

A BLOCK LANCZOS METHOD TO COMPUTE THE SINGULAR VALUES AND
CORRESPONDING SINGULAR VECTORS OF A MATRIX

by

Gene H. Golub, Franklin T. Luk, and Michael L. Overton

STAN-CS-77-635
OCTOBER 1977

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

A BLOCK LANCZOS METHOD TO COMPUTE THE SINGULAR VALUES AND
CORRESPONDING SINGULAR VECTORS OF A MATRIX

Gene H. Golub, Franklin T. Luk, and Michael L. Overton^{*}
Stanford University

ABSTRACT

We present a block Lanczos method to compute the largest singular values and corresponding left and right singular vectors of a large sparse matrix. Our algorithm does not transform the matrix A but accesses it only through a user-supplied routine which computes AX or A^tX for a given matrix X .

This paper also includes a thorough discussion of the various ways to compute the singular value **decomposition** of a banded upper triangular matrix; this problem arises as a subproblem to be solved during the block Lanczos procedure.

Key Words and Phrases: Block Lanczos method, singular values, singular vectors, large sparse matrix, singular value decomposition, banded upper triangular matrix.

CR Categories: 5.14

^{*}Research supported in part under Army Research Grant DAHC04-75-G-0195 and in part under National Science Foundation Grant MCS75-13497-A01.

1. Introduction

In many applications, we wish to solve the following problem:

Compute accurate approximations to the g largest singular values and corresponding left and right singular vectors of a large sparse $m \times n$ real matrix A , where g is much less than both m and n .

Problems of this type frequently occur in factor analysis, regression, and image processing (see Golub and Luk [5]).

The matrix A is too large to be stored in core as an $m \times n$ array, but since it is sparse it can be stored in packed form, e.g. by storing only the row index, column index and value of each non-zero element. When A is stored in this way it is not practical to apply transformations to A but matrix products AX or A^tX for a given matrix X of much smaller dimension than A can be performed very efficiently. Thus the usual algorithm for computing singular values by transforming A (Golub and Reinsch [6]) is not practical for large sparse matrices. We propose a block Lanczos algorithm for solving such problems. Our algorithm does not transform A . It accesses A only through a user-supplied routine that computes AX or A^tX for a given matrix X .

2. Algorithm

We restate our problem: we have an $m \times n$ matrix A , where $m \geq n$, and we wish to compute the g largest singular values and corresponding vectors of A , assuming that the h ($h < g$) largest singular values and corresponding vectors have already been computed to some known accuracy.

We discuss an idea of Lanczos [7]; the matrix $\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}$ has for its non-zero eigenvalues the positive singular values of A , each appearing with both a positive and a negative sign. If \underline{u}_i and \underline{v}_i are the left and right singular vectors corresponding to the positive singular value σ_i of A , then $\begin{pmatrix} \underline{u}_i \\ \underline{v}_i \end{pmatrix}$ and $\begin{pmatrix} \underline{u}_i \\ -\underline{v}_i \end{pmatrix}$ will be the eigenvectors corresponding to the eigenvalues σ_i and $-\sigma_i$, resp., of $\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}$.

Our problem can therefore be regarded as computing the g largest eigenvalues and eigenvectors of $\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}$, when the h largest eigenvalues and eigenvectors are known to some good accuracy.

We shall use the Euclidean norm for vectors and the Frobenius norm for matrices, viz.

$$\|\underline{x}\| = \|\underline{x}\|_2 = \left(\sum_{i=1}^n x_i^2 \right)^{1/2} \quad \text{for } \underline{x} = (x_1, \dots, x_n)^t,$$

$$\|A\| = \|A\|_F = \left(\sum_{i=1}^n \sum_{j=1}^n a_{ij}^2 \right)^{1/2} \quad \text{for } A = (a_{ij}).$$

2.1 Restricting A to a Subspace of Interest

Let $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_h$ be the h largest computed singular values of A and let X_0 and Y_0 be matrices whose columns are the computed left and right singular vectors, resp., such that $X_0^t X_0 = I$ and

$Y_0^t Y_0 = I$. We desire accurate approximations to the $(g-h)$ largest singular values and vectors of \bar{A} , defined by $\bar{A} = (I - X_0 X_0^t)^t A (I - Y_0 Y_0^t)$ so that the left singular vectors of \bar{A} are orthogonal to the columns of X_0 and the right singular vectors of \bar{A} are orthogonal to the columns of Y_0 . This restriction is necessary because our algorithm, if applied to A without taking the already computed singular vectors into account, will recompute the same largest singular values of A .

We can exploit **Lanczos's** idea and examine $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$. We can show that $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$ is the restriction of $\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}$ to a **subspace** that is orthogonal to the space spanned by the columns of $\begin{pmatrix} X_0 & X_0 \\ Y_0 & -Y_0 \end{pmatrix}$.

Let X_1 and Y_1 be the matrices consisting of the orthonormal vectors that are orthogonal to the **subspace** spanned by the columns of X_0 and Y_0 , **resp.** . Define

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} X_0 & X_0 & X_1 \\ Y_0 & -Y_0 & Y_1 \end{pmatrix}.$$

Note

$$Q^t Q = \frac{1}{2} \begin{pmatrix} 2I & & \\ & 2I & \\ & & 2I \end{pmatrix} = I.$$

Consider

$$\begin{aligned} B &= Q^t \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} Q \\ &= C + \Delta, \end{aligned}$$

where

$$c = \begin{pmatrix} \Lambda & 0 \\ 0 & \frac{1}{2}(Y_1^t A^t X_1 + X_1^t A Y_1) \end{pmatrix},$$

$$\Lambda = \frac{1}{2} \begin{pmatrix} Y_0^t A^t X_0 + X_0^t A Y_0 & Y_0^t A^t X_0 - X_0^t A Y_0 \\ -Y_0^t A^t X_0 + X_0^t A Y_0 & -Y_0^t A^t X_0 - X_0^t A Y_0 \end{pmatrix},$$

and

$$\Delta = \frac{1}{2} \begin{pmatrix} 0 & 0 & Y_0^t A^t X_1 + X_0^t A Y_1 \\ 0 & 0 & -Y_0^t A^t X_1 + X_0^t A Y_1 \\ Y_1^t A^t X_0 + X_1^t A Y_0 & Y_1^t A^t X_0 - X_1^t A Y_0 & 0 \end{pmatrix}.$$

Note

$$\Lambda = \frac{1}{\sqrt{2}} \begin{pmatrix} X_0^t & Y_0^t \\ X_0^t & -Y_0^t \end{pmatrix} \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} X_0 & X_0 \\ Y_0 & -Y_0 \end{pmatrix},$$

and

$$\frac{1}{2}(Y_1^t A^t X_1 + X_1^t A Y_1) = \frac{1}{\sqrt{2}} \begin{pmatrix} X_1^t & Y_1^t \end{pmatrix} \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} X_1 \\ Y_1 \end{pmatrix}.$$

Since B is similar to A, they have equal eigenvalues. By the perturbation theory for symmetric matrices [14, Chap. 2], the eigenvalues of C differ from those of B (and hence A) by amounts that are bounded by $\|\Delta\|$.

