ONCOMPUTINGTHE SINGULAR VALUE DECOMPOSITION

by

Tony Fan C. Chan

STAN-CS-77-588
FEBRUARY 1977

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

LELAND STANFORD JUNIOR UNIVERSITY
ORGANIZED 1891

ON COMPUTING THE SINGULAR VALUE DECOMPOSITION

by

Tony Fan C. CHAN*

August, 1976.

*Computer Science Dept., Stanford Univ., Ca94305.

This work was supported by NSF Grant DCR75-13497 and NASA Ames Contract NCA2-OR745-520. The computing time was provided by the Stanford Linear Accelerator Center (SLAC).

ABSTRACT

The most well-known and widely-used algorithm for computing the Singular Value Decomposition (SVD) of an m x n rectangular matrix A nowadays is the Golub-Reinsch algorithm [1]. In this **paper,** it is shown that by (1) first triangularizing the matrix A by Householder transformations before bidiagonalizing it, and (2) accumulating some left transformations on an n x n array instead of on an m x n array, the resulting algorithm is often more efficient than the Golub-Reinsch algorithm, especially for matrices with considerably more rows than columns ($m >> n$), such as in least squares applications. The two algorithms are compared in terms of operation counts, and computational experiments that have been carried out verify the theoretical comparisons. The modified algorithm is more efficient even when m is only slightly greater than n, and in some cases can achieve as much as 50% savings when $m >> n$. If accumulation of left transformations is desired, then n^2 extra storage locations are required (relatively small if $m >> n$), but otherwise no extra storage is required. The modified algorithm uses only orthogonal transformations and is therefore numerically stable. In the Appendix, we give the FORTRAN code of a hybrid method which automatically selects the more **effiecient** of the two algorithms to use depending upon the input values for m and n.

(0) INTRODUCTION

Let A be a real m x n matrix, with m >> n. It is well-known [1,2] that the following decomposition of A always exists :

$$A - u \Sigma V^T \qquad (0.1)$$

where U is a m x n matrix and consists of n orthonormalized eigenvectors associated with the n largest eigenvalues of AA^T, V is a n x n matrix and consists of the orthonormalized eigenvectors of A^TA, and Σ is a diagonal matrix consisting of the "singular values" of A, which are the non-negative square roots of the eigenvalues of A^TA.

Thus,

$$U^TU = V^TV = VV^T = I_n$$

and $\Sigma = \mathrm{diag}(\sigma_1, \ldots\ldots \sigma_n).$ (0.2)

It is usually assumed for convenience that

$$\sigma_1 >= \sigma_2 " \ldots\ldots >= \sigma_n >= 0.$$

The decomposition (0.1) is called the <u>Singular Value Decomposition</u> (SVD) of A.

Remarks:

(1) If rank(A) = r, then $\sigma_{r+1} = \sigma_{r+2} = \ldots\ldots = \sigma_n = 0$.

(2) There is no loss of generality in assuming that m >= n, for if m < n, then we can instead compute the SVD of A^T. If the SVD of A^T is equal to $U\Sigma V^T$, then the SVD of A is equal to $V\Sigma U^T$.

The SVD plays a very important role in linear algebra. It has applications in such areas as least squares problems [1,2,3], in computing the pseudo-inverse [2], in computing the Jordan Canonical form [4], in solving integral equations [5], in digital image processing [6], and in optimization [7]. Many of the applications often involve large matrices. It is therefore important that the computational procedures for obtaining the SVD be as efficient as possible.

It is perhaps difficult to find an algorithm that has optimal efficiency for all matrices, so it would be desirable to know for what kind of matrices a given algorithm is best suited. It is in this spirit that we were first motivated to look for improvements of the Golub-Reinsch algorithm when the mat-rix A has considerably more rows than columns, i.e. m >> n. It turns out that such an improvement is indeed possible, with only slight modifications to the Golub-Reinsch algorithm, even when m is only slightly greater than n, and can sometimes achieve as much as 50% savings in execution time when m >> n.

In section (1) we will briefly describe the Golub-Reinsch algorithm. We will then present the modified algorithm in section (2), with some computational details deferred to section (3). Operation counts for the two algorithms will be given in section (4) and some computational results in section (5). We wili make some conclusions in section (6). In the Appendix, we will give the FORTRAN implementation of a hybrid method which automatically selects the more efficient of the two algorithms to use depending upon the input values for m and n.

(1) THE GOLUB-REINSCH ALGORITHM (GR-SVD)

We will use the same notation as in [1].

This algorithm consists of two phases. In the first phase one constructs two finite sequences of Householder transformations

$$P^{(k)} \qquad (k-1,2,\ .\ .\ .\ ,n)$$

$$\text{and} \qquad Q^{(k)} \qquad (k=1,2,\ .\ .\ ..n-2)$$

such that

$$P^{(n)} \ldots P^{(1)}\, A\, Q^{(1)} \ldots\ldots Q^{(n-2)} = \left[\begin{array}{c} \begin{matrix} \times & \times & & 0 \\ & \ddots & \ddots & \\ & & \ddots & \times \\ 0 & & & \times \end{matrix} \\ \hline 0 \end{array}\right] \begin{matrix} = J^{(0)}, \\ \left.\right\} (m-n)\times n \end{matrix}$$

an upper bidiagonal matrix. Specifically, $P^{(i)}$ zeros out the subdiagonal elements in column i and $Q^{(j)}$ zeros out the appropiate elements in row j.

The singular values of $J^{(0)}$ are the same as those of A. Thus,

if $J = G\Sigma H^T$ is the SVD of J,

then $A = P\ G\Sigma H^T Q^T$

so that $U = P\ G,\quad V = Q\ H$ (1.2)

with $P - P^{(1)} \ldots . P^{(n)},\ Q = Q^{(1)} \ldots . Q^{(n-2)}.$

The second phase is to iteratively diagonalize $J^{(0)}$ by the QR method so that

$$J^{(0)} \to J^{(1)} \to \ldots \to \Sigma$$

where $$J^{(i+1)} = S^{(i)T} J^{(i)} T^{(i)} , \qquad (1.3)$$

where $S^{(i)}$ and $T^{(i)}$ are products of Givens transformations and are therefore orthogonal.

The matrices $T^{(i)}$ are chosen so that the sequence $M^{(i)} = J^{(i)T} J^{(i)}$ converges to a diagonal matrix while the matrices $S^{(i)}$ are chosen so that all $J^{(i)}$ are of bidiagonal form.
The products of the $T^{(i)}$'s and the $S^{(i)}$'s are exactly the matrices H^T and G^T respectively in Eqn (1.2). For more details, see [1].

It has been reported in [1] that the average number of iterations on $J^{(i)}$ in (1.3) is usually less than 2n. In other words, $J^{(2n)}$ in Eqn (1.3) is usually a good approximation to a diagonal matrix.

We will briefly describe how the computation is usually implemented. Assume for simplicity, that we can destroy A and return U in the storage for A. In the first phase, the $P^{(i)}$ are stored in the lower part of A, and the $Q^{(i)}$ are stored in the upper triangular part of A. After the bidiagonalization, the $Q^{(i)}$ are accumulated in the storage provided for V, the two diagonals of $J^{(0)}$ are copied to two other linear arrays, and the $P^{(i)}$ are accumulated in A.

In the second phase, for each i,

$S^{(i)}$ is applied to P from the right and

${T^{(i)}}^T$ is applied to Q^T from the left

in order to accumulate the transformations.

(2) THE MODIFIED ALGORITHM (MOD-SVD)

Our original motivation for this algorithm is to find an improvement of GR-SVD when $m \gg n$. In that case, two improvements are possible:

(i) In Eqn (1.1), each of the transformations $P^{(i)}$ and $Q^{(i)}$ has to be applied to a submatrix of size $(m-i+1) \times (n-i+1)$.

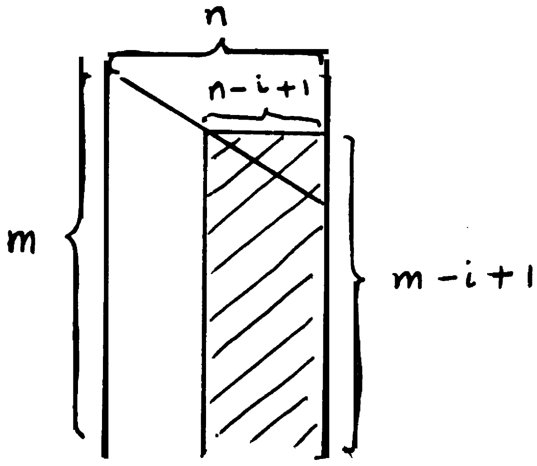
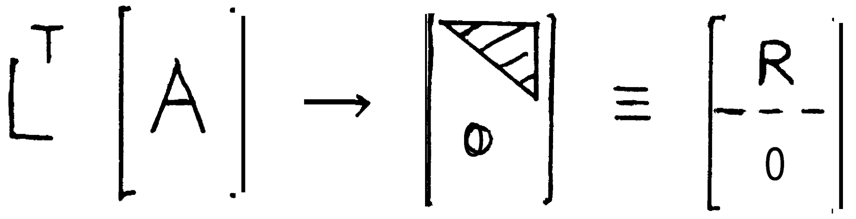


Fig. 2.1 $P^{(i)}$ and $Q^{(i)}$ affects the shaded portion of the matrix

Now, since most entries of this submatrix are ultimately going to be zeros, it is intuitive that if it can somehow be arranged that the $Q^{(i)}$ does not have to be applied to the subdiagonal part of this submatrix, then we will be saving a great amount of work when $m \gg n$.

This can indeed be done by first transforming A into upper triangular form by Householder transformations on the left.



where R is n x n upper triangular and L is orthogonal, and then proceed to bidiagonalize R. The important difference is that this time we will be working with a much smaller matrix R than A (if $n^2 << mn$), and so it is conceivable that the work required to bidiagonalize R is much smaller than that originally done by the right transformations when m >> n.

The question still remains as to how to bidiagonalize R. An obvious way is to treat R as an input matrix to GR-SVD, using alternating left and right Householder transformations. In fact, it can be easily verified that if the SVD of R is equal to $X \Sigma Y^T$, then the SVD of A is given by

$$A = L\begin{bmatrix} X \\ \hline 0 \end{bmatrix} \Sigma Y^T \qquad (2.1)$$

We can identify U with $L\begin{bmatrix} X \\ \hline 0 \end{bmatrix}$ and V with Y. Notice that in order to obtain U, we have to form the extra product $L\begin{bmatrix} X \\ \hline 0 \end{bmatrix}$. If U is not needed (e.g. in least squares), then we do not have to accumulate any left transformations and in that case, for m >> n, it seems likely that we will make a substantial saving.

It is also possible to take advantage of the structure of R to bidiagonalize it. This will be discussed in section (3).

(ii) The second improvement over GR-SVD that can be made is the following. In GR-SVD, each of the $S^{(i)}$ is applied to the m x n matrix P from the right to accumulate U. If $m >> n$, then this accumulation involves a large amount of work because a single Givens transformation affects two columns of P (of length m) and each $S^{(i)}$ is the product of on the average n/2 Givens transformations. Therefore, in such cases, it would seem more efficient to first accumulate all $S^{(i)}$ on an n x n array Z and later form the matrix product PZ after $J^{(i)}$ has converged to Σ.

In essence, improvement (i) works best when U is not needed, improvement (ii) works best when U is needed and both work best when $m >> n$.

We now present the modified algorithm:

MOD-SVD:

(1) $L^T\begin{bmatrix}A\end{bmatrix} \rightarrow \begin{bmatrix}R\\ \hline 0\end{bmatrix}$ where R is n x n upper triangular,

(2) Find the SVD of R by GR-SVD, $R = X\Sigma Y^T$,

(3) Form $A = L\begin{bmatrix}X\\ \hline 0\end{bmatrix}\Sigma Y^T$, the SVD of A.

The idea of transforming A to upper triangular form when m >> n and then calculating the SVD of R is mentioned in Lawson & Hanson [3,pp.119,122] in the context of least squares problems where U is not explicitly required.

In the next section we will discuss some computational details of this modified algorithm, and in section (4) we will compare the operation counts of the two algorithms.

(3) SOME COMPUTATIONAL DETAILS

(i) It should be obvious that when U is not needed then MOD-SVD does not require any extra storage. When U is needed, we can store L^T in the lower part of A, copy R into another n x n array W and ask GR-SVD to return X in W. Therefore we need at most n^2 extra storage locations which is relatively small when $m >> n$.

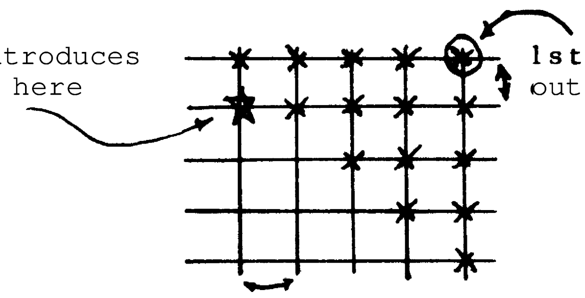
(ii) The next question is how to form $I\begin{bmatrix} X \\ \hline 0 \end{bmatrix}$ without using extra storage. This can be done by noting that

$$L\begin{bmatrix} X \\ \hline 0 \end{bmatrix} = L\begin{bmatrix} I \\ \hline 0 \end{bmatrix} X$$

so we can first accumulate $L\begin{bmatrix} I \\ \hline 0 \end{bmatrix}$ in the space provided for U and then do a matrix multiplication by X.

