University of Maryland | College Park

Institute for Advanced Computer Studies | TR–94–53
Department of Computer Science | TR–3263

On Graded QR Decompositions of Products of Matrices*

G. W. Stewart[†]

April 1994

ABSTRACT

This paper is concerned with the singular values and vectors of a product $M_m = A_1 A_2 \cdots A_m$ of matrices of order n. The chief difficulty with computing them from directly from M_m is that with increasing m the ratio of the small to the large singular values of M_m may fall below the rounding unit, so that the former are computed inaccurately. The solution proposed here is to compute recursively the factorization $M_m = QRP^{\mathrm{T}}$, where Q is orthogonal, R is a graded upper triangular, and P^{T} is a permutation.

*This report is available by anonymous ftp from `thales.cs.umd.edu` in the directory `pub/reports`.

[†]Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. This work was supported in part by The National Science Foundation under Grant CCR 9115568.

On Graded QR Decompositions of Products of Matrices

G. W. Stewart

ABSTRACT

This paper is concerned with the singular values and vectors of a product $M_m = A_1 A_2 \cdots A_m$ of matrices of order n. The chief difficulty with computing them from directly from M_m is that with increasing m the ratio of the small to the large singular values of M_m may fall below the rounding unit, so that the former are computed inaccurately. The solution proposed here is to compute recursively the factorization $M_m = QRP^{\mathrm{T}}$, where Q is orthogonal, R is a graded upper triangular, and P^{T} is a permutation.

1. Introduction

This paper is concerned with calculating the singular values of a product $M_m = A_1 A_2 \cdots A_m$ of matrices of order n. The chief difficulty with the natural algorithm — compute M_m and then compute its singular value decomposition — is that as m increases, the singular values will tend to spread out and the smaller singular values will be inaccurately computed. The rule of thumb is that singular values of M_m whose ratio to the largest singular value is less than the rounding unit will have no accuracy.

The product singular value decomposition (PSVD) is one way of circumventing this problem [1]. Here the product M_m is decomposed in the form

$$M_m = (UT_1Q_1^{\mathrm{T}})(Q_1T_2Q_2^{\mathrm{T}}) \cdots (Q_{m-1}T_mV^{\mathrm{T}}), \tag{1.1}$$

where U, V, and the Q_k are orthogonal, the T_k are triangular, and

$$\Sigma = T_1T_2 \cdots T_m$$

is diagonal. Thus the product (1.1) collapses into the singular value decomposition $U\Sigma V^{\mathrm{T}}$ of M_m.

The chief drawback of the PSVD is that it is expressed in terms of all the factors of M_m. This means that as m increases the storage required for the decomposition increases. Moreover, there seems to be no easy way to pass from the PSVD of M_m

to that of M_{m+1}; the multiplication by A_{m+1} changes the entire decomposition, so that the work required to append a factor also increases with m.

An alternative is an algorithm for computing the singular value decomposition of a product of two matrices [4]. Given the singular value decomposition of M_m, it can be used to calculate the singular value decomposition of $M_{m+1} = M_m A_{m+1}$. However, it is not clear how the increasing spread of the singular values affects this algorithm.

In this paper we take a different tack and work with decompositions of the form $A = QRP^{\mathrm{T}}$, where Q is orthogonal, R is upper triangular, and P is a permutation. The algorithm works to insure that R is a graded; that is, $R = D\hat{R}$, where the elements in the upper half of $\hat{R}$ are of order one and D is a diagonal matrix whose diagonal elements decrease in magnitude. The rule of thumb mentioned above — that the small singular values of a matrix are calculated inaccurately — need not hold for graded triangular matrices, so that very small singular values of M_m can usually be calculated from its graded factor R_m. Moreover, the product of graded triangular matrices is graded, so that in a decomposition analogous to (1.1), we can multiply out the triangular factors. Thus the passage from a graded QRP decomposition of M_m to one of M_{m+1} does not involve all the previous decompositions.