Assume

$$\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} X_0 & X_0 \\ Y_0 & -Y_0 \end{pmatrix} = \begin{pmatrix} X_0 & X_0 \\ Y_0 & -Y_0 \end{pmatrix} \Sigma_1 + R, \quad ,$$

where

$$\Sigma_1 = \begin{pmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_h & & \\ & & & -\sigma_1 & \\ & & & & \ddots \\ & & & & & -\sigma_h \end{pmatrix},$$

and

$$R = \begin{pmatrix} \xi_1 & \cdots & \xi_n & \xi_1 & \cdots & \xi_h \\ \eta_1 & \cdots & \eta_h & -\eta_1 & \cdots & -\eta_h \end{pmatrix}.$$

Then

$$\begin{aligned} & (X_1^t \quad Y_1^t) \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix} \\ &= (X_1^t \quad Y_1^t) \left\{ \begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix} \Sigma_1 + R \right\} \\ &= (X_1^t \quad Y_1^t) R. \end{aligned}$$

Now

$$\begin{aligned} \|\Delta\| &= \frac{1}{2} \| (Y_1^t A^t X_o + X_1^t A Y_o \quad Y_1^t A^t X_o - X_1^t A Y_o) \| \\ &= \frac{1}{2} \| (X_1^t \quad Y_1^t) \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix} \| \\ &= \frac{1}{2} \| (X_1^t \quad Y_1^t) R \| \\ &\leq \frac{1}{2} \| (X_1^t \quad Y_1^t) \| \|R\| \\ &= \frac{1}{\sqrt{2}} \|R\|. \end{aligned}$$

If all the $\|\xi_i\|$ and $\|\eta_i\|$ were **small**, then $\|\Delta\|$ would be small also. For example, if

$$\|\xi_i\| = \epsilon_i \quad ,$$

and

$$\|\eta_i\| = \delta_i \quad ,$$

then

$$\|R\| = \sqrt{2} \left(\sum_{i=1}^h \epsilon_i^2 + \sum_{i=1}^h \delta_i^2 \right) \equiv \sqrt{2} \epsilon$$

and

$$\|\Delta\| \leq \frac{1}{\sqrt{2}} \cdot \sqrt{2} \epsilon = \epsilon \quad ;$$

thus the eigenvalues of C will differ from those of B , and hence A , by quantities that are less in modulus than ϵ .

We see, therefore, that the $(g-h)$ largest eigenvalues of

$$\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix} \text{ approximate the } (h+1), (h+2), \dots, g \text{ eigenvalues of } \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}$$

by errors less than ϵ .

2.2 Block Bidiagonalization

Let us describe a block Lanczos algorithm that computes a block bidiagonal matrix. We shall **call** this matrix $J^{(s)}$, where s is the number of blocks and each block is of order p . Then $J^{(s)}$ has order ps (where we assume $ps \leq n$). We shall show in section 2.3 that the p largest singular value of $J^{(s)}$ are usually good approximations to those of \bar{A} .

We start with an arbitrary $n \times p$ matrix Q_1 such that $Q_1^t Q_1 = I$, and perform a **QR** factorization of the product $\bar{A}Q_1$:

$$P_1 A_1 := \bar{A} Q_1 ,$$

where P_1 is an $m \times p$ matrix such that $P_1^t P_1 = I$, and A_1 is a $p \times p$ upper triangular matrix. Our algorithm continues with

$$\text{and } \left. \begin{aligned} Q_i B_{i-1} &:= \bar{A}^t P_{i-1} - Q_{i-1} A_{i-1}^t , \\ P_i A_i &:= \bar{A} Q_i - P_{i-1} B_{i-1}^t , \end{aligned} \right\} \quad i = 2, 3, \dots, s ,$$

where $Q_i B_{i-1}$ and $P_i A_i$ are the QR factorizations of the respective right hand sides, i.e.

$$\begin{aligned} Q_i &\text{ is an } n \times p \text{ matrix such that } Q_i^t Q_i = I , \\ P_i &\text{ is an } m \times p \text{ matrix such that } P_i^t P_i = I , \end{aligned}$$

and both B_{i-1} and A_i are $p \times p$ upper triangular matrices.

Thus ,

$$\bar{A}(Q_1, Q_2, \dots, Q_s) = (P_1 P_2, \dots, P_s) \begin{pmatrix} A_1 & B_1^t & & & \\ & A_2 & B_2^t & & \\ & & \ddots & \ddots & \\ \bigcirc & & & A_{s-1} & B_{s-1}^t \\ & & & & A_s \end{pmatrix} ,$$

and

$$\begin{pmatrix} P_1 \\ P_2^t \\ \vdots \\ P_s^t \end{pmatrix} \bar{A}(Q_1, Q_2, \dots, Q_s) = \begin{pmatrix} A_1 & B_1^t & & & \\ & A_2 & B_2^t & & \\ & & \ddots & \ddots & \\ 0 & & & A_{s-1} & B_{s-1}^t \\ & & & & A_s \end{pmatrix} \equiv J^{(s)} ,$$

provided that $P_i^t P_j = 0$ for if j . In order to show this we first note that

$$\begin{pmatrix} P_1^t & 0 \\ P_2^t & 0 \\ \vdots & \vdots \\ P_s^t & 0 \\ 0 & Q_1^t \\ 0 & Q_2^t \\ \vdots & \vdots \\ 0 & Q_s^t \end{pmatrix} (0 \quad \bar{A}) \begin{pmatrix} P_1 & P_2 & \dots & P_s & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & Q_1 & Q_2 & \dots & Q_s \end{pmatrix} =$$

$$\begin{pmatrix} 0 & Q_1^t \\ P_1^t & 0 \\ 0 & Q_2^t \\ -P_2^t & 0 \\ \vdots & \vdots \\ 0 & Q_s^t \\ P_s^t & 0 \end{pmatrix} \begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix} \begin{pmatrix} 0 & P_1 & 0 & P_2 & \cdots & 0 & P_s \\ Q_1 & 0 & Q_2 & 0 & \cdots & Q_s & 0 \end{pmatrix} = \begin{pmatrix} 0 & A_1^t & & & & & \\ A_1 & 0 & B_1^t & & & & \\ & B_1 & 0 & A_2^t & & & \\ & & A_2 & 0 & \ddots & & \\ & & \ddots & \ddots & \ddots & & \\ & & & B_{s-1} & 0 & A_s^t & \\ & & & & A_s & 0 & \end{pmatrix}$$