In the experiments that we have carried out, we actually accumulate the Householder transformations L on $\begin{bmatrix} X \\ \hline 0 \end{bmatrix}$.
We do not recommend doing this in practice because it requires mn instead of n^2 extra storage locations. But one can show that both methods take about the same amount of work and so it will not affect the comparisons.

(iii) The question arises whether it is possible to bidiagonalize R in a way that takes advantage of the zeros that are already in R. One way is to use **Givens** transformations to zero out the elements at the upper right hand corner of R, one column or one row at a time. **Pictorially,** (for n-5) to zero out the **(1,5)** element, we do two Givens transformations as follows:


1st rotation introduces **nonzero** element here
1st rotation to zero out the **(1,5)** element
2nd **rotation to** zero out the (2,1j element introduced by the **1st** rotation

It turns out however, by simple counting, that this method takes about the same operations ($4n^3/3$ multiplications) as the previous method to bidiagonalize R, provided that we do not have to accumulate transformations. If we do need to accumulate either the left or the right transformations, then this method will require more work ($4n^3$ versus $4n^3/3$ mult.) mainly because it requires two rotations to zero out each element and these rotations have to be accumulated.
So it seems that taking advantage of the zero structure of R in this fashion actually makes the method less efficient.

We have to note, however, that Givens transformations involve fewer additions and array accesses than Householder transformations per multiplication (see section 4.1). Therefore this method tends to be more competitive on modern computers where the time taken for floating point additions and multi-dimensional array indexings are not negligible compared to that for multiplications.

There may be other ways to bidiagonalize R using orthogonal transformations, but we shall not pursue this subject further.

(4) OPERATION COUNTS

In section (2), we indicated that MOD-SVD should be more efficient than CR-SVD when $m >> n$. In this section, we study the relative efficiency between CR-SVD and MOD-SVD as a function of m and n. We do this by computing the asymptotic operation counts for each algorithm.

In the operation counts given below, we only keep the highest order terms in m and n, and so the results are correct for relatively iarge m and n.

CR-SVD:

(1) **Bidiagonalization** (using Householder transformations)

$J = P^{(n)} \ldots P^{(1)} A Q^{(1)} \ldots Q^{(n-2)}$	$2(mn^2 - n^3/3)$	mult.
accumulate $P = P^{(1)} \ldots P^{(n)}$	$mn^2 - n^3/3$	mult.
accumulate $Q = Q^{(1)} \ldots Q^{(n-2)}$	$2n^3$	mult.

(2) Diagonalization (using Givens **transformations**)

accumulate $S^{(i)}$ on P	Cmn^2 $(C=4)$	mult.
accumulate $T^{(i)}$ on Q	Cn^3 $(C=4)$	mult.

MOD-SVD:

(1) Triangularization (using Householder transformations)

$L^T \begin{bmatrix} A \end{bmatrix} \rightarrow \begin{bmatrix} R \\ \hline 0 \end{bmatrix}$	$mn^2 - n^3/3$	mult.
(2) CR-SVD of R, $R = X \Sigma Y^T$	depends on whether accumulations are needed.	
(3) **Form** $L \begin{bmatrix} X \\ \hline 0 \end{bmatrix}$ (using Householder **transf.**)	$2mn^2 - n^3$	mult.

Some comments are in order:

(0) The entries Cmn^2 and Cn^3 with C-4 in the diagonalization phase of CR-SVD are obtained by assuming that the iterative phase of the SVD takes on the average two complete QR iterations per singular value [1], [3,p122]. We have checked this experimentally and found it to be quite accurate.
It is assumed that slow Givens is used throughout the calculation. If fast Givens [8] had been used, then the entries would become approximately $2mn^2$ and $2n^3$ instead (viz C-2).

(1) For the Householder transformations, each multiplication also invokes 1 addition and approximately 2 array addressings. For the Givens transformations, each multiplication invokes 1/2 an addition and 1 array addressing. On many large computers today, a floating point multiplication is not much slower than a floating point addition. Also, array indexing is usually quite expensive. In such cases, a Householder multiplication actually involves more work than a Givens multiplication because of the extra additions and array indexings. Therefore, the operation counts given for the diagonalization phase of GR-SVD may be misleading because it may actually involve relatively less work. The total effect, however, can be accounted for by using a smaller value for C. For example, if 1 Givens "multiplication" takes half the work needed by a Householder "multiplication", then the effect on the relative efficiency can be accounted for by

setting C-2 instead of C-4. On older or non-scientific machines where multiplications take much more time than additions and array addressings, the operation count based on multiplications alone is usually a good measure of relative efficiency.

(2) The application of $S^{(i)^T}$ and $T^{(i)}$ on $J^{(i)}$ is actually of order $O(n^2)$ and is therefore not included in the above counts.

(3) We have to distinguish between 4 cases in the comparison:

Case a: both U and V are required explicitly,
Case b: only U is required explicitly,
case c: only V is required explicitly,
Case d: only Σ is required explicitly.

These four cases do arise in applications. We will mention a few here:

Case a arises in the computation of pseudo-inverses [1].
Case b is Case c for A^T.
Case c arises in least squares applications [1,3] and in the solution of homogeneous linear equations [1].
Case d arises in the estimation of the condition number of a matrix and in the determination of the rank of a matrix [10].

The total operation count for each case is given in Table 4.1 .

Table 4.1

Total operation counts of GR-SVD and MOD-SVD for each of the cases a, b, c, and d.

Case	GR-SVD	MOD-SVD
a	$(3+C)mn^2 + (C-1/3)*3$	$3mn^2 + (2C+4/3)n^3$
b	$(3+C)mn^2 - n^3$	$3mn^2 + (C+2/3)n^3$
c	$2mn^2 + Cn^3$	$mn^2 + (C+5/3)n^3$
d	$2mn^2 - 2n^3/3$	$mn^2 + n^3$

Using Table 4.1 , we can compute the ratio of the operation counts of MOD-SVD to that of GR-SVD for each of the four cases. This is given in Table 4.2 where the ratio is expressed as a function of $r = m/n$.

Table 4.2

Ratio of operation count of MOD-SVD to that of GR-SVD.

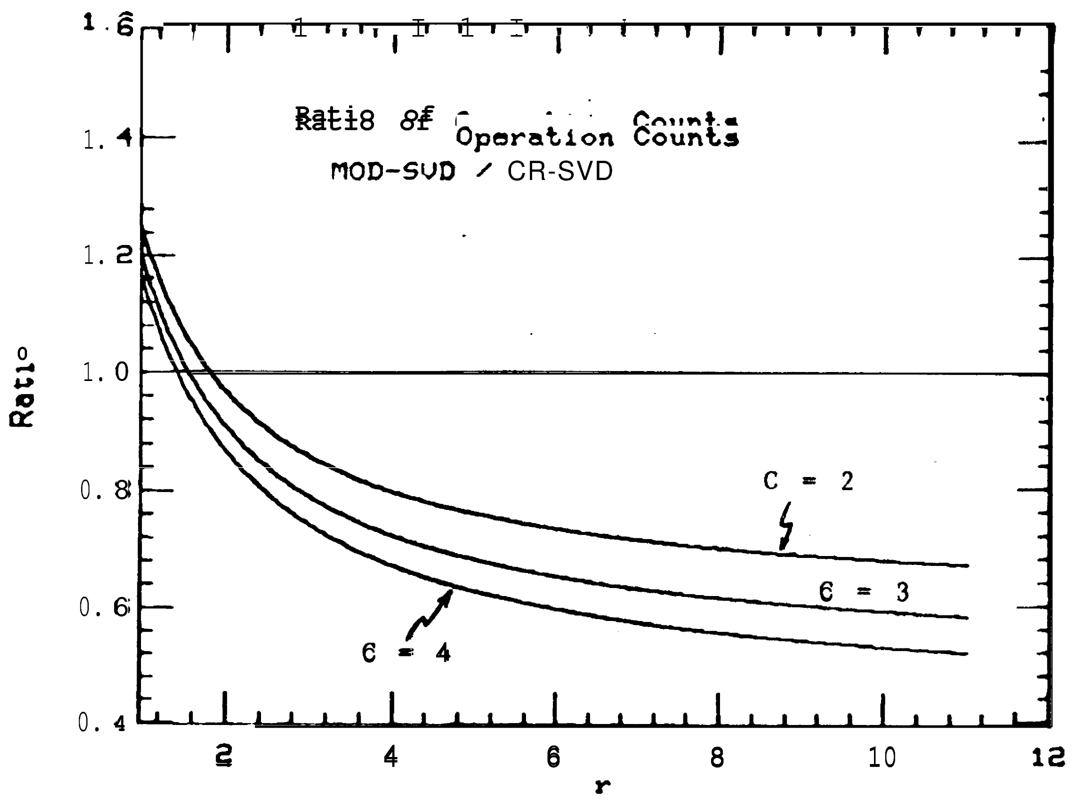
$r = m/n$

Case	Ratio	Cross-over point
a	$[3r+(2C+4/3)]/[(3+C)r+(C-1/3)]$	$(C+5/3)/C$
b	$[3r+(C+2/3)]/[(3+C)r-1]$	$(C+5/3)/C$
c	$[r+(C+5/3)]/[2r+C]$	$5/3$
d	$[r+1]/[2r-2/3]$	$5/3$

These ratios are plotted in Fig. 4.1 to Fig. 4.4 for $C=2,3,4$. The cross-over point $\mathbf{r}^*$ is the value of r which makes the ratio equal to 1. If $r > r^*$, then MOD-SVD is more efficient than CR-SVD.

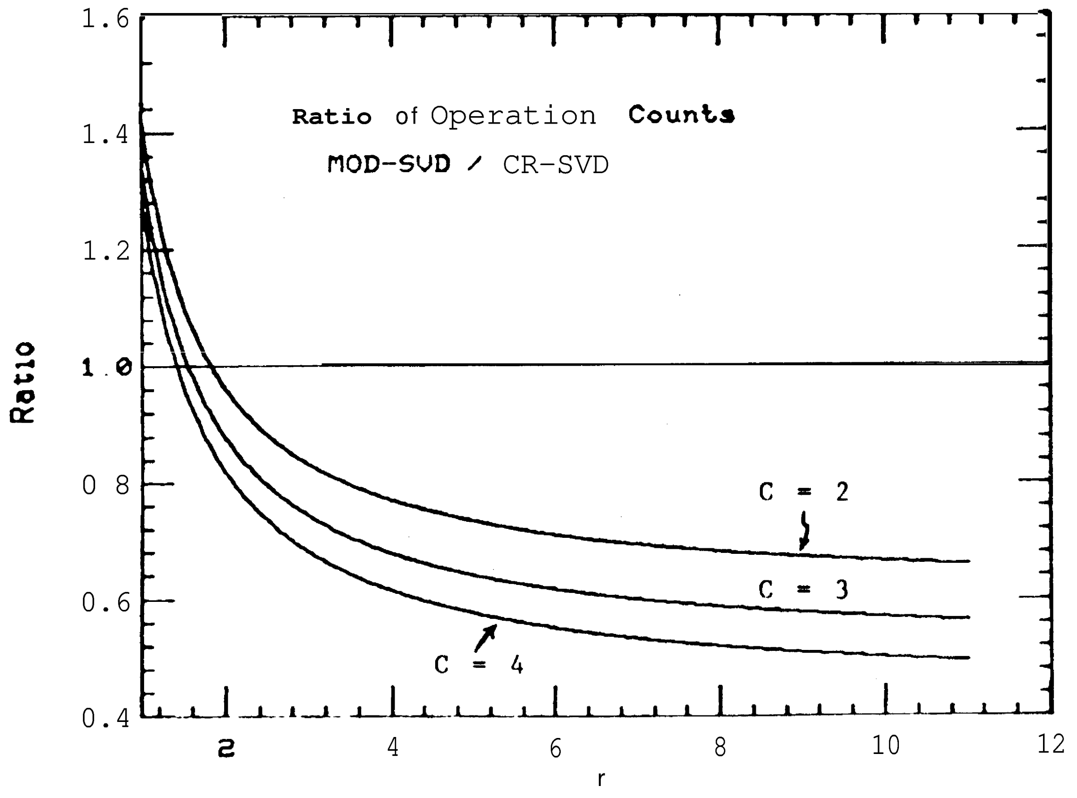
From Figures 4.1 - 4.4, we see that, in all 4 cases **a,b,c** and d, MOD-SVD becomes more efficient than CR-SVD when r starts to get bigger than 2 approximately, and the savings can be as much as 50% when r is about 10. On the other hand, when r is about 1, CR-SVD is more efficient. This agrees with our eariier conjectures. However, the important

Fig. 4.1 Case a



Ratio of Operation Counts
MOD-SVD / CR-SVD
C = 2
C = 3
C = 4
Ratio
r
1.6
1.4
1.2
1.0
0.8
0.6
0.4
2
4
6
8
10
12

