rotations also need to be applied to the product $R^{(1)}R^{(2)}\cdots R^{(p)}$.

Rather than apply hyperbolic rotations directly to

$$S = \begin{bmatrix} R \\ \tau I \end{bmatrix},$$

we take out the factor of R on the right to give

$$S = S'R = \begin{bmatrix} I \\ \tau R^{-1} \end{bmatrix} R.$$

Then we can apply the hyperbolic rotations to S'. The application of partitioned hyperbolic rotations to S' can in fact produce an upper triangular matrix. If H' is the set of rotations applied to S' to make S' upper triangular, then

$$H'S = H'S'\,R = \begin{bmatrix} S_+ \\ 0 \end{bmatrix} R.$$

Then we can replace $R^{(1)}$ with $S_+R^{(1)}$. This can be done with small backward error, and since this error can be made into an outer multiplicative error, this results in only a small *relative* perturbation of the singular values. An alternative to actually doing this multiplication is to make S_+ the first factor. The implementations of this algorithm of the author has not used this technique. To recover the singular values of R, if $\widehat{\sigma}_i$ is the ith singular value of S_+R, then the ith singular value of R is $\sigma_i = \sqrt{\tau^2 + \widehat{\sigma}_i^2}$.

Note that as $\tau < \sigma_n(R)$, $\sigma_i(\tau R^{-1}) \leq \tau/\sigma_n(R) < 1$ so $I - \tau^2 R^{-T}R^{-1}$ is positive definite. Thus, τR^{-1} is of modest size and can be computed to high absolute accuracy, as each row of R^{-1} can be computed to high accuracy relative to the 2-norm of that row.

This shifting strategy requires about $n^3(p-1)/3$ flops to compute the product R and the row scales. Each hyperbolic rotation requires 6 flops and one square root to compute the c and s values; the total amount of work needed to compute and apply all the hyperbolic rotations is about n^3 flops plus n square roots. The total amount of computation needed for the shifting strategy is about $n^3(p+2)/3$ flops and n square roots. This is of the same order as a single pass of the basic algorithm. If each iteration of the above algorithm incorporates shifting at each step, then the cost of a single complete iteration is about $n^3(8p-4)/3$ flops and n square roots.

9. Numerical results. The LRCH algorithm, optionally including shifting using hyperbolic rotations and deflation, has been implemented in 'C' using the Meschach matrix library [22]. All calculations using the C implementations were performed on a 95MHz Intel Pentium running Linux. Double precision ($\mathbf{u} \approx 2.2 \times 10^{-16}$) was used throughout. The unshifted algorithm was also implemented in MATLAB^{TM} [18]. The version implementing shifting, sets τ to be 0.95 times the smallest singular value of the bottom 2×2 submatrix of R.

Deflation occurred when $\|\eta\|_2 < 10^{-15}$. All computations were done in double precision with $\mathbf{u} \approx 2 \times 10^{-16}$.

For comparison, the Jacobi method of Bojanczyk *et al.* [3] was implemented in Meschach. This used the scheme of Heath, Laub, Paige and Ward [14] for scheduling the Givens rotations of the product. It was found to be necessary to explicitly zero entries in the factors; without this step it was found that the product matrix with the scaled rows did not converge to a diagonal matrix to within the specified tolerance of 10^{-15}.

These methods were applied to a number of test problems to determine their reliability and accuracy. The results for these examples are summarized in Table 9.1. The results in this table show the number of floating point operations, number of square roots and the number of iterations needed for the methods discussed in this paper. The first is the LRCH method without shifts (LRCH); the second is the LRCH method with shifts (LRCH/sh); third is the Jacobi based method of Bojanczyk *et al.* [3]; fourth is the graded QR algorithm of G.W. Stewart implemented essentially as described in [23] (GRQR1); the fifth is the graded QR algorithm implemented by taking the QR factorization of $A^{(p)}$ with column pivoting, then then forming the QR factorization of the product times the permutation matrix by *treppeniteration* (GRQR2). These last two methods give identical results in exact arithmetic as noted above; comparison of the results shown relative discrepancies between the approximate singular values between GRQR1 and GRQR2 of no more than 5×10^{-14} for the first three examples. In Table 9.1, the errors are actually the natural logarithms of the ratios of the computed results to the true answer. Dashes are used to indicate results for which the errors cannot be reliably determined. Note that for examples 4 and 5, there were discrepancies between the LRCH results and the Jacobi results of about 1×10^{-5} and 3×10^{-4} respectively. This discrepancy can be explained in part by noting that the "average" condition number of the factor matrices can be estimated, using the Lyapunov exponents, to be about 5×10^{6}; for 1000 3×3 factors with machine epsilon (2.2×10^{-16}) perturbations to each entry, an error estimate of about 3×10^{-6} can be obtained without considering the effects of any "instability" in the product.