So our algorithm to generate the block bidiagonal matrix $\begin{pmatrix} & \\ & \end{pmatrix}_s$ from \bar{A} is equivalent to the **Lanczos** algorithm (Underwood [12]) to generate a block tridiagonal matrix from the symmetric matrix $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$. From [12, pp. 47-51], it follows that

$\begin{pmatrix} 0 \\ q_1 \end{pmatrix}, \begin{pmatrix} p_1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ q_2 \end{pmatrix}, \begin{pmatrix} p_2 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ q_s \end{pmatrix}, \begin{pmatrix} p_s \\ 0 \end{pmatrix}$ form a sequence of

orthonormal matrices. Therefore $\{P_i\}$ and $\{Q_i\}$ are two sequences of orthonormal matrices.

The restricted matrix $\bar{\mathbf{A}}$ is not readily available. We wish

to work with the original matrix A. Consider

$$\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix} \\ = \left\{ I - \frac{1}{\sqrt{2}} \begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} X_o^t & Y_o^t \\ X_o^t & -Y_o^t \end{pmatrix} \right\} \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \left\{ I - \frac{1}{\sqrt{2}} \begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} X_o^t & Y_o^t \\ X_o^t & -Y_o^t \end{pmatrix} \right\} \\ = \begin{pmatrix} (I - X_o X_o^t) & 0 \\ 0 & (I - Y_o Y_o^t) \end{pmatrix} \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} (I - X_o X_o^t) & 0 \\ 0 & (I - Y_o Y_o^t) \end{pmatrix},$$

and the fact that $\begin{pmatrix} 0 \\ Q_1 \end{pmatrix}, \begin{pmatrix} P_1 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ Q_s \end{pmatrix}, \begin{pmatrix} P_s \\ 0 \end{pmatrix}$ all belong to the Krylov space* generated by $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ Q_1 \end{pmatrix}$. We conclude that we may replace \bar{A} by A in our algorithm if we orthogonalize the P_i 's, $1 \leq i \leq s$, with respect to X_o and the Q_j 's, $2 \leq j \leq s$, with respect to Y_o :

Algorithm

Start with an arbitrary $n \times p$ matrix Q_1 such that $Q_1^t Q_1 = I$.

Compute

$$\hat{P}_1 := A Q_1$$

and

$$\hat{P}_1 := (I - X_o X_o^t) \hat{P}_1.$$

Factorize \hat{P}_1 such that

* The Krylov space generated by A and X is the space spanned by $\{X, AX, A^2 X, A^3 X, \dots\}$.

$$P_1 A_1 := \hat{P}_1, \text{ where } P_1^t P_1 = I \text{ and } A_1 = \{\nabla\}.$$

For $i = 2, 3, \dots, s$

(1) Compute

$$\hat{Q}_i := A^t P_{i-1} - Q_{i-1} A_{i-1}^t$$

and

$$\hat{Q}_i := (I - Y_O Y_O^t) \hat{Q}_i$$

Factorize \hat{Q}_i such that

$$Q_i B_{i-1} := \hat{Q}_i, \text{ where } Q_i^t Q_i = I \text{ and } B_{i-1} = \{\nabla\}.$$

(2) Compute

$$\hat{P}_i := A Q_i - P_{i-1} B_{i-1}^t$$

and

$$\hat{P}_i := (I - X_O X_O^t) \hat{P}_i.$$

Factorize \hat{P}_i such that

$$P_i A_i := \hat{P}_i, \text{ where } P_i^t P_i = I \text{ and } A_i = \{\nabla\}.$$

2.3 Error Bounds

We give a theorem to show that the singular values of $J^{(s)}$ are usually accurate approximations to those of \bar{A} .

Theorem

Let $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n \geq 0$ be the singular values of the $m \times n$ restricted matrix \bar{A} and let $\sigma_1^{(s)} \geq \sigma_2^{(s)} \geq \dots \geq \sigma_{ps}^{(s)} \geq 0$ be the singular values of the $ps \times ps$ matrix $J^{(s)}$ generated by the block Lanczos algorithm. Let τ be the smallest singular value of $Q_1^t V_1$,

where Q_1 is an $n \times p$ starting matrix for the Lanczos algorithm such that $Q_1^T Q_1 = I$ and V_1 is an $n \times p$ matrix consisting of the right singular vectors corresponding to the p largest singular values of \bar{A} . We assume $\tau > 0$ and we see $\tau \leq 1$ since $V_1^T V_1 = I$. Then for $k = 1, 2, \dots, p$, we obtain

$$\sigma_k - \epsilon_k^2 \leq \sigma_k^{(s)} \leq \sigma_k, \quad ,$$

where

$$\epsilon_k^2 = (\sigma_1 + \sigma_k) \frac{\tan^2 \theta}{T_{2s-1}^2\left(\frac{1+\gamma_k}{1-\gamma_k}\right)},$$

$$\theta = \cos^{-1} \tau, \quad ,$$

$$\gamma_k = \frac{\sigma_k - \sigma_{p+1}}{\sigma_k + \sigma_1}, \quad ,$$

and T_{2s-1} is the $(2s-1)$ -th Chebyshev polynomial of the first kind.

Proof

Since the largest singular values of a matrix B are minus the smallest eigenvalues of $\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$, we obtain the desired result by applying Underwood's theorem [12, pg. 37] to $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^T & 0 \end{pmatrix}$.

We consider an example that shows how a proper choice of the block size p reduces the error bounds, and how $\sigma_i^{(s)}$, $1 \leq i \leq p$, generally approximates σ_i , $1 \leq i \leq p$, well even for a small s .