Fig. 4.2 Case b



Ratio of Operation Counts
MOD-SVD / CR-SVD
1.6
1.4
1.2
1.0
0 8
0.6
0.4
Ratio
2
4
6
8
10
12
r
C = 2
C = 3
C = 4

Fig. 4.3 Case c

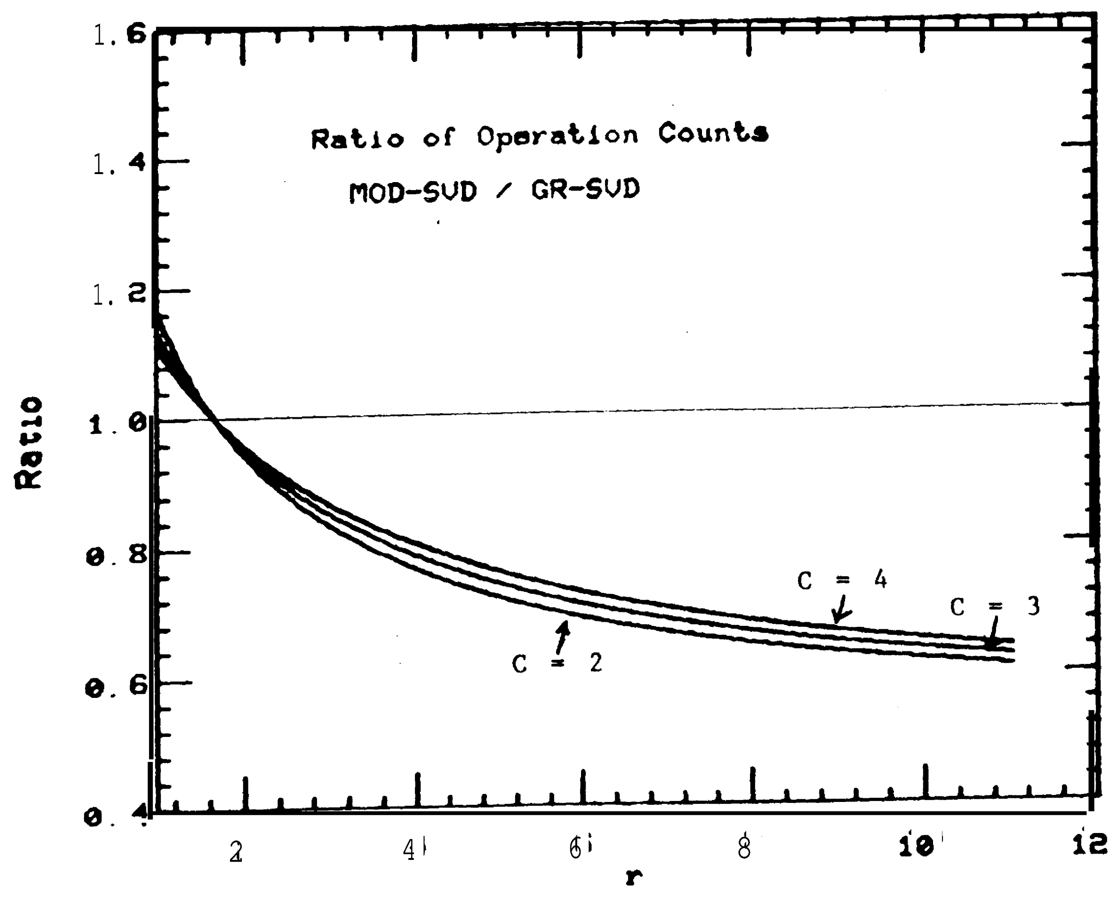
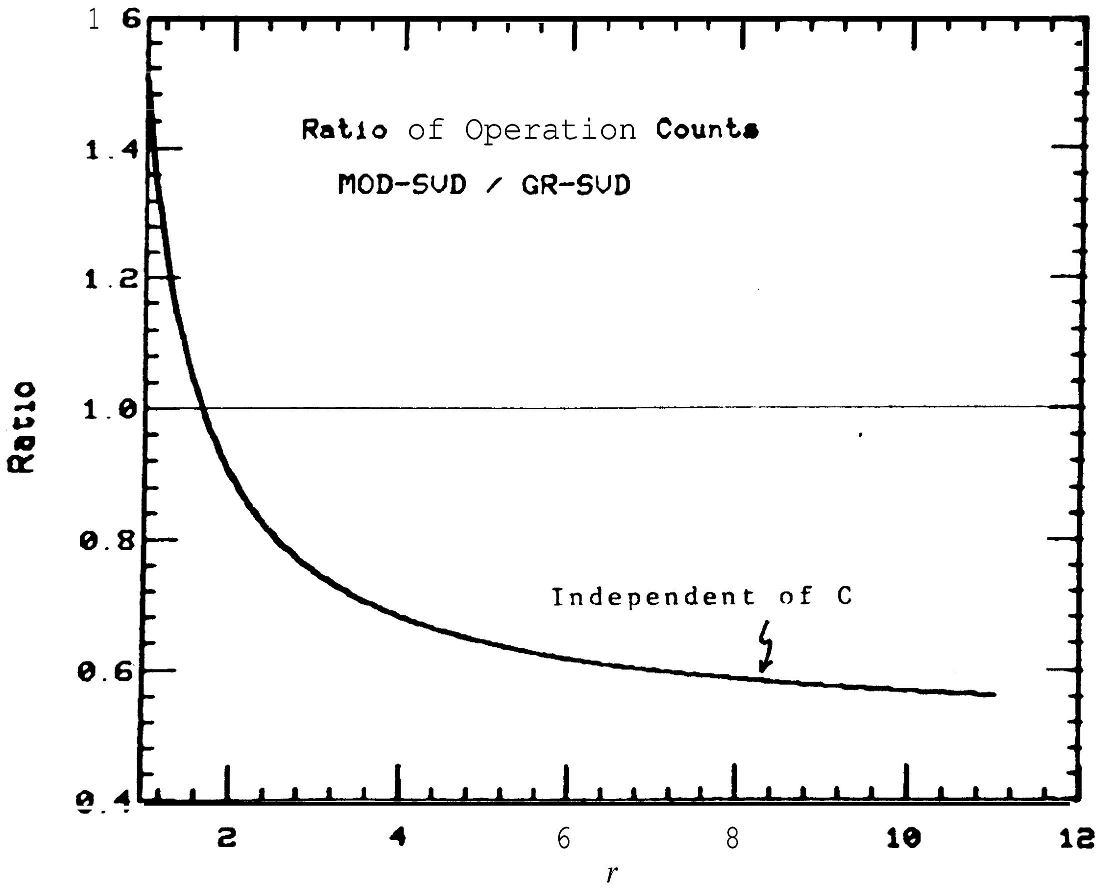

Ratio of Operation Counts
MOD-SVD / GR-SVD
Ratio
1.6
1.4
1.2
1.0
0.8
0.6
0.4
2
4
6
8
10
12
r
C = 4
C = 3
C = 2

Fig. 4.4 Case d



Ratio of Operation Counts
MOD-SVD / GR-SVD
Independent of C
1.6
1.4
1.2
1.0
0.8
0.6
0.4
Ratio
2
4
6
8
10
12
r

thing is that all the curves decrease quite fast as r becomes large. If we assume that it is equally likely to encounter matrices with any value of $r >= 1$ (this is not an unreasonable assumption for designers of general mathematical software, for example), then MOD-SVD is obviously preferable. In any case, Fig. 4.1 - 4.4 give indications as to when one of the methods is more efficient, at least when m and n are large enough so that our operation counts apply.

In the context of least squares applications, we can also compare the operation counts of GR-SVD and MOD-SVD to that of the orthogonal triangularization methods [9] (OTLS) often used for such problems. This comparison is shown in Table 4.4 .

Table 4.4

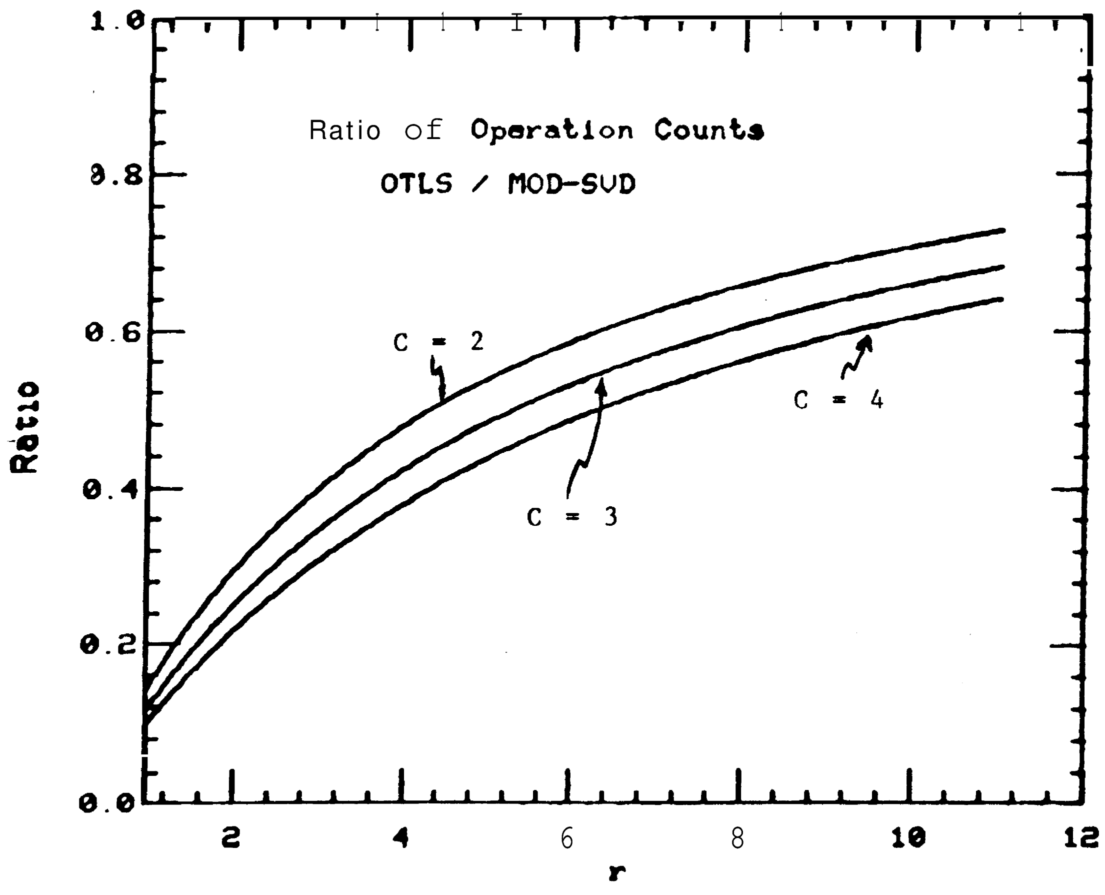
Least squares using orthogonal triangularization versus using SVD

OTLS = orthogonal triangularization method for least squares problems.

OTLS : GR-SVD =	$[r-1/3] / [2r+C]$
OTLS : MOD-SVD =	$[r-1/3] / [r+C+5/3]$

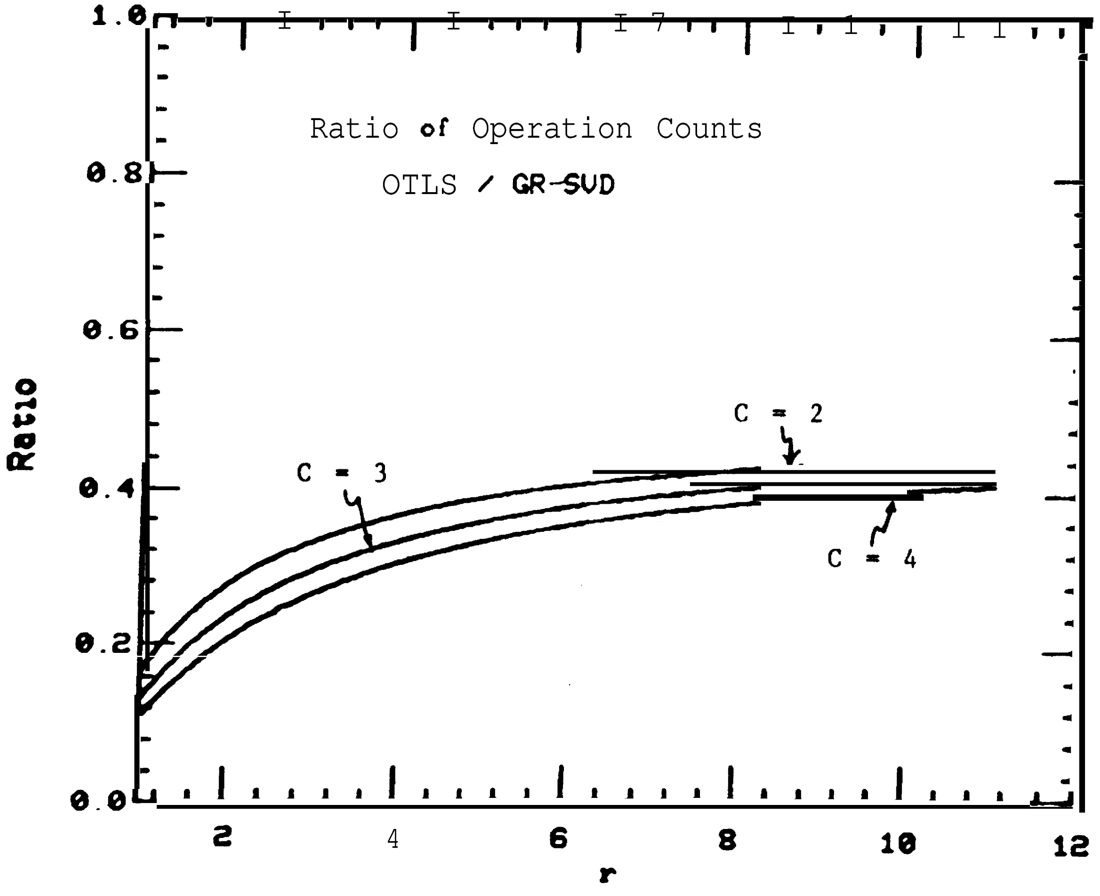
These ratios are plotted in Fig. 4.5 and Fig. 4.6 for $C=2,3,4$.