In the next section, we will show how to compute a graded QRP decomposition of a product AB from a graded QRP decomposition of A. In Section 3 we argue informally that the algorithm preserves grading. Here we also point out that good estimates of the singular values can be obtained with a small amount of additional work (the mathematical justification is given in Appendix A). In Section 4 we give numerical examples to show that this procedure can be applied recursively to compute singular values of a product of matrices whose singular values have ratios far below the rounding unit. Matlab code for the procedure is given in an appendix to this paper.

A final point. Although we have chosen to work with QRP decompositions in which P is a permutation, it will be clear that P could equally well be an orthogonal matrix. Thus the approach taken here also applies to two-sided orthogonal decompositions such as the URV decomposition [6].

2. The Algorithm

In this section we will describe the algorithm for updating graded QRP decompositions. The description will be in two stages: first an overview at the matrix level, then a detailed description. The latter is required to understand why the algorithm preserves grading. We will assume that the reader is familiar with the pivoted QR decomposition and plane rotations.

The input to the algorithm is a graded QRP decomposition

$$A = Q_A R_A P_A^{\mathrm{T}}$$

of a matrix A and an unfactored matrix B. The output is a graded QRP decomposition of

$$C \equiv AB = Q_C R_C P_C^{\mathrm{T}}. \tag{2.1}$$

The steps are as follows.

1. Compute the pivoted QR decomposition $P_A^{\mathrm{T}} B = Q_B R_B P_C^{\mathrm{T}}$
2. Compute an orthogonal matrix U such that $\hat{R}_A = U^{\mathrm{T}} R_A Q_B$ is upper triangular.
3. Set $Q_C = Q_A U$.
4. Set $R_C = \hat{R}_A R_B$.

It is easy to verify that the quantities so computed satisfy (2.1).

In some applications it is necessary to work with products of the form AB^{-1}. The above algorithm can be adapted to compute a graded QRP decomposition of AB^{-1}. The trick is to compute a graded PRQ decomposition of B, so that when the decomposition is inverted, it becomes a graded QRP decomposition of B^{-1}. This leads to the following algorithm.

1. Compute the pivoted QR decomposition $BP_A = P_C R_B Q_B^{\mathrm{T}}$
2. Compute an orthogonal matrix U such that $\hat{R}_A = U^{\mathrm{T}} R_A Q_B$ is upper triangular.
3. Set $Q_C = Q_A U$.
4. Set $R_C = \hat{R}_A R_B^{-1}$.

We will now consider the particulars. Since the two algorithms sketched above are variants of one another, we will treat only the first. The reader may find it helpful to consult the matlab program in Appendix B.

The first step in the above algorithm is accomplished with plane rotations. Specifically, rotations in the $(i, i+1)$-plane eliminate elements in $P^{\mathrm{T}} B$ in the

$$
\begin{array}{cccc}
 & & \downarrow & \downarrow \\
r & r & r & r \\
0 & r & r & r \\
0 & 0 & r & r \\
0 & 0 & 0 & r
\end{array}
\Longrightarrow
\begin{array}{ccccc}
 & r & r & r & r \\
 & 0 & r & r & r \\
\rightarrow & 0 & 0 & r & r \\
\rightarrow & 0 & 0 & \hat{r} & r
\end{array}
\Longrightarrow
\begin{array}{cccc}
 & \downarrow & \downarrow & \\
r & r & r & r \\
0 & r & r & r \\
0 & 0 & r & r \\
0 & 0 & 0 & r
\end{array}
\Longrightarrow
$$

$$
\begin{array}{ccccc}
 & r & r & r & r \\
\rightarrow & 0 & r & r & r \\
\rightarrow & 0 & \hat{r} & r & r \\
 & 0 & 0 & 0 & r
\end{array}
\Longrightarrow
\begin{array}{cccc}
\downarrow & \downarrow & & \\
r & r & r & r \\
0 & r & r & r \\
0 & 0 & r & r \\
0 & 0 & 0 & r
\end{array}
\Longrightarrow
\begin{array}{ccccc}
\rightarrow & r & r & r & r \\
\rightarrow & \hat{r} & r & r & r \\
 & 0 & 0 & r & r \\
 & 0 & 0 & 0 & r
\end{array}
\Longrightarrow
\begin{array}{cccc}
r & r & r & r \\
0 & r & r & r \\
0 & 0 & r & r \\
0 & 0 & 0 & r
\end{array}
$$