Let $\sigma_1 = 1.0$, $\sigma_2 = 0.9$, $\sigma_3 = 0.5$, and $\tau = \cos 0.1$. Let $ps \leq 10$. We shall see in section 3 how the available computer storage places an upper bound on the value ps . If we choose $p = 1$, $s = 10$, then

$$\tan^2 \epsilon = \frac{1-0.1^2}{0.1^2} = 99 ,$$

$$\gamma_1 = \frac{1.0-0.9}{2(1.0)} = 0.05 ,$$

$$\frac{1+\gamma_1}{1-\gamma_1} \doteq 1.105 ,$$

$$T_{19}(1.105) \doteq 2.8 \times 10^3 ,$$

and
$$\epsilon_1^2 \doteq \frac{2 \times 99}{(2.8 \times 10^3)^2} \doteq 2.5 \times 10^{-5} ;$$

whereas if $p = 2$, $s = 5$, then

$$\gamma_1 = \frac{1.0 - 0.5}{1.0 + 1.0} = 0.25 ,$$

$$Y_2 = \frac{0.9 - 0.5}{0.9 + 1.0} \doteq 0.21 ,$$

$$\frac{1+\gamma_1}{1-\gamma_1} = \frac{1.25}{0.75} \doteq 1.67 ,$$

$$\frac{1+\gamma_2}{1-\gamma_2} \doteq \frac{1.21}{0.79} \doteq 1.53 ,$$

$$T_9(1.67) \doteq 10^4 ,$$

$$T_9(1.53) \doteq 3.7 \times 10^3 ,$$

and

$$\epsilon_1^2 = \frac{2 \times 99}{10^8} \doteq 2.0 \times 10^{-6} ,$$

$$\epsilon_2^2 \doteq \frac{1.9 \times 99}{(3.7 \times 10^3)^2} \doteq 1.4 \times 10^{-5} .$$

We see that for the block method, we can expect a more accurate

approximation to σ_1 and we note that σ_2 is computed to the same accuracy as σ_1 when $p = 1$.

2.4 Reorthogonalization

We have shown that the $\{P_i\}$ and $\{Q_i\}$ are two sequences of orthogonal matrices. But the property holds only in exact arithmetic. In practice, the two sequences lose orthogonality very rapidly due to cancellation errors in the Lanczos steps:

$$\left\{ \begin{array}{l} \hat{Q}_i := A^t P_{i-1} - Q_{i-1} A_{i-1}^t, \\ \hat{P}_i := A Q_i - P_{i-1} B_{i-1}^t, \end{array} \right. \quad 2 \leq i \leq s.$$

A remedy is to reorthogonalize $P_i(Q_i)$ with respect to $P_j(Q_j)$, $j < i$, as soon as $P_i(Q_i)$ is computed.

The loss of orthogonality does not have adverse effects on the accuracy of the computed singular values (Paige [8]). But their multiplicities are questionable because once orthogonality is lost, the Lanczos method essentially restarts and recomputes the singular values that it has already computed. Reorthogonalization apparently stabilizes the Lanczos process but its cost in machine time is high. The cost in storage may even be prohibitive, for all the $\{P_j\}$ and $\{Q_j\}$ must now be stored in core. The Lanczos method without reorthogonalization allows us to keep only the most recently computed P_i and Q_i in memory and store the others on disk or magnetic tape.

Partial reorthogonalization, i.e. reorthogonalization of $P_i(Q_i)$ with respect to only some of the previously computed P_j 's (Q_j 's), looks promising too. It appears that just reorthogonalizing $P_i(Q_i)$

with respect to $P_{i-1}(Q_{i-1})$ may reduce the effects of cancellation errors present in the computation of $P_i(Q_i)$ and help preserve orthogonality at a very low cost in machine time and storage.

We have tacitly assumed that we can carry out the Lanczos iterations for s steps. Clearly this may not always be the case. We decide to check the length of each column of $P_i(Q_i)$ as soon as it has been generated in the QR factorization. If a column has a Euclidean length less than some tolerance, chosen in the program as the square root of the machine precision, it is set equal to the zero vector. We thus eliminate the errors caused by normalizing vectors consisting of numerical roundoffs to unit Euclidean length.

Before a Lanczos iteration begins, our program checks the starting matrix Q_1 for columns of all zeros. It first replaces any such columns with columns of random numbers and then orthonormalizes the resultant matrix. In this way, our program can restart itself even after linear independence has been lost. Since the work to check for columns of all zeros is prohibitive, we check for zero singular values computed in the previous iteration instead, assuming that they are caused only by columns of all zeros. Since our problem is to **compute** the few (usually < 10) largest singular values of a matrix of large order (usually > 1000), it is extremely unlikely that a desired singular value is zero.

2.5 Computation of Singular Values and Vectors of $J^{(s)}$

We now wish to compute the singular values and vectors of the $p_s \times p_s$ block bidiagonal matrix $J^{(s)}$:

$$X^{(s)} \Sigma^{(s)} Y^{(s)T} = J^{(s)} \quad .$$

In the rest of this section we shall omit the superscript s . from $J^{(s)}$ and denote its order by $t = ps$. Since the $p \times p$ blocks which form the block diagonal of J are upper triangular and the $p \times p$ blocks which form the block superdiagonal are lower triangular, we see that the blocks all fit together to form an upper triangular band matrix, dense within the band and with bandwidth (number of super-diagonals) equal to p . The rest of this section treats the problem of computing the singular values and vectors of an upper triangular band matrix J . The case where the vectors are not required is also considered since this section may be useful outside the block Lanczos context.

The method consists of two phases. The first phase reduces J to bidiagonal form by a finite sequence of orthogonal transformations. The problem of doing this efficiently is the main subject of this section. The singular values of A are preserved under the transformations. The second phase reduces the bidiagonal form to diagonal form by a modified version of the QR algorithm. This process is described in detail in Golub and Reinsch [6] and will not be discussed any further here. The singular values of J are the final diagonal elements, and the matrices of left and right singular vectors are the products of all the left and right transformations (*resp.*) used in the two phases of the reduction.