Fig. 4.5



Ratio of Operation Counts
OTLS / MOD-SVD
C = 2
C = 3
C = 4
Ratio
r
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12

Fig. 4.6



Ratio of Operation Counts
OTLS / GR-SVD
C = 2
C = 3
C = 4
Ratio
r
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12

One sees from these figures that for m nearly equal to n, the two SVD algorithms require much more work than OTLS. However, when **r** is bigger than about 3, MOD-SVD requires only about 3 times more work than OTLS. It may therefore become economically feasible to solve the least squares problems at hand by MOD-SVD instead of OTLS. The reward is that the SVD returns much more useful information about the problem than OTLS [3].

It is easy to see that as **r** becomes arbitrarily large, MOD-SVD is as efficient as OTLS since the bulk of the work is in the triangufarization of the data matrix A. However, GR-SVD can be at most half as efficient as OTLS.

(5) COMPUTATIONAL RESULTS

The conclusions in the last section hold only if m and n are both large. In this section, some computational experiments are carried out to see if the conclusions are still valid for matrices with realistic sizes.

We computed the SVD of some randomly generated **matrices** using both GR-SVD and MOD-SVD. The version of GR-SVD that we used is a modified ALGOL W translation of the procedure that appeared in [1]. MOD-SVD is realized by writing a procedure to triangularize the input matrix by Householder transformations and then using the same above-mentioned GR-SVD procedure for computing the SVD of R.

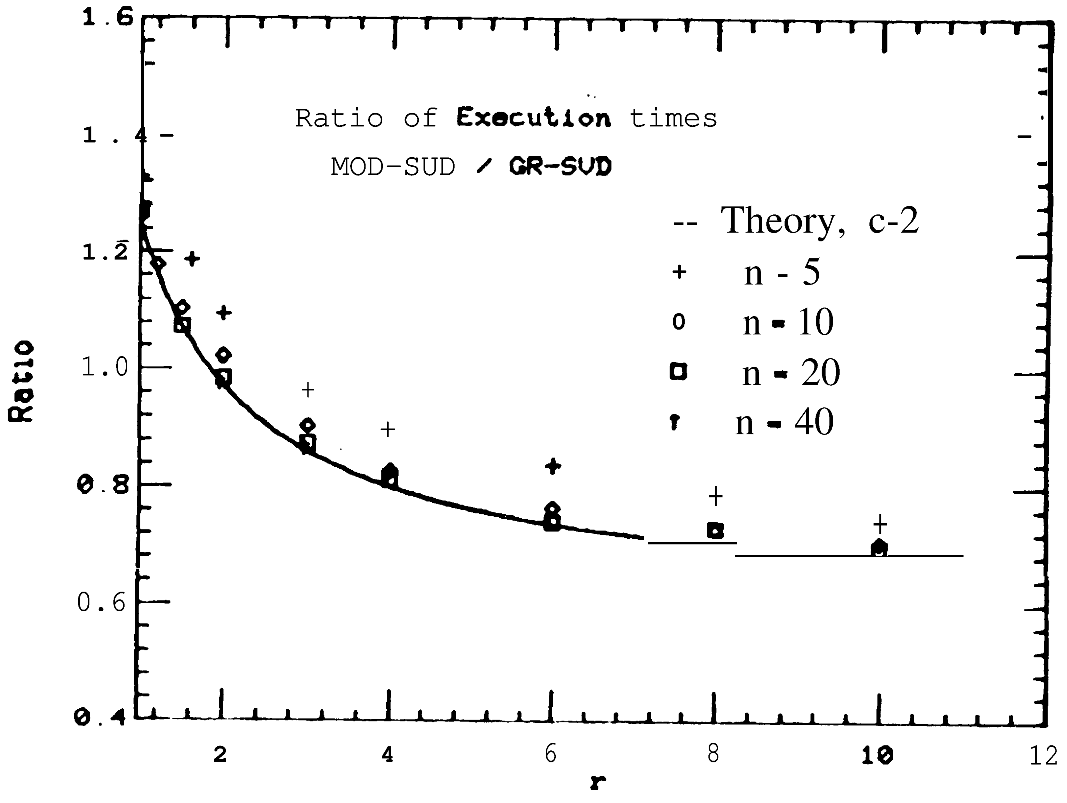
All tests were run on the IBM 370/168's at the Stanford Linear Accelerator Center (SLAC). Long precision was used throughout the calculation. The mantissa of a floating point number is represented by 56 bits (approximately 16 decimal digits).

For each of the 4 cases, we fixed some values for n and computed the SVD of a sequence of randomly generated matrices with different values of **r**. The execution times taken by GR-SVD and MOD-SVD were then compared, together with the accuracies of the computed answers. Since we are working in a multi-programming environment, the execution times we measured cannot be taken as the actual computing time taken. Moreover, the influence of the compiler on the relative efficiency of the two algorithms may be the deciding factor [11]. However, keeping these points in mind, we can still expect a qualitative agreement with the analysis based on operation counts.

On the IBM 370/168's at SLAC, a floating point multiplication takes only about 1.5 times the work taken for a floating point addition. Also, array indexing in ALGOL W is very expensive due to subscript checking (it actually can be more expensive than floating point multiplications). Therefore, as noted in section 4.1, we should use C approximately equal to 2 instead of 4 in Table 4.2 and Table 4.4, for the purpose of comparing the relative efficiency of the two algorithms based on the computational results.

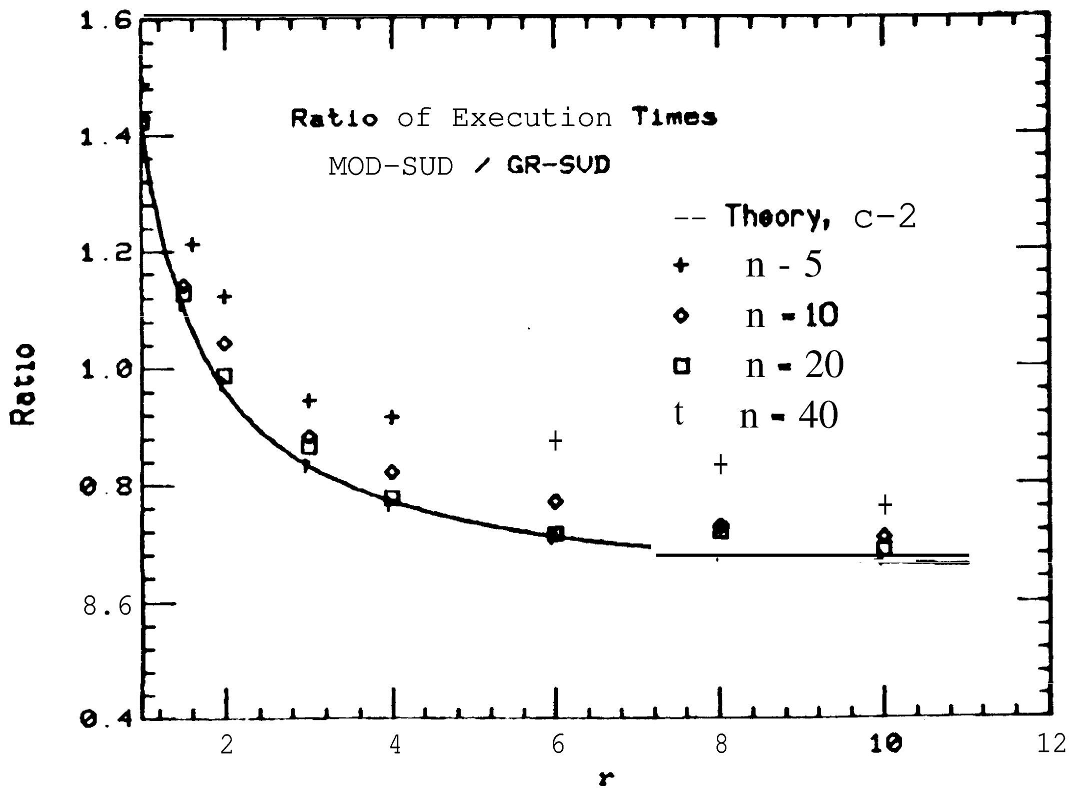
The results of the computations are plotted in Fig. 5.1 - Fig. 5.6 . In general, they agree very well qualitatively with the asymptotic results we obtained by operation counts (with C-2). We observe that the larger n is the better the agreement, as it should be. However, even when n is small, the theoretical results based on asymptotic operation counts still describe very well the qualitative behavior of the computational results in many cases. The computational results also show that large savings in work are indeed realizable for reasonably-sized matrices (For example, see Fig. 5.3 and Fig. 5.4).

Fig. 5.1 Case a



1.6
1.4
1.2
1.0
0.8
0.6
0.4
Ratio
2
4
6
8
10
12
r
Ratio of Execution times
MOD-SUD / GR-SVD
-- Theory, c-2
+ n - 5
o n - 10
□ n - 20
† n - 40

Fig. 5.2 Case b



Ratio of Execution Times
MOD-SUD / GR-SVD
-- Theory, c-2
+ n - 5
◇ n - 10
□ n - 20
t n - 40
Ratio
r
1.6
1.4
1.2
1.0
0.8
8.6
0.4
2
4
6
8
10
12