Figure 2.1: Reduction of R_A

order indicated below:

$$
\begin{pmatrix}
b & b & b & b & b \\
b^4 & b & b & b & b \\
b^3 & b^7 & b & b & b \\
b^2 & b^6 & b^9 & b & b \\
b^1 & b^5 & b^8 & b^{10} & b
\end{pmatrix}.
$$

Before the kth column is processed, the column in the trailing principal submatrix whose 2-norm is largest is located and the *entire* column is swapped with the kth. (The matlab code in the appendix inefficiently computes the norms *ab initio*. In practice, the norms should be downdated as in [2].)

In the second step, the rotations generated in the first are applied to R_A. When a rotation in the $(i, i+1)$ plane is postmultiplied into R_A, it creates a nonzero element in the $(i+1, i)$ position. This element is eliminated by premultiplying by a rotation in the $(i, i+1)$-plane. The process is illustrated in Figure 2.1. Here the arrows indicate the plane of the rotation to be applied and a hat indicates an element destined to be annihilated.

The rotations from the first step can be applied to R_A as soon as they are generated, which saves having to store them. Similarly the rotations that make up U can be accumulated in Q_A, so that at the end Q_A will be transformed into Q_C. Thus in the implementation steps one, two and three are completely interleaved.

When a product of the form $A_1 A_2 \cdots A_m$ is to be processed, the procedure can be started by setting $A = Q_A = P_A = I$ and $B = A_1$. In this case step two should be skipped and the rotations from step one accumulated in Q_A.

The algorithm is quite inexpensive. An operation count shows that it requires $5\frac{1}{6}n^3$ additions and $10\frac{1}{6}n^3$ multiplications. When n is large enough, scaled rotations [3, §5.1.13] can be used to reduce the number of multiplications to $5\frac{1}{6}n^3$. In this case, the algorithm can be compared with five matrix multiplications.

3. Grading

Grading enters into the algorithm in three places: in step one, where it is established, and in steps two and three, where it must be maintained. We will consider each step in turn.

In step one we have used pivoting on column norms to establish a grading. This is the simplest method in an arsenal of techniques designed to reveal the rank of a matrix. Although there is a famous counter-example for which the method fails [3, §5.5.7], in practice it works well. However, as we mentioned in the introduction, other rank revealing decompositions may be substituted for the QR decomposition obtained by pivoting on column norms.

Turning now to step two of the algorithm, let us see how grading in R_A can be lost. The transformation of the trailing 2×2 matrix is typical. Let it be written

$$\begin{pmatrix} \alpha & \beta \\ 0 & \delta \end{pmatrix}.$$

For definiteness we will suppose that $\sqrt{\alpha^2 + \beta^2} = 1$ and that δ is small, so that the matrix has a marked grading. After postmultiplying by the rotation from the reduction of $P^{\mathrm{T}}B$, we have

$$\begin{pmatrix} \bar{\alpha} & \bar{\beta} \\ \bar{\gamma} & \bar{\delta} \end{pmatrix},$$

where $\sqrt{\bar{\alpha}^2 + \bar{\beta}^2} = 1$ and $\sqrt{\bar{\gamma}^2 + \bar{\delta}^2} = |\delta|$. Thus the postmultiplication leaves the grading unaffected, at least normwise.