We are left with the first phase, reducing J to bidiagonal form. The methods of Givens and Householder for reducing a full symmetric matrix to tridiagonal form preserving eigenvalues are well known and described for example in Wilkinson [14]. In order to preserve eigenvalues, the same elementary transformations (either Givens or Householder) are applied to both the left and right sides of J to reduce it to

tridiagonal form. A similar method for reducing a nonsymmetric matrix to bidiagonal form preserving singular values (but not of course **eigenvalues**) is described in Golub and Reinsch [5]. Singular values are preserved when different elementary transformations are applied to the left and right sides of A. Golub and Reinsch use Householder transformations, but Givens transformations could easily be used instead. For the reduction of a full matrix to bidiagonal or tridiagonal form the method of Householder is about twice as fast as the method of Givens. However in 1972 Gentleman [3] showed how "fast Givens" transformations can be implemented. These are also described in Van Loan [13], and it appears that there is now little difference in the speed or effectiveness of the two methods.

Reducing a symmetric band matrix to tridiagonal form in a straightforward manner immediately fills in the zeros off the band. Rutishauser [9] shows how this may be avoided and the reduction completed while preserving the band structure, using either Givens or Householder transformations. Here we describe how to generalize this to the reduction of an upper triangular band matrix to bidiagonal form (in general a similar process would apply to any **nonsymmetric** band matrix).

Recall that a Givens transformation matrix $P^{(i,j)}$ is given by

$$P^{(i,j)} = \begin{matrix} & & i & & j & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & & & 1 & & \\ i & & & & c & & d \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ j & & & & -d & & c \\ & & & & & & 1 \\ & & & & & & \ddots \\ & & & & & & & 1 \end{matrix} \quad \text{where } c^2 + d^2 = 1 \text{ and has the property}$$

that if it is applied to J on the left then the resulting matrix $J' = P^{(i,j)}J$ has elements different from J only in rows i and j,

with zeros in both rows in those columns where there were zeros in both before, and, if c and d are chosen appropriately, with its (i,j) element equal to zero. Let us write $J = (\gamma_{ij})$, $J' = (\gamma'_{ij})$. Then in particular we have

$$\gamma'_{ik} = c\gamma_{ik} + d\gamma_{jk} \quad (1 \leq k \leq t)$$

$$\gamma'_{jk} = -d\gamma_{ik} + c\gamma_{jk}$$

so $\gamma'_{ji} = 0$ if $c = \gamma_{ii}/\sqrt{\gamma_{ii}^2 + \gamma_{ji}^2}$, $d = \gamma_{ji}/\sqrt{\gamma_{ii}^2 + \gamma_{ji}^2}$. The price paid for the annihilation is that a new nonzero element appears in one row wherever there was one already in the other. We say that row j is rotated against row i by the transformation. Similarly if $P^{(i,j)}$ is applied on the right only columns i and j of J are changed with $\gamma'_{ij} = 0$ if c and d are chosen correctly.

To describe the reduction process let us suppose that J is an upper triangular band matrix with order $t = 11$ and $p = 4$ superdiagonals. Then the first thing the algorithm does is to zero γ_{15} by multiplying J on the right by $P^{(4,5)}$ with c and d chosen correctly, or in other words by rotating column 5 against column 4. This introduces one new non-zero element γ'_{54} . This new element is annihilated by multiplying J' on the left by $P^{(4,5)}$, that is by rotating row 5 against row 4. This in turn introduces a new non-zero element γ'_{49} . Two more transformations, one from the left and one from the right, are now required to completely "chase the element off the matrix". At this point the resulting matrix has the same zero pattern as the original matrix J except that γ_{15} has been annihilated. Now the process is repeated for γ_{14} and then for γ_{13} , and then the first row has the desired bidiagonal form.

Finally, the entire process is repeated for every row until the matrix becomes **bidiagonal**. The method is illustrated in Figure 1. Let us call this method Band Givens I.

Reducing the matrix to bidiagonal form in this way requires approximately $4pt^2$ multiplications using ordinary Givens transformations, or $2pt^2$ using "fast Givens", assuming $1 \ll p \ll t$. This compares with a count of approximately $4t^3/3$ multiplications required to do the reduction by the standard Golub-Reinsch algorithm using Householder transformations and ignoring the band structure, filling in the zeros off the band. This is of course a big savings if $p \ll t$ as assumed, and furthermore only pt storage locations are required to store the band matrix while t^2 storage locations are required for the standard Golub-Reinsch reduction. If left and right singular vectors are required however, the rotations used in Band Givens I must be accumulated as the computation proceeds. This requires $4t^3$ multiplications using ordinary Givens transformations or $2t^3$ using "fast Givens", as opposed to $8t^3/3$ multiplications for the Golub-Reinsch reduction, so that if the vectors are required, Band Givens I still requires less multiplications than Golub-Reinsch if the fast Givens transformations are used. Both methods require approximately $2t^2$ storage locations.

There are several other possible methods to reduce J to bidiagonal form. The method we ~~shall~~ call Band Givens II applies a sequence of rotations to J as before, but instead of reducing each row in turn to two elements, it systematically **reduces** the bandwidth by zeroing each superdiagonal in turn. In other words, in the example presented in Figure 1, after zeroing γ_{15} and chasing it off the matrix, it next turns to γ_{26} instead of γ_{14} . This method requires more rotations, since the

FIGURE 1.

Bidiagonalizing a Pentadiagonal Upper Triangular Matrix of Order 11

Using Givens Rotations by the Method Band Givens I

$$\begin{bmatrix} x & x & c & b & a & & & & & & \\ & x & x & x & x & x & c & & & & \\ & & c & x & x & x & x & x & b & & \\ & & & b & x & x & x & x & x & a & \\ & & & & a & x & x & x & x & x & \\ & & & & & x & x & x & x & x & c \\ & & & & & & c & x & x & x & x \\ & & & & & & & b & x & x & x \\ & & & & & & & & a & x & x \\ & & & & & & & & & x & x \\ & & & & & & & & & & x \end{bmatrix}$$

STEP 1:

(i) Zero γ_{15} and chase it a a a off the matrix:

Rotate col. 5 against col. 4 to zero γ_{15} and introduce γ'_{54} .

Rotate row 5 against row 4 to zero γ'_{54} and introduce γ'_{49} .

Rotate col. 9 against col. 8 to zero γ'_{49} and introduce γ'_{98} .

Rotate row 9 against row 8 to zero γ'_{98}

- chased off

(ii) Zero γ_{14} and chase it b b b off the matrix similarly.

(iii) Zero γ_{13} and chase it c c c also.

STEP 2: Repeat for the second row - etc.

decreasing bandwidth causes more **nonzero** elements to be introduced before a certain element is chased off the matrix, but for the same reason each rotation is less work if the vectors are not required. The two considerations cancel each other out so that Rand Givens I and II require about the same number of multiplications if vectors are not required, but the latter is slower by a factor of about $\ln p$ if vectors are required.