Fig. 5.3 Case c

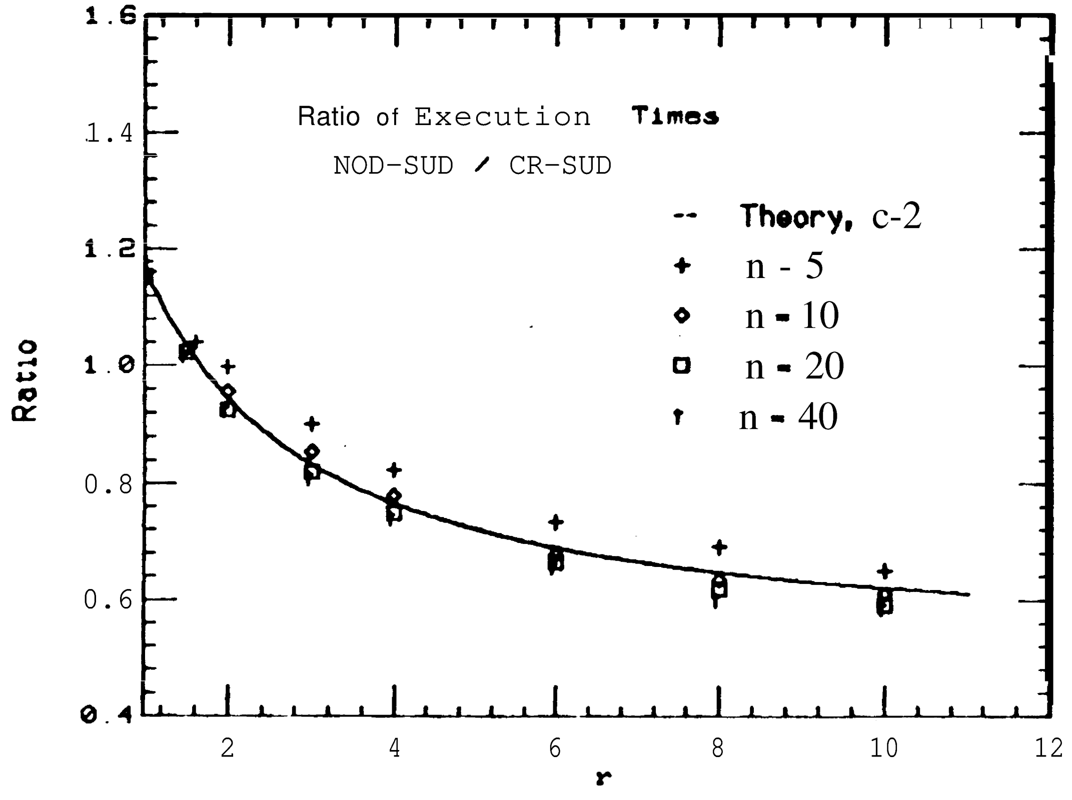
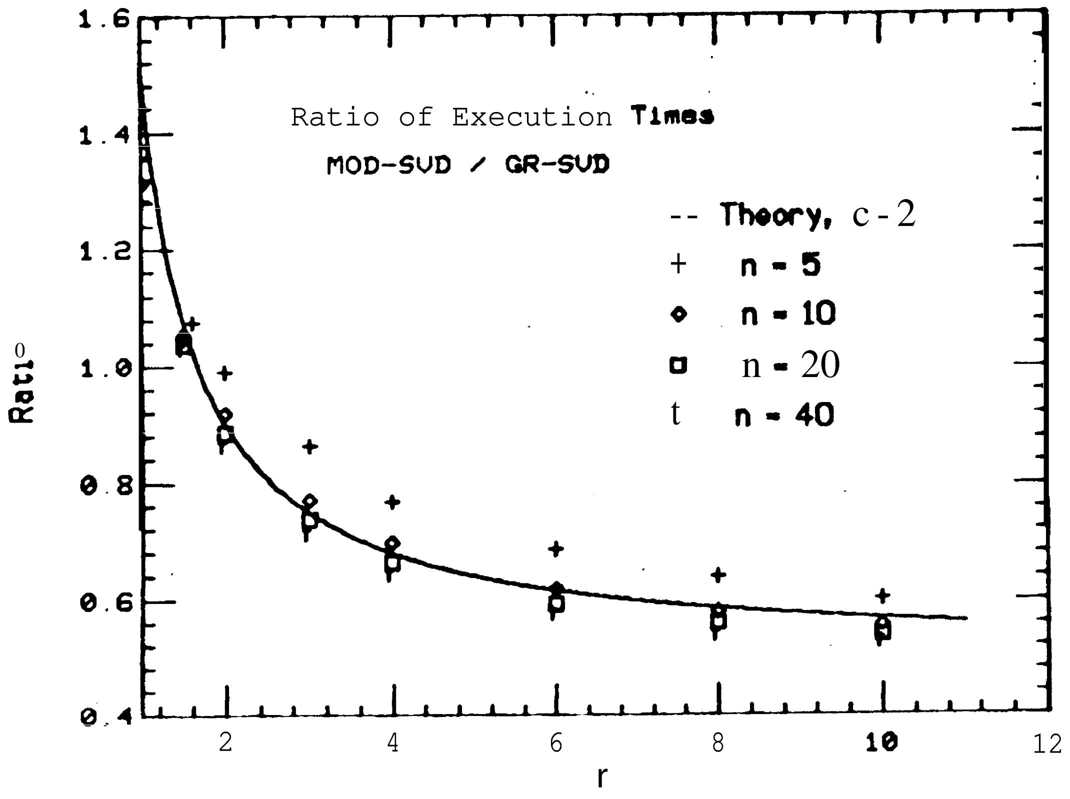
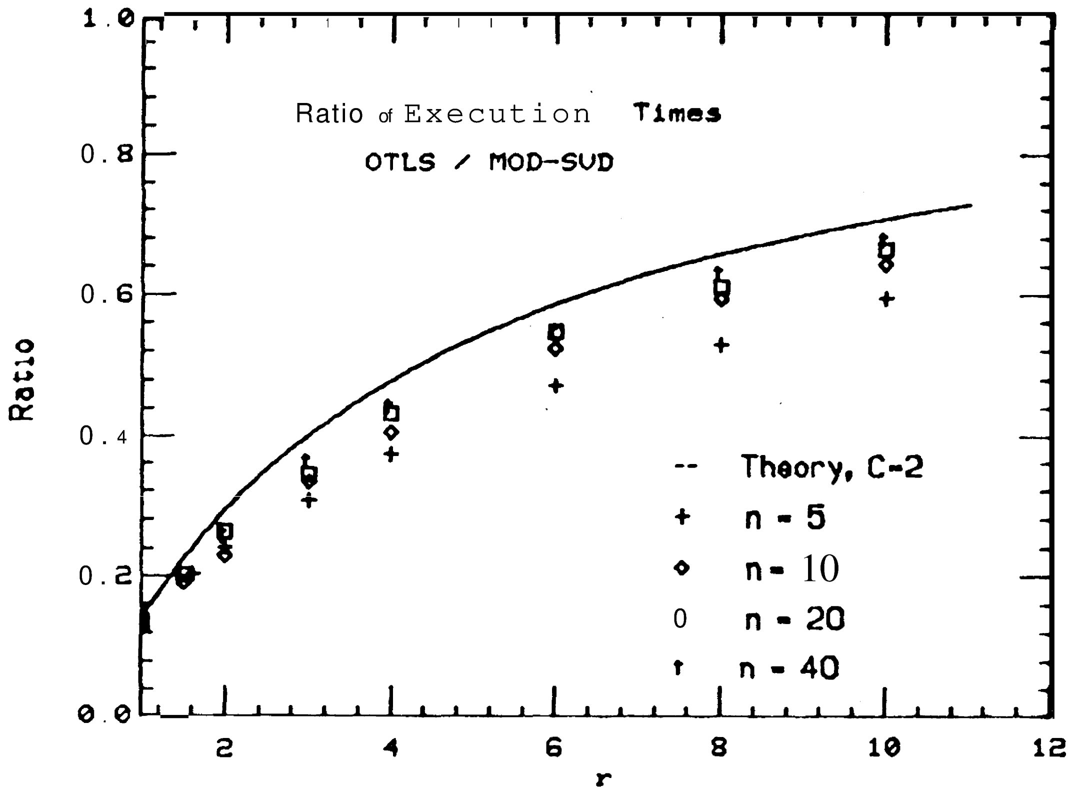

Ratio of Execution Times
NOD-SUD / CR-SUD
-- Theory, c-2
+ n - 5
◇ n - 10
□ n - 20
† n - 40
Ratio
r
1.6
1.4
1.2
1.0
0.8
0.6
0.4
2
4
6
8
10
12

Fig. 5.4 Case d



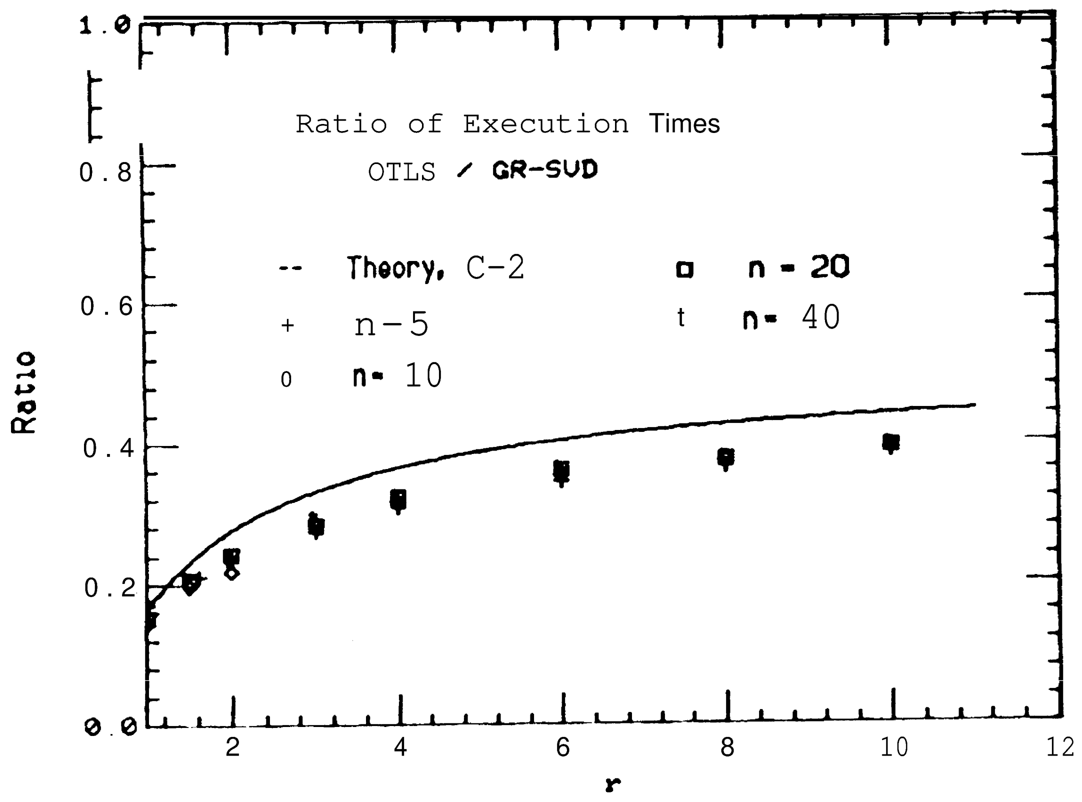
Ratio of Execution Times
MOD-SVD / GR-SVD
-- Theory, c-2
+ n - 5
◇ n - 10
□ n - 20
t n - 40
Ratio
r
1.6
1.4
1.2
1.0
0.8
0.6
0.4
2
4
6
8
10
12

Fig. 5.5



Ratio of Execution Times
OTLS / MOD-SVD
Ratio
r
-- Theory, C-2
+ n = 5
◇ n = 10
0 n = 20
† n = 40
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
12

Fig. 5.6


Ratio of Execution Times
OTLS / GR-SVD
-- Theory, C-2
+ n-5
o n= 10
□ n = 20
t n= 40
Ratio
r
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10
12

We also checked the accuracies of the computed results, The singular values returned by both procedures GR-SVD and MOD-SVD agree to within a few units of the machine precision in almost all cases that we have tested. The matrices U and V also agree to the same precision but the signs of the corresponding columns may be reversed. However, the SVD is only unique to within such a sign change, so this is acceptable [10].

We also computed the singular values of the following 30 x 30 matrix:

$$\begin{vmatrix} 1 & -1 & \cdots & \cdots & -1 \\ & 1 & -1 & & \vdots \\ & & \ddots & \ddots & \vdots \\ & 0 & & \ddots & -1 \\ & & & & 1 \end{vmatrix}$$

This matrix is very ill-conditioned (with respect to computing its inverse) and is very close to being a matrix of rank 29 even though the determinant equals 1 for all values of n. The computed singular values from both GR-SVD and MOD-SVD agree exactly with those given in [1] to 15 significant digits (which are all the digit8 printed in ALGOL W).

(6) CONCLUSIONS

Firstly, the theoretical results we obtained do seem to predict the actual computational efficiencies quite well, and they can therefore be used to indicate which algorithm to choose for a given matrix.

The MOD-SVD algorithm clearly work8 better than GR-SVD for matrices that have many more rows than columns. The price that MOD-,SVD ha8 to pay when m is nearly equal to n is not that big (usually **less** than 30%). We have also seen that the cost of solving a least squares problem by MOD-SVD can often be less than twice that of the usual orthogonal triangularization algorithms. It may therefore become economically feasible to solve many least squares problems by the SVD algorithms.

Some improvements can probably be made on the bidiagonalization of the upper triangular matrix R in MOD-SVD by taking advantage of the the special structure of R. We also want to note again that MOD-SVD requires n^2 extra storage locations if the left transformations have to be accumulated. This may be a disadvantage when storage is at a premium.

We have also seen that the usual practice of counting only multiplications in operation counts for numerical algorithms is no longer viable for many modern computers. Other properties, such as the amount of array accesses involved, may influence the efficiencies of algorithms decisively.

To be sure, there may be other ways to compute the SVD that will work better in some cases but not in others. It is perhaps impossible to find an “optimal” algorithm that works best for all matrices. Nevertheless, we hope this paper has shown that it may be worthwhile to look for improvements in the organizations of existing algorithms.

Appendix : **Fortran** Code of a Hybrid Algorithm

Based on the results of earlier sections, we can implement a hybrid method for computing the SVD of a rectangular matrix A which automatically chooses to use the more efficient algorithm between GR-SVD and MOD-SVD. For each of the four Cases a,b,c and **d**, if the input matrix A has a value of r (= m/n) which is less than the cross-over point r^* for that case, then we use GR-SVD, otherwise we use MOD-SVD. The cross-over points depend on the value of C used. As noted before, the value of C to be used depends on the relative efficiencies of floating point multiplications, floating point additions and array indexings on the particular machine concerned. However, C can be determined once for all for any particular machine and compiler combination. For example, if floating point multiplications take much more time than floating point additions and array indexing8 on the machine in question, then we should use C approximately equal to 4.

In this Appendix, we give the codes of a **Fortran** subroutine called HYBSVD which implements the above-mentioned hybrid algorithm. HYBSVD will need to call a standard Golub-Reinsch SVD subroutine during part of its computation and so we have included such a routine, called GRSVD, in the listing of the codes of HYBSVD.

The routine GRSVD is actually a slightly modified version of the subroutine SVD in the EISPACK [12] package. The main modification that we have made is to eliminate the requirement in subroutine SVD that the row dimension of V declared in the calling program be equal to that of A. This minimizes the storage requirements of GRSVD at the cost of one more argument in the argument list.

There is one additional feature implemented in HYBSVD (and also in GRSVD). In least squares applications, where we are looking for the minimal length least squares solution to the overdetermined linear system $Ax = b$, the left transformations U^T have to be accumulated on the right-hand side vectors b (there may be more than one b). This can be done by putting the vectors b in the matrix argument B when calling HYBSVD and -setting IRHS to the number of b's.