We must now premultiply by a plane rotation to reduce $\bar{\gamma}$ to zero. This rotation has the form

$$\frac{1}{\nu}\begin{pmatrix} \bar{\alpha} & \bar{\gamma} \\ -\bar{\gamma} & \bar{\alpha} \end{pmatrix},$$

where

$$\nu = \sqrt{\bar{\alpha}^2 + \bar{\gamma}^2}.$$

It follows that the $(2,2)$-element is

$$\hat{\epsilon} = \frac{\bar{\alpha}\bar{\delta} - \bar{\beta}\bar{\gamma}}{\nu},$$

whence

$$|\hat{\epsilon}| \le \sqrt{2}\frac{\epsilon}{\nu}. \tag{3.1}$$

Now the way grading is lost is for a large element in one row to overwhelm a small element in another row. The inequality (3.1) shows that this can happen only if ν is small or equivalently $\bar{\alpha}$ is small. But

$$\bar{\alpha} = c\alpha + s\beta,$$

where c and s are the cosine and sine from the reduction of $P_A^{\mathrm{T}}B$. Since these numbers are unrelated to α and β, it is extremely unlikely that α will be very much less than one.

Finally, consider the grading of the product $\hat{R}_A R_B$. Let

$$\hat{R}_A = \mathrm{diag}(\epsilon_1, \ldots, \epsilon_n)\hat{R} \quad \text{and} \quad R_B = \mathrm{diag}(\delta_1, \ldots, \delta_n)R,$$

where the elements of $\hat{R}$ and R are of order one in magnitude and the ϵ's and δ's are decreasing. Then the (i,j)-element of $\hat{R}_A R_B$ is

$$\delta_i\epsilon_i\Big(\hat{r}_{ii}r_{ij} + \sum_{k=i+1}^{j}\frac{\delta_k}{\delta_i}\hat{r}_{ik}r_{kj}\Big).$$

Since $\delta_k/\delta_i < 1$, we can expect the ith row of R_C to be about $\delta_i\epsilon_i$ in size.

It is hardly necessary to point out that these arguments are informal. Even the definition of a graded triangular matrix is too stringent: in practice we would expect the rows of a graded matrix to have occasional elements that are uncharacteristically small. Nonetheless, experience, folklore, and limited analytical results all suggest than the ills that can potentially beset graded matrices do not often occur in practice.

If the grading is sharp enough, we can get a cheap estimate of the singular values of a graded triangular matrix R. Let δ_1, δ_2, …, δ_n be the grading factors. Suppose that we have determined an orthogonal matrix V such that RV is lower triangular. Since postmultiplication by V does not change the norms of the rows of R, the elements of RV will be of the sizes indicated below for $n = 5$:

$$RV \cong \begin{pmatrix} \delta_1 & & & & \\ \delta_2 & \delta_2 & & & \\ \delta_3 & \delta_3 & \delta_3 & & \\ \delta_4 & \delta_4 & \delta_4 & \delta_4 & \\ \delta_5 & \delta_5 & \delta_5 & \delta_5 & \delta_5 \end{pmatrix} \tag{3.2}$$

Let $\rho_1 = 0$, $\rho_i = \delta_i/\delta_{i-1}$ $(i = 2, \ldots, n)$, and $\rho_{n+1} = 0$. Then if ρ_i and ρ_{i+1} are not too large, the ith diagonal element of RV approximates a singular value of R with relative error that is approximately bounded by $\frac{1}{2}(\rho_i^2 + \rho_{i+1}^2)$. (For more details see Appendix A.) Thus for about a third again the work (that is, the cost of computing RV) we can obtain estimates of the singular values, estimates that are often very accurate.

4. Numerical Results

The main problem with testing the algorithm is constructing test cases whose answers can be easily recognized. The solution taken here is an extension of an idea in [4].

The matrices U and V of left and right singular vectors are calculated from a random normal matrix, and a diagonal matrix Σ is chosen. Two matrices A and B are defined by

$$A = U\Sigma V^{\mathrm{T}} \quad \text{and} \qquad B = V\Sigma U^{\mathrm{T}}.$$

For a given m, a QRP decomposition of the product

$$M_{2m+1} = A(BA)^m$$

is calculated. The singular value decomposition of M_{2m+1} is

$$M_{2m+1} = U\Sigma^{2m+1}V^{\mathrm{T}},$$

so that a correct answer can easily be recognized.