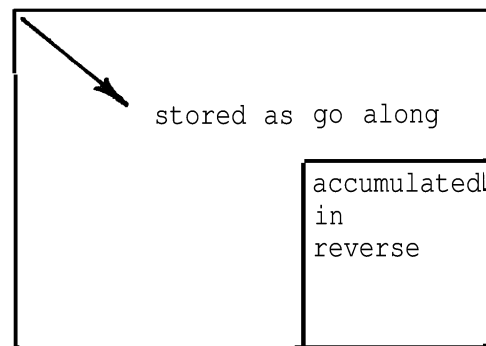
Let us consider now a method we shall call Rand Householder. This follows an idea suggested in Rutishauser [9] for the corresponding eigenvalue tridiagonal reduction problem. Recall that a Householder transformation matrix $Q^{(i,j,p)}$ can be chosen to have the property that when applied to A on the left the resulting matrix $A' = Q^{(i,j,p)}A$ has zeros in positions $i+1, \dots, j$ of column p but is different from A only in rows i, \dots, j and has zeros in all rows in those columns where there were zeros in all before. As before the role of rows and columns is reversed when the transformation is applied on the right. Let us describe the algorithm for the $t = 11, p = 4$ case again. The first step is to zero all of a_{12}, a_{13}, a_{14} simultaneously by applying a Householder transformation $Q^{(2,4,1)}$ to A on the right. Instead of introducing one new non-zero element as in the first step of the algorithm using Givens transformation, this introduces a whole lower triangle (of order 3) of non-zero elements. This is annihilated by a sequence of 3 Householder transformations (the last a degenerate one) which introduces another upper triangle on the other side of the band. The triangle is chased off the matrix, as the single element was before, after another two repetitions of this. However a little thought will make it clear that the extra triangle of elements

at every step makes the method much less efficient than Band Givens I - indeed, it introduces an extra factor of p in the number of multiplications required, whether or not vectors are needed.

There is yet another possible approach, which we call the triangle Givens method -- it does not attempt to preserve the band structure, but does preserve the upper triangle structure. It is considered in Chan [2] for finding the singular values of an upper triangular matrix. In this method elements are eliminated row by row in the upper triangle using column rotations, and after each column rotation one row rotation is applied to move the **nonzero** element introduced in the lower triangle back up to the upper triangle. Since the upper triangle is filled in, this method requires more multiplications than Band Givens I. If fast Givens transformations are used and no vectors are required the number of **multiplications** required for Triangle Givens is less than for Golub-Reinsch, but if vectors are required they are the same.

Finally we describe a rather complicated variant of Band Givens I which we call Band Givens III, which requires less multiplications when vectors are required. In the standard Golub-Reinsch algorithm Householder transformations are used to eliminate elements, but instead of accumulating the transformations directly the transformations are stored in place of the elements just annihilated and after the reduction is complete they are then accumulated in reverse order. The reason for this is that when they are accumulated in forward **order**, the j^{th} transformation on either the left or the right, having been chosen to annihilate $t-j$ elements of the j^{th} column or row of J , will affect $(t-j)t$ elements of the $t \times t$ matrix of transformations so far accumulated, whereas when they are accumulated in reverse order the same

transformation need only be applied to the $(t-j) \times (t-j)$ matrix of transformations so far accumulated. This eliminates one third of the multiplications needed. This trick is also employed in computing a tridiagonal reduction for eigenvalue problems or the complete QR factorization of a matrix using Householder transformations. When Givens transformations are used in the band eigenvalue problem however they are always accumulated in the forward direction as the reduction proceeds although the same savings potential exists if they are accumulated in reverse. Storing **all** the transformations used in Band Givens I would be a complicated task, but it is by no means impossible. The method requires approximately $t^2/2$ transformations each on the left and the right, and since each transformation can be stored in and recovered from one storage location (see Stewart [11]), all the transformations may be stored in the two $t \times t$ arrays in which they are to be accumulated. **Furthermore** they can be accumulated one by one in reverse order without disturbing the transformations stored but not yet accumulated, since the number of transformations required to reduce the first j rows to bidiagonal form is approximately $t^2/2 - (t-j)^2/2$ on each side which may be stored with room to spare without being disturbed by the two $(t-j) \times (t-j)$ submatrices needed to accumulate the transformations operating on rows $j+1$ through t :



However the storing and retrieving of these transformations would indeed be an arduous task, and although Band Givens III requires only $8t^3/3$ multiplications using ordinary Givens transformations and $4t^3/3$ using fast Givens, the big question is whether it would still be worthwhile with all the extra bookkeeping.

Thus the best method seems to be either Band Givens I or III, but we should make some disclaimers. These results are only valid assuming $1 \ll p \ll t$ which may not be the case. Multiplications are not the whole story, since indexing operations also take time and on modern machines multiplications do not take much more time than indexing. Of course the amount of overhead required by a method is also important. Another thing to note is that the second phase reducing the bidiagonal form to diagonal form to obtain the singular values typically takes $8t^3$ multiplications using ordinary Givens transformations or $4t^3$ using fast Givens so that this may dominate any slight savings in the reduction phase. Of course no final conclusion about which method is best can be made without extensive numerical tests.

The multiplication counts for the different methods are summarized in Table I.

TABLE I

Summary of approximate multiplication counts for the different algorithms to reduce a band matrix with order t and bandwidth p to bidiagonal form, using ordinary Givens transformations except as noted in the last column. It is assumed that $1 \ll p \ll t$, but even for small p the same conclusions regarding the relative efficiency of the methods results except that there is then little difference between Band Givens I and II.