Example 1. The first test problem was the product

$$A = (A^{(1)})^{20} = \begin{bmatrix} 10^4 & 10^{-2} & 0 \\ 10^{-2} & 1 & 10^{-2} \\ 0 & 10^{-2} & 1 \end{bmatrix}^{20}.$$

The largest eigenvalue is close to $(10^4)^{20} = 10^{80}$ while the two smaller eigenvalues are relatively close to 1. MATLAB gave the eigenvalues of $A^{(1)}$ to be $1.00000000000100 \times 10^4$, 1.00999999499982 and 0.98999999499982; the 20th power of these are the singular values of A, which are, according to MATLAB,
$1.00000000002002 \times 10^{80}$, 1.22018991913264 and 0.81790685497746. The smaller eigenvalues are not extremely close together, yet the asymptotic convergence rate (without shifting) of $\approx 0.8179/1.2202 \approx 0.67$ is not very good.

Example 2. This is the same as the first example except that

$$A^{(1)} = \begin{bmatrix} 1 & 10^{-2} & 0 \\ 10^{-2} & 1 & 10^{-2} \\ 0 & 10^{-2} & 10^4 \end{bmatrix}.$$

The eigenvalues of $A^{(1)}$ are the same for this problem, and so the singular values of the product are also the same.

Example 3. This example is a product of 100 random 5×5 matrices with entries independently drawn from a uniform probability distribution on $[-1,+1]$. The theory of Lyapunov exponents due to Oseledec [19] shows that infinite products of such matrices have well-defined Lyapunov exponents.

Example 4. This example illustrates the use of product SVD algorithms for computing Lyapunov exponents for dynamical systems. The product is a product of 1000 factors which was obtained by integrating the Lorenz equations [17, 20], which were developed as a very crude approximation to the behavior of weather systems. The differential equations are [20, p. 180]

$$\begin{aligned} x' &= \sigma(y-x), \\ y' &= \rho x - y - xz, \\ z' &= -\beta z + xy. \end{aligned}$$

The standard values of the parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$, were used in the calculations. The factor matrices were generated by integrating the variational equations $\Phi' = \nabla f(x(t))\,\Phi$ along with the differential equations $x' = f(x)$. This integration was performed numerically using the standard fixed step-size fourth order Runge–Kutta method with step size $h = 0.01$ over intervals of length one, after which the integration is re-started with Φ reset to the identity matrix.

Example 5. This is the same as example 4, except that 10,000 factors were used with the same integration scheme, step sizes, interval lengths, and initial conditions. Algorithm GRQR2 was not use for this problem as it would take an excessive length of time to run. Note that GRQR2 is an $O(p^2\,n^3)$ algorithm, so increasing p by a factor of ten roughly increases the running time by a factor of 100, requiring about 6.3 Gflops to finish.

The singular values are, to eight digits, $2.1368676 \times 10^{+3941}$, 1.0701122, and $3.1999193 \times 10^{-63295}$, which correspond to Lyapunov exponents $+0.9075247$, 6.776×10^{-6}, and -14.5740960. These are for integration over a time period of 10^4. Note that from the theory of Lyapunov exponents, if a trajectory of an ODE does not come arbitrarily close to fixed points or go to infinity, then zero is a Lyapunov exponent for that trajectory. Since only a finite time integration of the Lorenz equations have been used, and a Runge–Kutta method is used to approximate the trajectory, it cannot be expected that any of the computed Lyapunov exponents would be exactly zero. However, the computed Lyapunov exponent of $\approx 6.8 \times 10^{-6}$ seems to be close to zero.