The calling sequences and usages of HYBSVD and GRSVD are explained in the comments in the beginning of the listings of the subroutines.

```
C     :::::::::::::: FIRST CARD OF HYBSVD ::::::::
C
      SUBROUTINE HYBSVD(NAU,NV,NZ,M,N,A,W,MATU,U,MATV,V,Z,B,IRHS,IERR,
     *                  RV1)
      INTEGER NAU,NV,NZ,M,N,IRHS,IERR,IP1,I,J,K,IM1,IBACK
      DOUBLE PRECISION A(NAU,N),W(N),U(NAU,N),V(NV,N),Z(NZ,N),
     *                 B(NAU,IRHS),RV1(N)
      DOUBLE PRECISION XOVRPT,C,R,G,SCALE,DSIGN,DABS,DSQRT,F,S,H
      REAL FLOAT
      LOGICAL MATU,MATV
C
C     THIS SUBROUTINE IS A MODIFICATION OF THE GOLUB-REINSCH PROCEDURE
C                                                                    T
C     (1) FOR COMPUTING THE SINGULAR VALUE DECOMPOSITION A = UWV  OF A
C     REAL M BY N RECTANGULAR MATRIX.  THE ALGORITHM IMPLEMENTED IN THIS
C     ROUTINE HAS A HYBRID NATURE.  WHEN M IS APPROXIMATELY EQUAL TO N,
C     THE GOLUB-REINSCH ALGORITHM IS USED, BUT WHEN M IS GREATER THAN
C     APPROXIMATELY 2*N, A MODIFIED VERSION OF THE GOLUB-REINSCH
C     ALGORITHM IS USED.  THIS MODIFIED ALGORITHM FIRST TRANSFORMS A
C                                                                    T
C     INTO UPPER TRIANGULAR FORM BY HOUSEHOLDER TRANSFORMATIONS L
C     AND THEN USES THE GOLUB-REINSCH ALGORITHM TO FIND THE SINGULAR
C     VALUE DECOMPOSITION OF THE RESULTING UPPER TRIANGULAR MATRIX R.
C     WHEN U IS NEEDED EXPLICITLY, AN EXTRA ARRAY Z (OF SIZE AT LEAST
C     N BY N) IS NEEDED, BUT OTHERWISE Z MAY COINCIDE WITH EITHER
C     A OR V AND NO EXTRA STORAGE IS REQUIRED.  THIS HYBRID METHOD
C     SHOULD BE MORE EFFICIENT THAN THE GOLUB-REINSCH ALGORITHM WHEN
C     M IS MUCH BIGGER THAN N.  FOR DETAILS, SEE (2).
C
C     HYBSVD CAN ALSO BE USED TO COMPUTE THE MINIMAL LENGTH LEAST
C     SQUARES SOLUTION TO THE OVERDETERMINED LINEAR SYSTEM A*X=B.
C
C     NOTICE THAT THE SINGULAR VALUE DECOMPOSITION OF A MATRIX
C     IS UNIQUE ONLY UP TO THE SIGN OF THE CORRESPONDING COLUMNS
C     OF U AND V.
C
C     THIS ROUTINE HAS BEEN CHECKED BY THE PFORT VERIFIER (3) FOR
C     ADHERENCE TO A LARGE, CAREFULLY DEFINED, PORTABLE SUBSET OF
C     AMERICAN NATIONAL STANDARD FORTRAN CALLED PFORT.
C
C     REFERENCES:
C
C     (1) GOLUB,G.H. AND REINSCH,C. (1970) 'SINGULAR VALUE
C         DECOMPOSITION AND LEAST SQUARES SOLUTIONS,'
C         NUMER. MATH. 14,403-420, 1970.
C
C     (2) CHAN,T.F. (1976) 'ON COMPUTING THE SINGULAR VALUE
C         DECOMPOSITION,' TO APPEAR AS A STANFORD COMPUTER
C         SCIENCE REPORT.
C
C     (3) RYDER,B.G. (1974) 'THE PFORT VERIFIER.' SOFTWARE
C         PRACTICE AND EXPERIENCE, VOL.4, 359-377, 1974.
C
C     HYBSVD ASSUMES M .GE. N.  IF M .LT. N, THEN COMPUTE THE
C                                      T          T       T          T
C     SINGULAR VALUE DECOMPOSITION OF A .  IF A  =UWV  , THEN A=VWU  .
C
C     ON INPUT:
C
```