The first test, in which $\Sigma = \mathrm{diag}(1, 0.1, 0.01, 0.001, 0.0001)$, is designed to push the singular values toward the underflow point. The following table contains the singular values of M_{2m+1} and their relative errors for three values of m.

```
m=5
   1.0e+00   1.0e-11   1.0e-22   1.0e-33   1.0e-44
   3.9e-15   1.1e-14   1.1e-14   4.0e-14   6.3e-13
m=10
   1.0e+00   1.0e-21   1.0e-42   1.0e-63   1.0e-84
   7.4e-15   2.0e-14   2.1e-14   6.2e-14   1.3e-12
m=20
   1.0e+00   1.0e-41   1.0e-82   1.0-123   1.0-164
   1.4e-14   3.9e-14   4.1e-14   1.0e-13   2.6e-12
```

For $m = 20$ the smallest singular value decreases by 160 orders of magnitude, yet it is computed with almost full accuracy. Incidentally, most of the inaccuracy in this singular value is not due to the algorithm at all, but to the fact that the singular value 0.0001 is represented inaccurately in the original matrix.

The second test takes longer run down a gentler grade. In this case $\Sigma = \mathrm{diag}(1, 0.99, 0.8, 0.7, 0.6)$. The first two rows of the following results are as above.

```
m=20
   1.0e+00   6.6e-01   1.3e-02   1.0e-04   4.4e-07
   1.3e-14   4.6e-15   1.8e-14   4.0e-15   6.5e-15
   9.1e-01   7.2e-01   1.3e-02   1.0e-04   4.4e-07
   9.3e-02   8.5e-02   7.7e-05   5.0e-06   4.4e-07
m=40
   1.0e+00   4.4e-01   1.9e-04   1.4e-08   2.8e-13
   2.5e-14   8.6e-15   3.8e-14   7.0e-15   1.3e-14
   9.3e-01   4.7e-01   1.9e-04   1.4e-08   2.8e-13
   6.8e-02   6.4e-02   3.6e-08   4.3e-10   1.0e-11
m=80
   1.0e+00   1.9e-01   4.2e-08   2.4e-16   1.1e-25
   4.8e-14   1.8e-14   7.1e-14   1.5e-14   2.7e-14
   9.8e-01   2.0e-01   4.2e-08   2.4e-16   1.1e-25
   1.7e-02   1.7e-02   7.9e-14   1.5e-14   2.7e-14
```

Again the algorithm computes the singular values with almost full accuracy. Rounding error accumulates, but very slowly.

The third row contains the diagonal elements of RV [see (3.2)], which approximate the singular values of the product, and the fourth row contains their relative errors. For $m = 20$, the last three singular values are well approximated (note how it is the square of the grading that governs the accuracy). There is even useful information about the magnitudes of the first two singular values. The approximations become more accurate as m increases.

Finally to see how the algorithm performs on a matrix of moderate size with a natural distribution of singular values, A was taken to be a random normal matrix of order 50 with B generated as described above. The algorithm was run for $m = 2$. The results are too voluminous to present in their entirity, but the data for the smallest six singular values illustrates what is happening.

2.9e+00	6.7e-01	1.5e-01	3.7e-02	2.7e-04	1.2e-07
7.4e-15	5.0e-15	1.0e-15	1.1e-14	1.2e-14	1.0e-15
3.0e+00	5.4e-01	1.8e-01	3.8e-02	2.7e-04	1.2e-07
5.9e-02	2.3e-01	1.8e-01	4.7e-03	2.5e-08	7.0e-08

The singular values of the computed product are quite accurate. The approximations give one or two figures — more when the local grading is strong.

5. Conclusions

The main recommendation for this algorithm is its simplicity and efficiency. The matlab code attests its simplicity. Its efficiency is attested by the operation counts, though not, unfortunately, its running time in matlab. The only question is: does it work? Here we have only been able to give an informal analysis and limited examples, either of which alone might be considered insufficient. Together, I believe, they make a strong case for the algorithm.

References

[1] A. Bojanczyk, J. G. Nagy, and R. J. Plemmons. Block RLS using row Householder reflections. *Linear Algebra and Its Applications*, 188–189:31–62, 1993.