The relative differences between the singular values computed by the LRCH algorithm and the Jacobi algorithm for examples #4 and #5 are greatest for the smallest singular value, and the logarithm of the ratios is about 4.6×10^{-4} and 5.3×10^{-4} for examples #4 and #5 respectively. For the second largest singular value, the relative discrepancies are 2.9×10^{-13} and 1.1×10^{-13}, and are equally small for the largest singular values of the products.

Test problems of the type given in G.W. Stewart [23] were also used for making comparisons. The products $ABAB \cdots BA$ with $2m+1$ were formed where $A = U\Sigma V^T$, $B = V\Sigma U^T$ with randomly generated orthogonal matrices U and V, and diagonal matrices Σ. The SVD of the product $ABAB \cdots BA$ is $U\Sigma^{2m+1}V^T$. The choices for Σ used were $\Sigma = \Sigma_1 = \operatorname{diag}(1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4})$ and $\Sigma = \Sigma_2 = \operatorname{diag}(1, 0.99, 0.8, 0.7, 0.6)$.

The computational work for the different methods, problems, and values of m are shown in Table 9.2. The relative errors are shown in Table 9.3.

As can be seen from these tables, all algorithms considered here produce singular values that are accurate to essentially the full accuracy afforded by double precision arithmetic.

Example	Method	# iter'ns	# flops	# sqrt's	max. error
#1	LRCH	69	282,146	4140	4.7×10^{-14}
	LRCH/sh	8	34,009	575	2.3×10^{-14}
	Jacobi	1	7,278	132	2.3×10^{-14}
	GRQR1	–	2,7780	630	0.16
	GRQR2	–	2,646	60	0.16
#2	LRCH	69	282,146	4140	2.0×10^{-13}
	LRCH/sh	9	38,236	642	2.3×10^{-13}
	Jacobi	1	7,278	132	2.3×10^{-13}
	GRQR1	–	27,780	630	0.16
	GRQR2	–	2,646	60	0.16
#3	LRCH	3	205,689	1300	–
	LRCH/sh	3	206,731	1341	–
	Jacobi	2	278,120	2580	–
	GRQR1	–	3,127,800	25,250	0.19
	GRQR2	–	62,510	500	0.19
#4	LRCH	2	408,131	6000	–
	LRCH/sh	2	408,417	6016	–
	Jacobi	1	360,078	6012	–
	GRQR1	–	63,129,000	1,501,500	0.93
	GRQR2	–	132,006	3000	0.93
#5	LRCH	2	4,080,131	60,000	–
	LRCH/sh	2	4,080,417	60,016	–
	Jacobi	1	3,600,078	60,012	–
	GRQR2	–	1,320,006	30,000	1.63

TABLE 9.1
Results for examples 1 to 5

For $\Sigma = \Sigma_1$, all algorithms are competitive in terms of the amount of computation needed. However, the close singular values for $\Sigma = \Sigma_2$ cause problems for the LRCH algorithms, especially for the unshifted algorithm.

10. Conclusions. A new algorithm for computing SVD's of long products $A = A^{(1)}A^{(2)}\cdots A^{(p)}$ of matrices are given which is based on the LRCH algorithm of Rutishauser and Schwartz [21], and *treppeniteration*, and has complexity $O(n^3 p)$. It avoids computing the product, and avoids creating extremely ill-conditioned matrices, which the product A often is. Unlike the algorithm of Abarbanel *et al.* [1], the LRCH algorithm lends itself to a backward error analysis in terms of the factors $A^{(k)}$ of A.

The unshifted LRCH algorithm can converge slowly, though shifting can be incorporated by means of hyperbolic rotations. Care must be taken with the shifting strategy, as when the shifts are too large, the shifting operation may need to be re-started with a new shift as was done by Rutishauser and Schwartz [21].

For long products of matrices, the shifting is not usually necessary for rapid convergence. In this case the LRCH algorithm is competitive with the Jacobi method [3], although for small systems and short products, the Jacobi method appears to be the faster method. Both methods are numerically stable and produce highly consistent answers.