41

```
C        NAU MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C           ARRAY PARAMETERS A,U AND B AS DECLARED IN THE CALLING PROGRAM
C           DIMENSION STATEMENT.    NOTE THAT NAU MUST BE AT LEAST
C           AS LARGE AS M;
C
C        NV MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C           ARRAY PARAMETER V AS DECLARED IN THE CALLING PROGRAM
C           DIMENSION STATEMENT. NV MUST BE AT LEAST AS LARGE AS N;
C
C        NZ MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C           ARRAY PARAMETER Z AS DECLARED IN THE CALLING PROGRAM
C           DIMENSION STATEMENT.    NOTE THAT NZ MUST BE AT LEAST
C           AS LARGE AS N;
C
C        M IS THE NUMBER OF ROWS OF A (AND U);
C
C        N IS THE NUMBER OF COLUMNS OF A (AND U) AND THE ORDER OF V;
C
C        A CONTAINS THE RECTANGULAR INPUT MATRIX TO BE DECOMPOSED;
C
C        B CONTAINS THE IRHS RIGHT-HAND SIDES OF THE OVERDETERMINED
C         LINEAR SYSTEM A*X=B.  IF IRHS.GT.0,
C          THEN ON OUTPUT, THESE IRHS COLUMNS IN B
C                           T
C          WILL CONTAIN U B. THUS, TO COMPUTE THE MINIMAL LENGTH LEAST
C                                                    +
C          SQUARES SOLUTION, ONE MUST COMPUTE V *W  TIMES THE COLUMNS OF
C                   +                           +
C          B, WHERE W  IS A DIAGONAL MATRIX, W (I)=0 IF W(I) IS
C          NEGLIGIBLE, OTHERWISE IS 1/W(I).  IF IRHS=0, B MAY COINCIDE
C          WITH A OR U AND WILL NOT BE REFERENCED;
C
C        IRHS IS THE NUMBER OF RIGHT HAND-SIDES OF THE OVERDETERMINED
C         SYSTEM A*X=B. IRHS SHOULD BE SET TO ZERO IF ONLY THE SINGULAR
C         VALUE DECOMPOSITION OF A IS DESIRED;
C
C        MATU SHOULD BE SET TO .TRUE. IF THE U MATRIX IN THE
C           DECOMPOSITION IS DESIRED, AND TO .FALSE. OTHERWISE;
C
C        MATV SHOULD BE SET TO .TRUE. IF THE V MATRIX IN THE
C           DECOMPOSITION IS DESIRED, AND TO .FALSE. OTHERWISE.
C
C        WHEN HYBSVD IS USED TO COMPUTE THE MINIMAL LENGTH LEAST
C        SQUARES SOLUTION TO AN OVERDETERMINED SYSTEM, MATU SHOULD
C        BE SET TO .FALSE., AND MATV SHOULD BE SET TO .TRUE..
C
C     ON OUTPUT:
C
C        A IS UNALTERED (UNLESS OVERWRITTEN BY U OR V);
C
C        W CONTAINS THE N (NONNEGATIVE) SINGULAR VALUES OF A (THE
C           DIAGONAL ELEMENTS OF W).   THEY ARE UNORDERED.   IF AN
C           ERROR EXIT IS MADE, THE SINGULAR VALUES SHOULD BE CORRECT
C           FOR INDICES IERR+1,IERR+2,...,N;
C
C        U CONTAINS THE MATRIX U (ORTHOGONAL COLUMN VECTORS) OF THE
C           DECOMPOSITION IF MATU HAS BEEN SET TO .TRUE.   OTHERWISE
C           U IS USED AS A TEMPORARY ARRAY.   U MAY COINCIDE WITH A.
C           IF AN ERROR EXIT IS MADE, THE COLUMNS OF U CORRESPONDING
C           TO INDICES OF CORRECT SINGULAR VALUES SHOULD BE CORRECT;
```

```
C
C         V CONTAINS THE MATRIX V (ORTHOGONAL) OF THE DECOMPOSITION IF
C            MATV HAS BEEN SET TO .TRUE.   OTHERWISE V IS NOT REFERENCED.
C            V MAY ALSO COINCIDE WITH A IF U IS NOT NEEDED.   IF AN ERROR
C            EXIT IS MADE. THE COLUMNS OF V CORRESPONDING TO INDICES OF
C            CORRECT SINGULAR VALUES SHOULD BE CORGECT:
C
C         Z CONTAINS THE MATRIX X IN THE SINGULAR VALUE  DECOMPOSITION
C                          T
C            OF R=XSY,   IF THE MODIFIED ALCOFITHM IS USED. IF THE
C            GOLUB-FEINSCH PROCEDURE IS USED, THEN IT IS NOT REFERENCED.
C            IF MATU HAS BEEN SET TO .FALSE.,   Z MAY COINCIDE
C            WITH A OR V  AND IS NOT REFERENCED;
C
C         IERR IS SET TC
C            ZERO          FCR NORMAL RETURN,
C            K             IF THE K-TH SINGULAR VALUE HAS NOT BEEN
C                          DETERMINED AFTER 30 ITERATIONS:
C            - 1           IF  IRHS  .LT.  0  .
C            - 2           IF   M .LT. N  .
C            - 3           IF  NAU  .LT. M .
C              4           IF  NV   .LT.  N  .
C              5           IF  NZ   .LT.  N  .
C
C         RV1 IS A TEMPORARY STORAGE ARRAY.
C
C      PROGRAMMED BY: TONY CHAN, COMP. SCI. DEPT.,
C                     STANFORD UNIV.  , CA 94305 .
C      LAST MODIFIED: 12 SEPTEMBER, 1976 .
C
C     ------------------------------------------------------------------
      IERR=0
      IF(IRHS.GE.0)GO TO 2
      IERR=-1
      RETURN
    2 IF(M.GE. N) GO TO 3
      IERR= 2
      RETURN
    3 IF (NAU.GE. M) GO TO 4
      IERR=-3
      RETURN
    4 IF  (NV.GE.N) GO TO 5
      IERR=-4
      RETURN
    5 IF (NZ.GE.N)  GO TO 6
      IERR= 5
      RETURN
    6 CONTINUE
C
C     SET VALUE FOR C. THE VALUE FOR C DEPENDS ON THE RELATIVE
C     EFFICIENCY OF FLOATING POINT MULTIPLICATIONS, FLOATING POINT
C     ADDITIONS AND TWO DIMENSIONAL ARRAY INDEXINGS ON THE
C     COMPUTER WHERE THIS SUBROUTINE IS TO BE RUN.  C SHOULD
C     USUALLY BE BETWEEN 2 AND 4.   FOR DETAILS ON CHOOSING C, SEE
C     (2).  THE ALGORITHM IS NOT SENSITIVE TO THE VALUE OF C
C     ACTUALLY USED AS LONG AS C IS BETWEEN 2 AND 4.
C
      C = 4.000
C
C     DETERMINE CROSS-OVER POINT
```

```
C
      IF(MATU.AND.MATV) X O V R P T  =(C+5.D0/3.D0)/C
      I  F(MATU .AND.  .NOT .MATV ) XOVRPT  =  (C+5.D0/3.D0)/C
      I  F(.NOT.MATU . A N D . MATV) XOVRPT  = 5.D0/3.D0
      I  F(.NOT.MATU .AND..NOT.MATV) X O V R P T  = 5.D0/3.D0
c
C     DETERMINE WHETHER TO USE GOLUB-REINSCH OR THE  MODIFIED
c     ALGORITHM.
C
      R  = FLOAT(M)/FLOAT(N)
      I F(R. G E .  XOVRPT)G O  T O 8
C
C     U S E GOLUB REINSCH PROCEDURE
C
      CALL  GRSVD(NAU,NV,M,N,A,W,MATU,U,MATV,V,B,IRHS,IERR,RV1)
      RETURN
C
C     USE MODIFIED ALGORITHM
C
  8   DO 10  I=1,M
         D  O 10 J=1,N
 10         U(I ,J) =A(I ,J)
C
C     T R I ANGULARIZE  U  BY HOUSEHOLDER T R A N S F O R M A T I O N S ,USING
C     W  A N D RV1  A  S T EMPORARY STORAGE.
C
      D O  2 0 I=1,N
         G=0.0D0
         S=0.0D0
         SCALE=0 .0D0
C
C        PERFORM  SCALING  OF COLUMNS T O  A V O I D  UNNECSSARY  OVERFLOW
C        O R UNDERFLOW
C
         DO 3  0 K=I,M
 30         SCALE=   SCALE  + DABS(U(K,I))
         IF  (SCALE .EQ.  0.0D0)G O   T O 20
         D O 4  0 K=I,M
            U(K,I)  = U(K, I )/SCALE
            S  = S    + U(K,I)**2
 40      CONTINUE
C
C        THE  VECTOR  E OF THE  HOUSEHOLDER  TRANSFORMATION I + EE'/H
C        WILL BE S T O R E D  IN COLUMN I O F  U .  T H E  T R A N S F O R M E D ELEMENT
C        U(I,I) WILL BE S  T  O  R  E  D   I N W(I) AND T  H  E SCALAR H I  N
C        RV1(I).
C
         F  = U(I,I)
         G  = -DSIGN(DSQRT(S),F)
         H  = F*G -   S
         U(I,I)  =  F  -  G
         RV1(I)  =  H
         W(I)  =  S C A L E *G
C
         I    F(I .EQ.  N)  GO   TO   85
C
C        A P P L Y TRANSFORMATIONS T O  R E M A I N I N G  C O L U M N S  O F A
C
         IP1  =  I  +  1
         D O  5 0 J=IP1,N
```

```
                 S = 0.0DC
                 DO 60 K=I,M
   63                 S =S    + U(K,I)*U(K,J)
                 F = S/H
                 DO 70 K=I,M
                      U(K,J) = U(K,J) + F*U(K,I)
   70            CONTINUE
   50         CONTINUE
C
C             APPLY TRANSFORMATIONS TO COLUMNS OF B IF IRHS .GT.0
C
   85         IF (IRHS.EQ.0) GO TO 20
              DO 80 J=1,IRHS
                 S = 0.000
                 DO 90 K=I,M
   90                 S = S + U(K,I)*B(K,J)
                 F = S/H
                 DO 100 K=I,M
                      B(K,J) = B(K,J) + F*U(K,I)
  103            CONTINUE
   80         CONTINUE
   20      CONTINUE
C
C          COPY R INTO Z IF MATU = .TRUE.
C
           IF (.NOT.MATU) GO TO 300
           DO 110 I=1,N
              DO 110 J=1,N
                 IF (J .GE. I) GO TO 112
                    Z(I,J) = 0.900
                    GO TO 110
  112            IF (J .EQ. I) GO TO 114
                    Z(I,J) = U(I,J)
                    GO TO 110
  114            Z(I,I) = W(I)
  110         CONTINUE
C
C          ACCUMULATE HOUSEHOLDER TRANSFORMATIONS IN U
C
           DO 120 IBACK=1,N
              I = N - IBACK + 1
              IP1 = I + 1
              G = W(I)
              H = RV1(I)
              IF (I .EQ.N) GO TO 130
C
              DO 140 J=IP1,N
  140            U(I,J) = 0.300
C
  130         IF (H .EQ. C.0D0) GO TO 150
              IF (I .EQ. N) GO TO 160
C
              DO 170 J=IP1,N
                 S = 0.0DC
                 DO 180 K=IP1,M
  180                 S = S + U(K,I)*U(K,J)
                 F = S/H
                 DO 170 K=I,M
                      U(K,J) = U(K,J) + F*U(K,I)
   70         CONTINUE
```

```
C
 160      S = U(I,I) / H
          DO 190 J=I,M
 190         U(J,I) = U(J,I)*S
          GO TO 200
C
 150      DO 210 J=I,M
 210         U(J,I) = 0.0D0
 200      U(I,I) = U(I,I) + 1.0D0
 120  CONTINUE
C
C     COMPUTE SVD OF R (WHICH IS STORED IN Z)
C
      CALL GRSVD(NZ,NV,N,N,Z,W,MATU,Z,MATV,V,B,IRHS,IERR,RV1)
C
C                                            T
C     FORM L*X TO OBTAIN U (WHERE R=XWY ).  X IS RETURNED IN Z
C     BY GRSVD. THE MATRIX MULTIPLY IS DONE ONE ROW AT A TIME,
C     USING RV1 AS SCRATCH SPACE.
C
      DO 220 I=1,M
         DO 230 J=1,N
            S = 0.0D0
            DO 240 K=1,N
 240           S = S + U(I,K)*Z(K,J)
 230        RV1(J) = S
            DO 250 J=1,N
 250           U(I,J) = RV1(J)
 220  CONTINUE
      RETURN
C
C     FORM R IN U BY ZEROING THE LOWER TRIANGULAR PART OF R IN U
C
 300  IF (N.EQ.1) GO TO 280
      DO 260 I=2,N
         IM1 = I - 1
         DO 270 J=1,IM1
 270        U(I,J) = 0.0D0
         U(I,I) = W(I)
 260  CONTINUE
 280  U(1,1) = W(1)
C
      CALL GRSVD(NAU,NV,N,N,U,W,MATU,U,MATV,V,B,IRHS,IERR,RV1)
      RETURN
C     THE BODY OF SUBROUTINE GRSVD SHOULD BE INCLUDED WITH HYBSVD
C
C     ::::::::::::::::: LAST CARD OF HYBSVD ::::::::::::::::::
      END
```

```
C     ■...Q-o*::: FIRST CARD OF GRSVD :::::::::
C     ---------------------------------------------------------------------
C
      SUBROUTINE GRSVD(NAU,NV,M,N,A,W,MATU,U,MATV,V,B,IRHS,IERR,RV1)
C
      INTEGER I,J,K,L,M,N,II ,I1,KK,K1,LL,L1 ,MN,NAU,NV,ITS,IERR,IRHS
      DOJBLE PRECISION A(NAU,N),W(N),U(NAU,N),V(NV,N),B(NAU,IRHS),RV1(N)
      DOUBLE PRECISION C,F,G,H,S,X,Y,Z,EPS,SCALE,MACHEP
      DOU9LE PRECISION DSQRT,DMAX1 ,DABS,DSIGN
      LOGICAL MATU,MATV
C
C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE SVD,
C     NUM. MATH. 14, 403-420(1970) BY GOLUB A N D REINSCH.
C     HANDBOOK FOR AUTO. COMP., VOLIf-LINEAR ALGEBRA, 134-151(1971).
C
C     THIS SUBROUTINE DETERMINES THE SINGULAR VALUE DECOMPOSITION
C            T
C     A=UWV  OF A REAL M BY N RECTANGULAR MATRIX.  HOUSEHOLDER
C     BIDIAGONALIZATIJN AYD A VARIANT OF THE OR ALGORITHM ARE USED.
C     GRSVD ASSUMES M.GE.N .  IF M .LT.N, THEN COMPUTE THE SI'JGULA?
C                                T        T      T            T
C     VALUE DECOMPOSITION OF A .  IF A =UWV , THEN A=VWU .
C
C     GRSVD CAN ALSO BE USED TO COMPUTE THE MINIMAL LENGTH LEAST SQUARES
C     SOLUTION TO THE OVERDETERMINED LINEAR SYSTEM A*X=B.
C
C     ON INPUT:
C
C        NAU MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C          ARRAY PARAMETERS A,U AND 8 AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT.  NOTE THAT NAU MUST BE AT LEAST
C_         AS LARGE AS M;
C
C        NV MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C          ARRAY PARAMETER V AS DECLARED IN THE CALLING PROGRAM
C          DIMENSION STATEMENT. NV MUST BE AT LEAST AS LARGE AS N;
C
C        M IS THE NUMBER OF ROWS OF A (AND U);
C
C        N IS THE NUMBER OF COLUMNS OF A (AND U) AND THE ORDER OF V;
C
C        A COJTAINS THE RECTANGULAR INPUT MATRIX TO BE DECOMPOSED;
C
C        B CONTAINS THE IRHS RIGHT-HAND-SIDES OF THE OVERDETERMINED
C         LINEAR SYSTEM A*X=B. IF IRHS.GT.0,
C         THEN ON OUTPUT, THESE IRHS COLUMNS
C                       T
C         WILL CONTAIN U B. THUS, TO COMPUTE THE MINI MAC LENGTH LEAST
C                                                +
C         SQUARES SOCUTI ON. ONE MUST COMPUTE V*W  TIMES THE COLUMNS OF
C                    +                          +
C         B, WHERE W  IS A DIAGONAL MATRIX, W (I)=0 IF W(I) IS
C         NEGLIGIBLE, OTHERWISE IS 1/W(I). IF IRHS=0,B MAY COINCIDE
C         WITH A OR U AND WILL NOT BE REFERENCED:
C
C        IPHS IS THE NUMBER OF RIGHT-HAND-SIDES OF THE OVERDETERMINED
C         SYSTEM A*X=B.  IRHS SHOULD BE SET TO ZERO IF ONLY THE SI NGULA?
C         VALUE DECOMPOSITION OF A IS DESIRED;
C
```

```
C         MATU SHOULD BE SET TO .TRUE. IF THE U MATRIX IN THE
C            DECOMPOSITION IS DESIRED, AND TO .FALSE. OTHERWISE:
C
C         MATV SHOULD BE SET TO .TRUE. IF THE V MATRIX IN THE
C            DECOMPOSITION IS DESIRED, AND TO .FALSE. OTHERWISE.
C
C      ON OUTPUT:
C
C         A IS UNALTERED (UNLESS OVERWRITTEN BY U OR V);
C
C         W CONTAINS THE N (NON-NEGATIVE) SINGULAR VALUES OF A (THE
C            DIAGONAL ELEMENTS OF W). THEY ARE UNORDERED. IF AN
C            ERROR EXIT IS MADE, THE SINGULAR VALUES SHOULD BE CORRECT
C            FOR INDICES IERR+1,IERR+2,....,N;
C
C         U CONTAINS THE MATRIX U (ORTHOGONAL COLUMN VECTORS) OF THE
C            DECOMPOSITION IF MATU HAS BEEN SET TO .TRUE. OTHERWISE
C            U IS USED AS A TEMPORARY ARRAY. U MAY COINCIDE WITH A.
C            IF AN ERROR EXIT IS MADE, THE COLUMNS OF U CORRESPONDING
C            TO INDICES OF CORRECT SINGULAR VALUES SHOULD BE CORRECT;
C
C         V CONTAINS THE MATRIX V (ORTHOGONAL) OF THE DECOMPOSITION IF
C            MATV HAS BEEN SET TO .TRUE. OTHERWISE V IS NOT REFERENCED.
C            V MAY ALSO COINCIDE WITH A IF U IS NOT NEEDED. IF AN ERROR
C            EXIT IS MADE, THE COLUMNS OF V CORRESPONDING TO INDICES OF
C            CORRECT SINGULAR VALUES SHOULD BE CORRECT;
C
C         IERR IS SET TO
C            ZERO        FOR NORMAL RETURN,
C            K           IF THE K-TH SINGULAR VALUE HAS NOT BEEN
C                        DETERMINED AFTER 30 ITERATIONS;
C            -1          IF IRHS .LT. 0 .
C            -2          IF M .LT. N .
C            -3          IF NAU .LT. M .
C            -4          IF NV .LT. N .
C
C         RV1 IS A TEMPORARY STORAGE ARRAY.
C
C
C      THIS SUBROUTINE HAS BEEN CHECKED BY THE PFORT VERIFIER
C      (RYDER, B.G. "THE PFORT VERIFIER", SOFTWARE - PRACTICE AND
C      EXPERIENCE, VOL.4, 359-377, 1974) FOR ADHERENCE TO A LARGE,
C      CAREFULLY DEFINED, PORTABLE SUBSET OF AMERICAN NATIONAL STANDARD
C      FORTRAN CALLED PFORT.
C
C      ORIGINAL VERSION OF THIS CODE IS SUBROUTINE SVD IN RELEASE 2 OF
C         EISPACK.
C      MODIFIED BY TONY CHAN. COMP. SCI. DEPT. STANFORD UNIV., CA94305.
C      LAST MODIFIED: 2 SEPTEMBER, 1976.
C
C      ------------------------------------------------------------------
C
C      ::::::::::: MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                  THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
C                  MACHEP = 16.0D0**(-13) FOR LONG FORM ARITHMETIC
C                  ON S360 ::::::::::
       DATA MACHEP/2.22D-16/
C
       IERR = 3
       IF (IRHS.GE.0) GO TO 2
```