<u>Method</u>	<u>Reduction Without Vectors</u>	<u>Accumulating the Vectors</u>	<u>Total When Vectors Desired</u>	<u>Total Using Fast Givens</u>
Golub-Reins ch	$\sum_{i=1}^{t-2} 2(2)(t-i)^2 \sim \frac{4t^3}{3}$	$\sum_{i=1}^{t-2} 2(2)(t-i)^2 \sim \frac{4t^3}{3}$	$\frac{8t^3}{3}$	---
Band Givens I	$\sum_{i=1}^{t-2} \sum_{k=2}^p \frac{t-(i+k)+1}{p} (2)(4)(p+1) \sim 4pt^2$	$\sum_{i=1}^{t-2} (p) \frac{t-i}{p} (8)t \sim 4t^3$	$4t^3$	$2t^3$
Band Givens II	$\sum_{k=2}^p \sum_{i=1}^{t-2} \frac{t-(i+k)+1}{k} (8)(k+1) \sim 4pt^2$	$\sum_{k=2}^p \sum_{i=1}^{t-2} \frac{t-i}{k} (8)t$ $\sim 4t^3 \sum_{k=2}^p \frac{1}{k} \sim 4(\ln p - \frac{1}{2})t^3$	$4(\ln p - \frac{1}{2})t^3$	$2(\ln p - \frac{1}{2})t^3$
Band Householder	$\sum_{i=1}^{t-2} \frac{t-i}{p} (2) \sum_{k=1}^{p-1} 2(k)(k+p) \sim \frac{5}{3} p^2 t^2$	$\sum_{i=1}^{t-2} \frac{t-i}{p} (2) \sum_{k=1}^p 2(k)t \sim pt^3$	pt^3	---
Triangle Givens	$\sum_{i=1}^{t-2} 4(t-i)^2 \sim \frac{4t^3}{3}$	$\sum_{i=1}^{t-2} (t-i)8t \sim 4t^3$	$\frac{16t^3}{3}$	$\frac{8t^3}{3}$
Band Givens III	$4pt^2$	$\sum_{i=1}^{t-2} (p) \frac{t-i}{p} (8)(t-i) \sim \frac{8t^3}{3}$	$\frac{8t^3}{3}$	$\frac{4t^3}{3}$

2.6 Convergence Tests

Let us examine what we have done so far. We apply the **Lanczos** method to generate a block bidiagonal matrix $J^{(s)}$ from the matrix \bar{A} :

$$P^t \bar{A} Q = J^{(s)}$$

where $P = (P_1, P_2, \dots, P_s)$,

and $Q = (Q_1, Q_2, \dots, Q_s)$.

Then we compute the singular value decomposition of $J^{(s)}$:

$$J^{(s)} = X^{(s)} \Sigma^{(s)} Y^{(s)t}$$

By considering the matrices $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$ and $\begin{pmatrix} 0 & J^{(s)} \\ J^{(s)t} & 0 \end{pmatrix}$, we can verify that

$$\begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix} \begin{pmatrix} X^{(s)} \\ Y^{(s)} \end{pmatrix} = \begin{pmatrix} PX^{(s)} \\ QY^{(s)} \end{pmatrix}$$

are the eigenvectors of the matrix $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$ restricted to the subspace spanned by the columns of $\begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix}$.

We have seen that the p smallest eigenvalues of $\begin{pmatrix} 0 & J^{(s)} \\ J^{(s)t} & 0 \end{pmatrix}$ are usually accurate approximations to those of $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$, in which case it can be shown that the p corresponding eigenvectors of $\begin{pmatrix} 0 & J^{(s)} \\ J^{(s)t} & 0 \end{pmatrix}$, when premultiplied by $\begin{pmatrix} P & 0 \\ 0 & Q \end{pmatrix}$, are also good approximations to those of $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$, albeit not to as high an accuracy.

Our convergence test uses Weinstein's inequality [14, pp. 170-171], which states that for a symmetric matrix A and a vector \tilde{x} of unit length, if

$$\|A\tilde{x} - \mu\tilde{x}\| = \delta$$

for some scalar μ , then there is an **eigenvalue** λ of A such that

$$|\lambda - \mu| \leq \delta .$$

Let $\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}$ be the i -th column of $\begin{pmatrix} PX^{(s)} \\ QY^{(s)} \end{pmatrix}$. Then

$$\begin{aligned} \left\| \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} - \sigma_i^{(s)} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} \right\|^2 &= \left\| \begin{pmatrix} A\tilde{v} \\ A^t\tilde{u} \end{pmatrix} - \sigma_i^{(s)} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} \right\|^2 \\ &= \|A\tilde{v} - \sigma_i^{(s)}\tilde{u}\|^2 + \|A^t\tilde{u} - \sigma_i^{(s)}\tilde{v}\|^2 . \end{aligned}$$

Assume ϵ is the user-supplied error tolerance for the singular values.

If

$$\left(\|A\tilde{v}_i - \sigma_i^{(s)}\tilde{u}_i\|^2 + \|A^t\tilde{u}_i - \sigma_i^{(s)}\tilde{v}_i\|^2 \right)^{1/2} \leq \epsilon \sigma_i^{(s)} ,$$

then there is a singular value of A within relative error ϵ of $\sigma_i^{(s)}$ and we may accept $\sigma_i^{(s)}$ as a singular value of A . (If $\sigma_i^{(s)}$ is less than one we use ϵ as an absolute error tolerance instead.)

We note that in our algorithm the **computed** singular values and vectors are converging to the singular values and vectors of \tilde{A} and not of A . Thus if we compute the residuals with respect to A and not to \tilde{A} , there is a lower bound to their values. We take this error

into account by adding to ϵ the residuals corresponding to the accepted singular values. To avoid an error tolerance that is close to the machine precision, we add to ϵ a third term combining the machine *precision $mcheps$ and the matrix dimensions m and n . Thus, if

$$\tau_k^2 = \|A\tilde{v}_k - \sigma_k^{(s)} \tilde{u}_k\|^2 + \|A^t \tilde{u}_k - \sigma_k^{(s)} \tilde{v}_k\|^2, 1 \leq k \leq i-1,$$

then

$$\hat{\tau}_i := \epsilon + \left(\sum_{k=1}^{i-1} \tau_k^2 \right)^{1/2} + 10 \times (m+n) \times mcheps,$$

where $mcheps = 2.20 \times 10^{-16}$ for double precision arithmetic on the IBM Systems 360 and 370. We shall accept $\sigma_i^{(s)}$ as a singular value of A if

$$\left(\|A\tilde{v}_i - \sigma_i^{(s)} \tilde{u}_i\|^2 + \|A^t \tilde{u}_i - \sigma_i^{(s)} \tilde{v}_i\|^2 \right)^{1/2} \leq \hat{\tau}_i \sigma_i^{(s)}$$

2.7 Updating p and s

We shall see in section 3 how the available computer memory places an upper bound on the product ps . We wish to determine optimal values for p and s subject to this constraint. We can see from the error bounds in section 2.3 that such choices are dependent on the singular value spectrum of A and thus are usually not *possible a priori without further information.

We shall discuss initial choices of p and s in section 3.3. We are concerned here with updating p and s after some singular values and vectors have converged.