	# iter'ns	# flops	# sqrt's
$m = 5$			
LRCH	3	30 558	165
LRCH/sh	3	32 061	217
Jacobi	2	31 056	355
$m = 10$			
LRCH	2	38 892	210
LRCH/sh	2	39 852	239
Jacobi	1	35 876	355
$m = 20$			
LRCH	2	75 932	410
LRCH/sh	2	76 892	439
Jacobi	1	69 796	655

$\Sigma = \Sigma_1$

	# iter'ns	# flops	# sqrt's
$m = 20$			
LRCH	80	3 075 246	16 605
LRCH/sh	13	232 966	1 934
Jacobi	3	158 876	1 555
$m = 40$			
LRCH	41	3 150 252	17 010
LRCH/sh	11	342 341	2 885
Jacobi	2	225 376	2 105
$m = 80$			
LRCH	21	3 279 892	17 710
LRCH/sh	9	516 879	4 447
Jacobi	2	447 456	4 105

$\Sigma = \Sigma_2$

TABLE 9.2
G.W. Stewart test examples: Computational work

REFERENCES

[1] H. D. I. ABARBANEL, R. BROWN, AND M. B. KENNEL, *Local Lyapunov exponents computed from observed data*, Chaos: An Int. J. of Nonlinear Science, 2 (1992), pp. 343–365.

[2] L. ARNOLD, H. CRAUEL, AND J.-P. ECKMANN, *Lyapunov exponents*, Springer–Verlag, Berlin, New York, 1991. Lecture Notes in Mathematics #1486.

[3] A. W. BOJANCZYK, M. EWERBRING, F. T. LUK, AND P. VAN DOOREN, *An accurate product SVD algorithm*, in SVD and Signal Processing II, R. J. Vaccaro, ed., Amsterdam, New York, 1991, Elsevier Science Publishers, pp. 113–131.

[4] B. DE MOOR, *Generalizations of the singular value and QR decompositions*, Signal Processing, 25 (1991), pp. 135–146.

[5] B. DE MOOR AND H. ZHA, *A tree of generalizations of the ordinary Singular Value Decomposition*, Linear Algebra Appl., 147 (1991), pp. 469–500.

[6] L. DEICI, R. D. RUSSELL, AND E. S. VAN VLECK, *On the computation of Lyapunov exponents for continuous dynamical systems*. Georgia Institute of Technology Technical Report Math:040893–006, 1993.

[7] J.-P. ECKMANN AND D. RUELLE, *Ergodic theory of chaos and strange attractors*, Rev. Modern Physics, 57 (1985), pp. 617–656.

[8] L. M. EWERBRING AND F. T. LUK, *Canonical correlations and generalized SVD: applications and new algorithms*, J. Comp. Appl. Math., 27 (1989), pp. 37–52.

[9] ———, *The HK singular value decomposition*, in Transaction of the 6th Army Conference on Applied Math., 1989, pp. 881–891.

	Error in singular value				
$m = 5$	1	2	3	4	5
LRCH	1.8×10^{-15}	8.9×10^{-16}	4.3×10^{-15}	1.1×10^{-13}	1.1×10^{-12}
LRCH/sh	1.8×10^{-15}	8.9×10^{-16}	4.3×10^{-15}	1.1×10^{-13}	1.1×10^{-12}
Jacobi	2.0×10^{-15}	1.3×10^{-15}	4.1×10^{-15}	1.1×10^{-13}	1.1×10^{-12}
$m = 10$	1	2	3	4	5
LRCH	4.1×10^{-15}	2.2×10^{-15}	7.7×10^{-15}	1.0×10^{-13}	1.6×10^{-12}
LRCH/sh	4.1×10^{-15}	2.2×10^{-15}	7.7×10^{-15}	1.0×10^{-13}	1.6×10^{-12}
Jacobi	3.9×10^{-15}	3.8×10^{-15}	7.3×10^{-15}	1.0×10^{-13}	1.6×10^{-12}
$m = 20$	1	2	3	4	5
LRCH	8.5×10^{-15}	8.2×10^{-15}	1.8×10^{-14}	2.7×10^{-13}	3.6×10^{-12}
LRCH/sh	8.5×10^{-15}	8.2×10^{-15}	1.8×10^{-14}	2.7×10^{-13}	3.6×10^{-12}
Jacobi	8.3×10^{-15}	1.1×10^{-14}	1.8×10^{-14}	2.7×10^{-13}	3.6×10^{-12}