[2] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

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

[4] M. T. Heath, A. J. Laub, C. C. Paige, and R. C. Ward. Computing the SVD of a product of two matrices. *SIAM Journal on Scientific and Statistical Computing*, 7:1147–1159, 1986.

[5] R. Mathias and G. W. Stewart. A block qr algorithm and the singular value decomposition. *Linear Algebra and Its Applications*, 182:91–100, 1993.

[6] G. W. Stewart. An updating algorithm for subspace tracking. *IEEE Transactions on Signal Processing*, 40:1535–1541, 1992.

Appendix A: On the Singular Values of Graded Matrices

In this appendix we show that the singular values of a matrix with the structure (3.2) are approximated by the diagonal elements of the matrix. We will use the following theorem of Mathias and Stewart [5] (which we have weakened slightly for the sake of simplicity). Here $\|A\|$ denotes the spectral norm of A and $\inf(A)$ is the smallest singular value of A.

Let

$$T = \begin{array}{cc} & \begin{array}{cc} k & n-k \end{array} \\ \begin{array}{c} k \\ n-k \end{array} & \begin{pmatrix} P & H \\ 0 & Q \end{pmatrix} \end{array}$$

be a block triangular matrix, and suppose that $\|Q\|/\inf(P), \|H\|/\inf(P) < 1$. *Let* $\sigma_1 \geq \cdots \geq \sigma_n$ *be the singular values of* T *and* $\hat{\sigma}_1 \geq \cdots \geq \hat{\sigma}_n$ *be the singular values of* $\mathrm{diag}(P, Q)$. *Then*

$$1 \geq \frac{\hat{\sigma}_i}{\sigma_i} \geq (1-\tau^2)^{\frac{1}{2}}, \qquad i = 1, 2, \ldots, k,$$

and

$$1 \geq \frac{\sigma_i}{\hat{\sigma}_i} \geq (1-\tau^2)^{\frac{1}{2}}, \qquad i = k+1, k+2, \ldots, n,$$

where

$$\tau^2 = \frac{(\|H\|/\inf(P))^2}{1-(\|Q\|/\inf(P))^2}$$

The theorem states that approximations of the singular values of T with high relative accuracy can be found in the diagonal blocks of P and Q.

To apply the theorem, let the matrix $L = RV$ of (3.2) be written in the form

$$L = \mathrm{diag}(\delta_1, \delta_2, \ldots, \delta_n)\hat{L},$$

where the rows of $\hat{L}$ have norm one. Partition L in the form

$$\begin{array}{cc} & \begin{array}{ccc} k-1 & 1 & n-k \end{array} \\ \begin{array}{c} k-1 \\ 1 \\ n-k \end{array} & \begin{pmatrix} L_{11} & 0 & 0 \\ \ell_{21}^{\mathrm{T}} & \lambda_{22} & 0 \\ L_{31} & \ell_{32} & L_{33} \end{pmatrix} \end{array}$$

We wish to assess the accuracy of λ_{22} as a singular value of L. The approach is to apply the above theorem twice.

In the first application we set $P = L_{11}$, which amounts to throwing out ℓ_{21}^{T} and L_{31}. In this case

$$\inf(P) = \inf(L_{11}) = \inf(\operatorname{diag}(\delta_1, \ldots, \delta_{k-1})\hat{L}_{11}) \geq \delta_{k-1} \inf(L_{11}).$$

Moreover, since the rows of $\hat{L}$ have norm one,

$$\|H\|, \|Q\| \leq \delta_k \sqrt{1 + (\delta_{k+1}/\delta_k)^2 + \cdots (\delta_n/\delta_k)^2}.$$

Consequently if we set

$$\rho_{1k} = \frac{\delta_k \sqrt{1 + (\delta_{k+1}/\delta_k)^2 + \cdots (\delta_n/\delta_k)^2}}{\delta_{k-1} \inf(L_{11})} \equiv \rho_k \frac{\sqrt{1 + (\delta_{k+1}/\delta_k)^2 + \cdots (\delta_n/\delta_k)^2}}{\inf(L_{11})}, \tag{A.1}$$

then

$$\tau_1^2 = \frac{\rho_{1k}^2}{1 - \rho_{1k}^2}$$

bounds the perturbation in the singular values.