48

```
      IERR=-1
      RETURN
    2 IF (M .GE. N )GO TO 3
      IERR=-2
      RETURN
    3 IF (NAU.GE.M) GO TO 4
      IERR=-3
      RETURN
    4 IF (NV .GE. N) GO TO 5
      IERR=-4
      RETURN
    5 CONTINUE
C
      DO 100 I = 1, M
C
         DO 190 J = 1, N
            U(I,J) = A(I,J)
  100 CONTINUE
C     :::::::::: HOUSEHOLDER REDUCTION TO BIDIAGONAL FORM ::::::::::
      G = 0.0D0
      SCALE = 0.0D0
      X = 0.0D0
C
      DO 303 I = 1, N
         L = I + 1
         RV1(I) = SCALE * G
         G = 0.0D0
         S = 0.0D0
         SCALE = 0.0D0
C
C     COMPUTE LEFT TRANSFORMATIONS THAT ZEROS THE SUBDIAGONAL ELEMENTS
C        OF THE IITH COLUMN.
C
         DO 120 K = I, M
  120    SCALE = SCALE + DABS(U(K,I))
C
         IF (SCALE.EQ.0.0D0) GO TO 210
C
         DO 130 K = I, M
            U(K,I) = U(K,I)/SCALE
            S = S + U(K,I)**2
  130    CONTINUE
C
         F = U(I,I)
         G = -DSIGN(DSQRT(S),F)
         H = F * G - S
         U(I,I) = F - G
         IF (I.EQ. N) GO TO 155
C
C     APPLY LEFT TRANSFORYATTONS TO REMAINING COLUMNS OF A
C
         DO 150 J = L, N
            S = 0.0D0
C
            DO 140 K = I, M
  140       S = S + U(K,I) * U(K,J)
C
            F = S / H
C
            DO 150 K = I, M
```

```
                  U(K,J) = U(K,J) + F *U(K,I)
  150        CONTINUE
C
C       APPLY LEFT TRANSFORMATIONS TO THE COLUMNS OF B IF IRHS .GT. 0.
C
 155          IF(IRHS.EQ.0)GO TO 190
              DO 160 J=1,IRHS
                  S=0.0D0
                  DO 170 K=I,M
  170                 S = S + U(K,I)*B(K,J)
                  F = S/H
                  DO 180 K=I,M
  180                 B(K,J) = B(K,J) + F*U(K,I)
  160         CONTINUE
C
C       COMPUTE RIGHT TRANSFORMATIONS.
C
  190        DO 200 K = I,M
  200        U(K,I) = SCALE*U(K,I)
C
  210        W(I) = SCALE* G
             G = 0.0D0
             S = 0.0D0
             SCALE = 0.0D0
             IF (I.GT. M .OR. I .EQ. N) GO TO 290
C
             DO 220 K = L, N
  220        SCALE = SCALE+ DABS(U(I,K))
C
             IF (SCALE.EQ. 0.0D0)GO TO 290
C
             DO 230 K = L, N
                U(I,K) = U(I,K)/SCALE
                S = S +U(I,K)**2
  230        CONTINUE
C
             F = U(I,L)
             G = -DSIGN(DSQRT(S),F)
             H = F I G - S
             U(I,L) = F - G
C
             DO 240 K = L, N
  240        RV1(K) = U(I,K) / H
C
             IF(I.EQ. M) GO TO 270
C
             DO 260 J = L,M
                S = 0.0D0
C
                DO 250 K = L,N
  250           S = S + U(J,K) * U(I,K)
C
                DO 26C K = L, N
                   U(J,K) = U(J,K)+ S * RV1(K)
  260        CONTINUE
C
  270        DO 280 K = L, N
  280        U(I,K) = SCALE*U(I,K)
C
  29c        X = DMAX1(X,DABS(W(I))+DABS(RV1(I)))
  300 CONTINUE
C     ::::::::: ACCUMULATION OF RIGHT-HAND TRANSFORYATIONS :::::::::
      IF(.NOT. MATV) GO TO 410
C     ::::::::: FOR I=N STEP -1 UNTIL 1 DO -- :::::::::
      DO 400 II = 1, N
         I = N + 1 - II
         IF (I .EQ. N) GC TO 390
         IF(G .EQ.0.0D0) GO TO 360
C
         DO 320 J = L, N
C     ::::::::: DOUBLE DIVISION AVOIDS POSSIBLE UNDERFLOW :::::::::
  320    V(J,I) = (U(I,J) / U(I,L)) / G
C
         DO 353 J = L, N
            S = 0.0D0
C
            DO 340 K = L, N
  340       S = S + U(I,K) * V(K,J)
C
            DO 353 K = L, N
               V(K,J) = V(K,J) + S * V(K,I)
  350    CONTINUE
C
  360    DO 380 J = L, N
            V(I,J) = 0.0D0
            V(J,I) = 0.0D0
  380    CONTINUE
C
  390    V(I,I) = 1.0D0
         G = RV1(I)
         L = I
  400 CONTINUE
C     ::::::::: ACCUMULATION OF LEFT-HAND TRANSFORMATIONS :::::::::
  41c IF (.NOT. MATU) GO TO 510
C     ::::::::: FOR I=MIN(M,N) STEP -1 UNTIL 1 DO -- :::::::::
      MN = N
      IF (M .LT. N) MN = M
C
      DO 500 II = 1, MN
         I = MN + 1 - II
         L = I + 1
         G = W(I)
         IF (I .EQ. N) GO TO 430
C
         DO 420 J = L, N
  420    U(I,J) = 0.0D0
C
  430    IF(G .EQ. 0.0D0) GO TO 475
         IF(I .EQ. MN ) GO TO 460
C
         DO 450 J = L, N
            S = 0.0D0
C
            DO 440 K = L, M
  440       S = S + U(K,I) * U(K,J)
C     ::::::::: DOUBLE DIVISION AVOIDS POSSIBLE UNDERFLOW :::::::::
            F = (S / U(I,I)) / G
C
            DO 450 K = I, M
               U(K,J) = U(K,J) + F * U(K,I)
  450     CONTINUE
C
  460     DO 470 J = I,M
  470     U(J,I) = U(J,I)/ G
C
          GO TO 490
C
  475     DO 480 J = I,M
  480     U(J,I) = 0.0D0
C
  490     U(I,I) = U(I,I) + 1.0D0
  500 CONTINUE
C     :::::::::: DIAGONALIZATION OF THE BIDIAGONAL FORM ::::::::::
  510 EPS = MACHEP * X
C     :::::::::: FOR K=N STEP -1 UNTIL 1 DO -- ::::::::::
      DO 700 KK = 1,N
         K1 = N - KK
         K = K1 + 1
         ITS = 0
C     .......... TEST FOR SPLITTING.
C                FOR L=K STEP -1 UNTIL 1 DO -- ::::::::::
  520    DO 530 LL = 1, K
            L1 = K - LL
            L = L1 + 1
            IF (DABS(RV1(L)) .LE.EPS) GO TO 565
C     :::::::::: RV1(1) IS ALWAYS ZERO, SO THERE IS NO EXIT
C                THROUGH THE BOTTOM OF THE LOOP ::::::::::
            IF (DABS(W(L1)).LE.EPS) GO TO 540
  530    CONTINUE
C     :::::::::: CANCELLATION OF RV1(L) IF L GREATER THAN 1 ::::::::::
  540    C = 0.0D0
         S = 1.0D0
C
         DO 560 I = L, K
            F = S * RV1(I)
            RV1(I) = C * RV1(I)
            IF (DABS(F) .LE.EPS) GO TO 565
            G = W(I)
            H = DSQRT(F*F+G*G)
            W(I) = H
            C = G / H
            S = -F / H
C
C     APPLY LEFT TRANSFORMATIONS TO B IF IRHS .GT. 0.
C
              IF(IRHS.EQ.0) GO TO 542
              DO 545 J=1,IRHS
                  Y=B(L1,J)
                  Z=B(I,J)
                  B(L1,J) = Y*C + Z*S
                  B(I,J) = -Y*S + Z * C
  545         CONTINUE
  542         CONTINUE
C
            IF(.NOT. MATU) GO TO 560
C
            DO 550 J = 1, M
               Y = U(J,L1)
               Z = U(J,I)
               U(J,L1) = Y * C + Z * S
```

```
                 U(J,I) = -Y * S + Z * C
  550          CONTINUE
C
  560       CONTINUE
C     :::::::::: TEST FOR CONVERGENCE ::::::::::
  565       Z = W(K)
            IF (L .EQ. K) GO TO 650
C     :::::::::: SHIFT FROM BOTTOM 2 BY 2 MINOR ::::::::::
            IF (ITS .EQ. 30) GO TO 1000
            ITS = ITS + 1
            X = W(L)
            Y = W(K1)
            G = RV1(K1)
            H = RV1(K)
            F = ((Y - Z) * (Y + Z) + (G - H) * (G + H)) / (2.0D0 * H * Y)
            G = DSQRT(F*F+1.0D0)
            F = ((X - Z) * (X + Z) + H * (Y / (F + DSIGN(G,F)) - H)) / X
C     :::::::::: NEXT QR TRANSFORMATION ::::::::::
            C = 1.0D0
            S = 1.0D0
C
            DO 600 I1 = L, K1
               I = I1 + 1
               G = RV1(I)
               Y = W(I)
               H = S * G
               G = C * G
               Z = DSQRT(F*F+H*H)
               RV1(I1) = Z
               C = F / Z
               S = H / Z
               F = X * C + G * S
               G = -X * S + G * C
               H = Y * S
               Y = Y * C
               IF (.NOT. MATV) GO TO 575
C
               DO 570 J = 1, N
                  X = V(J,I1)
                  Z = V(J,I)
                  V(J,I1) = X * C + Z * S
                  V(J,I) = -X * S + Z * C
  570          CONTINUE
C
  575          Z = DSQRT(F*F+H*H)
               W(I1) = Z
C     :::::::::: ROTATION CAN BE ARBITRARY IF Z IS ZERO ::::::::::
               IF (Z .EQ. 0.0D0) GO TO 580
               C = F / Z
               S = H / Z
  580          F = C * G + S * Y
               X = -S * G + C * Y
C
C     APPLY LEFT TRANSFORMATIONS TO B IF IRHS .GT. 0.
C
                  IF (IRHS .EQ. 0) GO TO 582
                  DO 585 J=1,IRHS
                     Y = B(I1,J)
                     Z = B(I,J)
                     B(I1,J) = Y*C + Z*S
                  B(I,J) = -Y*S + Z*C
  585          CONT :NUE
  582          CONTI NUE
C
             I F(.NOT. MATU)GO T O 600
C
             DO  590  J = 1, M
                Y = U(J,I1)
                Z = U(J,I)
                U(J,I1) = Y * C + Z * S
                U(J,I) = -Y * S + Z * C
  590        CONT IYUE
C
  600     CONTI NUE
C
          RV1(L) = 0.0D0
        RV1(K) =    F
          W(K) = X
          GO TO 520
C      : : : : : : : : : : CONVERGEYCE : : : : : : : : : :
  650     I F (Z .GE. 0.0D0) GO T O 700
C      : : : : : : : : : : W(K) IS MADE NON-NEGATIVE : : : : : : : : : :
          W(K) = -Z
          IF ( .NOT. MATV) GO TO 7C0
C
          DO  590  J = 1. N
  690     V(J,<) = -V(J,K)
C
  700 CONTINUE
C
      GO TO 1001
C      : : : : : : : : : : SET ERROR -- NO CONVERGENCE TO A
C                         SINGJLAR VALUE A F T E R 30 ITERATIONS : : : : : : : : : :
 1C00 IERR =   K
 100 1 RETURN
C     : : : : : : : : : : L A S T CARD O F GRSVD : : : : : : : : : :
      END
```

ACKNOWLEDGEMENT

The author would like to thank Prof. John Gregg Lewis for his initial interest in this work and for providing the ALGOL W SVD procedure that was used in the experiments. Special thanks are also due Prof. Gene Golub, Prof. Charles Van Loan and Mr. W.Coughran for their helpful discussions. The following persons also helped at one time or another: Dr. C.Lawson, Prof. R-Hanson, Prof. M.Gentleman, Prof. J.Oliger, Dr. P.Gill, Prof. J-Dennis, Mr. J.Bolstad and Dr. J.S.Pang. Finally, the author thanks SLAC for providing the computing time that was used.

REFERENCES

[1] Golub,G.H. and Reinsch,C. (1971) "Singular Vaiue Decomposition and Least Squares Solutions" in Handbook for Automatic Computation, II, Linear Algebra, by J.H.Wilkinson and C.Reinsch, Springer-Verlag, New York.

[2] Golub,G.H. and Kahan,W. (1965) "Calculating the Singular Values and Pseudoinverse of a Matrix,' SIAM J. Numer. Anal., '2, No.3, 205-224.

[3] Lawson,C.L. and Hanson,R.J. (1974) "Solving Least Squares Problems," Prentice-Hall, New Jersey.

[4] Golub,G.H. and Wilkinson,J.H. (1975) "Ill-conditioned Eigensystema and the Computation of the Jordan Canonical Form," To appear in SIAM Review, 1976.

[5] Hanson,R.J. (1971) "A Numerical Method for Solving Fredholm Integral Equations of the First Kind Using Singular Values," SIAM J. Numer. Anal., 8, No.3, 616-626.

[6] Andrews,H.C. and Patterson,C.L. (1976) "Singular Value Decompositions and Digital Image Processing," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP - 24, No.1, Feb. 1976.

[7] Bartels,R.H., Golub,G.H., and Saunders,M.A. (1970) "Numerical Techniques in Mathematical Programming," in Nonlinear Programming. Academic Press, New York, 123-176.

[8] Gentleman,W.M. (1972) 'Least Squares Computations by Givens Transformations without Square Roots,' Univ. of Waterloo Report CSRR-2062, Waterloo, Ontario, Canada.

[9] Golub,G.H. and Businger,P.A. (1965) "Linear Least Squares Solution by Householder Transformations,' Numer. Math., 7, Handbook series Linear Algebra, 269-276.

[10] Stewart.G.W. (1973) 'Introduction to Matrix Computations,' Academic Press, New York.

[11] Parlett,B.N. and Wang,Y. (1975) "The influence of the Compiler on the Cost of Mathematical Software,' ACM TOMS, Vol.1, No.1, March 1975, pp.35-46.

[12] Smith,B.T. et al (1976) "Matrix Eigensystem Routines - EISPACK Guide,' Second Edition, Springer Verlag, Lecture Notes in Computer Science Series.