$\Sigma = \Sigma_1$

	Error in singular value				
$m = 20$	1	2	3	4	5
LRCH	2.1×10^{-14}	3.7×10^{-14}	4.6×10^{-14}	6.2×10^{-14}	5.8×10^{-14}
LRCH/sh	4.7×10^{-15}	8.0×10^{-15}	3.6×10^{-15}	1.4×10^{-14}	1.8×10^{-15}
Jacobi	2.4×10^{-15}	4.4×10^{-15}	4.9×10^{-15}	4.8×10^{-15}	1.3×10^{-15}
$m = 40$	1	2	3	4	5
LRCH	1.8×10^{-15}	7.5×10^{-15}	4.8×10^{-15}	5.7×10^{-14}	4.2×10^{-14}
LRCH/sh	4.8×10^{-15}	1.0×10^{-14}	2.8×10^{-15}	1.9×10^{-14}	4.4×10^{-15}
Jacobi	3.1×10^{-15}	1.0×10^{-14}	6.0×10^{-15}	1.5×10^{-14}	1.8×10^{-15}
$m = 80$	1	2	3	4	5
LRCH	2.4×10^{-14}	1.1×10^{-13}	7.2×10^{-14}	1.0×10^{-13}	6.0×10^{-14}
LRCH/sh	4.4×10^{-15}	1.6×10^{-14}	1.3×10^{-15}	3.7×10^{-14}	1.7×10^{-14}
Jacobi	6.7×10^{-15}	1.9×10^{-14}	1.4×10^{-14}	2.8×10^{-14}	4.0×10^{-15}

$\Sigma = \Sigma_2$

TABLE 9.3
G.W. Stewart test examples: Relative errors in singular values

[10] ———, *The HK singular value decomposition of rank deficient matrix triplets*, in Computing in the 90's, Springer–Verlag, Berlin, New York, 1991, pp. 286–292. Lecture Notes in Computer Science Series, no. 507.

[11] V. FERNANDO AND S. HAMMARLING, *A product SVD (ΠSVD) for two matrices and balanced realizations*, in Linear Algebra in Signals, Systems and Control, B. N. Datta, eds., SIAM Publ., Philadelphia, PA, 1988, pp. 128–140.

[12] K. GEIST, U. PARLITZ, AND W. LAUTERSBORN, *Comparison of different methods for computing Lyapunov exponents*, Progr. Theoret. Phys., 83 (1990), pp. 875–893.

[13] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, 2nd ed., 1989.

[14] M. T. HEATH, A. J. LAUB, C. C. PAIGE, AND R. C. WARD, *Computing the singular value decomposition of a product of two matrices*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1147–1159.

[15] R. A. HORNE AND C. A. JOHNSON, *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[16] ———, *Topics in Matrix Analysis*, Cambridge University Press, Cambridge, 1991.

[17] E. N. LORENZ, *Deterministic non-periodic flow*, J. Atmospheric Sci., 20 (1963), pp. 130–141.

[18] C. MOLER, J. LITTLE, AND S. BANGERT, *Pro-MATLAB User's Guide*, The MathWorks Inc., South Natick, MA, 1990.

[19] V. I. OSELEDEC, *A multiplicative ergodic theorem: Ljapunov characteristic numbers for dynamical systems*, Trans. Moscow Math. Soc., 19 (1967), pp. 197–231.

[20] L. PERKO, *Differential Equations and Dynamical Systems*, Texts Appl. Maths. 7, Springer–Verlag, Berlin, Heidelberg, New York, 1991.

[21] H. RUTISHAUSER AND H. R. SCHWARTZ, *The LR transformation method for symmetric matrices*, Numer. Math., 5 (1963), pp. 273–289.
[22] D. E. STEWART AND Z. LEYK, *Meschach: Matrix Computations in C*, Australian National University, Canberra, 1994.
[23] G. W. STEWART, *On graded QR decompositions of products of matrices*, Elec. Trans. Numer. Anal., 3 (1995), pp. 39–49.
[24] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.
[25] H. Y. ZHA, *A numerical algorithm for computing the restricted singular value decomposition of matrix triplets*, Linear Algebra. Appl., 158 (1992), pp. 1–25.