The second step is to take $P = \lambda_{22}$ in the matrix

$$\begin{pmatrix} \lambda_{22} & 0 \\ \ell_{32} & L_{33}, \end{pmatrix}.$$

which amounts to setting ℓ_{32} to zero, leaving λ_{22} as a singular value of the perturbed matrix. Here $\inf(P) = |\lambda_{22}|$. Moreover,

$$\|H\|, \|Q\| \leq \delta_{k+1} \sqrt{1 + (\delta_{k+2}/\delta_{k+1})^2 + \cdots (\delta_n/\delta_{k+1})^2}.$$

Consequently, if we set

$$\rho_{2,k+1} = \rho_{k+1} \frac{\sqrt{1 + (\delta_{k+2}/\delta_{k+1})^2 + \cdots (\delta_n/\delta_{k+1})^2}}{|\lambda_{kk}|/\delta_k}$$

then

$$\tau_2^2 = \frac{\rho_{2,k+1}^2}{1 - \rho_{2,k+1}^2}$$

bounds the effect of the second perturbation.

For small ρ_{1k} and $\rho_{2,k+1}$, the total bound on the relative perturbation is approximately $(\rho_{1k}^2 + \rho_{2,k+1}^2)/2$. Note that in (A.1) we have written ρ_{1k} as the product of ρ_k with a magnification factor that in general will be greater than one. If the grading is strong the denominator $\sqrt{1 + (\delta_{k+1}/\delta_k)^2 + \cdots (\delta_n/\delta_k)^2}$ will be very near one. In our application, the pivoting in the algorithm tends to keep $\inf(L_{11})$ from becoming very small. Hence it is the ratio $\rho_k = \delta_k/\delta_{k-1}$ that controls the accuracy of the kth diagonal element as an approximate singular value. Similar comments can be made about $\rho_{2,k+1}$.

Appendix B: Program Listings

```
function [rc, qc, pc] = prodqrp(ra, qa, pa, b, first)
%
%  Given a qrp factorization of a matrix a and a matrix b
%  prodqrp computes a qrp factorization of a*b.
%
%  Designed and coded by G. W. Stewart, April 8, 1994.
%
   n = size(ra, 1);
   qc = qa;
%
%  Accumulate the permutation pa in b.
%
   for i=1:n-1
      b([i,pa(i)],:) = b([pa(i),i],:);
   end
%
%  Compute a qrp factorization of b and update.
%
   for k=1:n-1
      for j=k:n, nrm(j)=norm(b(k:n,j)); end
      [xx, pc(k)] = max(nrm(k:n));
      pc(k) = pc(k)+k-1;
      b(1:n, [k,pc(k)]) = b(1:n, [pc(k),k]);
      for i=n-1:-1:k
         [c, s, nu] = rotgen(b(i,k), b(i+1,k));
         b(i,k) = nu; b(i+1,k) = 0.;
         b(i:i+1, k+1:n) = [c, s; -s, c]*b(i:i+1, k+1:n);
         if (first == 0)
            ra(1:i+1, i:i+1) = ra(1:i+1, i:i+1)*[c, -s; s, c];
            [c, s, nu] = rotgen(ra(i,i),ra(i+1,i));
            ra(i,i) = nu; ra(i+1,i) = 0;
            ra(i:i+1, i+1:n) = [c , s; -s, c]*ra(i:i+1, i+1:n);
         end
         qc(i:i+1, 1:n) = [c, s; -s, c]*qc(i:i+1, 1:n);
      end
   end
   rc = ra*b;
```

```
function [c, s, norm] = rotgen(a,b)
%
%  This function generates a plane rotation.
%
   scale = abs(a) + abs(b);
   if scale == 0
      c = 1; s = 0; norm = 0;
   else
      norm = scale*sqrt((a/scale)^2+(b/scale)^2);
      c = a/norm;
      s = b/norm;
   end
```