We assume that before the current Lanczos iteration the block size

is p_0 , the step size is s_0 , and the bound on $p_0 s_0$ is q_0 .

Assume that g singular values are to be computed and $g_0 (1 \leq g_0 < g)$ singular values have been **computed** and accepted in the current iteration.

Our problem is to choose the new block size p_1 and step size s_1 .

Our strategy is that if $p_0 \geq g$, then

$$p_1 := p_0 - g_0 ,$$

and
$$s_1 := \left\lfloor \frac{q_0 - g_0}{p_1} \right\rfloor . \quad (\text{Here } \lfloor a \rfloor \text{ denotes the integer part of } a.)$$

The rationale is that if the user chooses a block size greater than the number of singular values desired, he must have a good reason, e.g. he may have chosen the block size to be the number of singular values in the cluster of largest singular values. We wish to preserve the user's choice of block size in this case.

If $p_0 < g$, then we pick p_1 to be the smaller of the current block size and the number of singular values remaining to be computed.

Thus,

$$p_1 := \min(p_0, g - g_0) ,$$

$$s_1 := \left\lfloor \frac{q_0 - g_0}{p_1} \right\rfloor .$$

We test s_1 to see if $s_1 \geq 2$. If it is not, then we set

$$p_1 := \left\lfloor \frac{q_0 - g_0}{2} \right\rfloor ,$$

$$s_1 := \left\lfloor \frac{q_0 - g_0}{p_1} \right\rfloor .$$

We note that the step size must be at least 2 to carry out the Lanczos method.

2.8 Complete Algorithm

We have described one iteration of the Lanczos method. We do not expect to compute all the desired singular values in one iteration and so we shall iterate the method with improving starting matrices. We saw in section 2.6 that the first p_0 columns of QY are usually better approximations than Q_1 to the p_0 right singular vectors corresponding to the p_0 largest singular values of A . If $g_0 = 0$, then those p_0 columns of QY will serve as a good starting matrix for another Lanczos iteration. If $g_0 > 0$, then the $(g_0+1), \dots, (g_0+p_1)$ -th columns will be chosen as the starting matrix for the next iteration. We have seen that the $(g_0+1), \dots, p_0$ -th columns of QY are usually good approximations to the $(g_0+1), \dots, p_0$ -th right singular vectors of A . Our experimental results show that the other columns are usually rich in the direction of the $(p_0+1), \dots, p_1$ -th right singular vectors of A .

We see that the convergence test in section 2.6 involves multiplications by A and A^t ; so we wish to avoid performing the test unless we think some of our singular values have converged. A good test is to look at the relative increase of the largest singular value from the previous iteration. We perform the convergence test only if the relative increase is less than the user supplied tolerance ϵ . The criterion is good in that we shall seldom overshoot the desired accuracy, because if the convergence test is satisfied, the computed singular values, as **Rayleigh** quotients, are likely to have errors proportional to ϵ^2 unless they are poorly separated.

Our complete block Lanczos algorithm follows:

Algorithm

1. Start with an arbitrary $n \times p$ matrix Q_1
2. Orthonormalize the columns of Q_1 .
3. Apply the Lanczos method to compute the block bidiagonal matrix $J^{(s)}$ using Q_1 as the starting matrix:

$$P^t \bar{A} Q = J^{(s)} .$$

4. Compute the singular value decomposition of $J^{(s)}$:

$$X^{(s)} \Sigma^{(s)} Y^{(s)t} = J^{(s)} .$$

5. If the relative increase in the largest singular value of $J^{(s)}$ is less than ϵ , then perform the convergence test. Otherwise go to step 8.

6. Stop if all desired singular values have converged.

7. If one or more singular values have converged, update the values of p and s .

8. Take the first p columns of QY that have not been accepted as singular vectors as the starting matrix Q_1 for the next iteration. Go to step 2.

It appears that step 2 is unnecessary after the first iteration since both Q and Y are matrices consisting of orthonormal columns. Numerical experiments have shown, however, that the columns of QY are not necessarily orthonormal and we need to perform step 2 to maintain numerical stability.

3. Implementation

We have written a set of subroutines implementing our algorithm. We use the Bell Laboratory PFORT language, a subset of the ANS FORTRAN language.

Our routines use integer and double precision arithmetic. We have a subroutine that **computes** the inner product of two vectors. We would have obtained better numerical results had we accumulated inner products in higher precision. We recommend the usage of extended precision arithmetic to compute inner products if the work is done by the computer hardware. The additional cost is small and the results are more accurate. We have not incorporated the extended precision computations into our routines to provide program portability. Experiments show that the numerical results are still satisfactory without recourse to higher precision arithmetic.

Our routines usually need a large core to store the matrices X and Y. On an IBM System 360 or 370, the requirement is $(m+n) \times q \times 8$ bytes, which forces q to be small for large m and n; e.g. if $m = n = 1000$, then an available core of size 200K bytes would force q to be less than or equal to 12.

MAXVAL is our main routine that calls all the other subroutines.

3.1 Formal Parameters

(a) Quantities to be given to MAXVAL:

m,n : the dimensions of the matrix A; $2 \leq n \leq m \leq 1000$.
q : the number of vectors of length m contained in the array X; also the number of vectors of length n contained in array Y; $q \leq 26$ and $q \leq n$.

pinit : the initial block size; if **pinit** < 0, then **-pinit** becomes the block size and columns **h+1,...,h+(-pinit)** of **Y** are assumed to be initialized to a matrix to be used to start the Lanczos method.

g : the number of singular values and left and right singular vectors desired; $1 \leq g < q$.

mmax : the maximum number of matrix-vector products $A\tilde{x}$ and $A^t\tilde{x}$ allowed.

eps : the relative precision to which singular values and vectors will be computed; **eps** becomes an absolute tolerance if the singular value is less than one.

op : subroutine **op (m,n,p,u,v,orig)** computes $U = AV$ when **orig** is true, and $V = A^tU$ when **orig** is false; **U** is an $m \times p$ matrix and **V** is an $n \times p$ matrix; the input matrix must not be altered by the subroutine call.

h : the number of singular values and vectors already computed; if $h > 0$, then columns 1 through **h** of **X(Y)** must contain the left (right) singular vectors of **A**.

D : an array of length at least **q**.

x : an array of length at least $m \times q$.

Y : an array of length at least $n \times q$.

iorthg : the number of immediately preceding blocks of vectors with respect to which reorthogonalization of the present block of vectors is to be carried out.

lout : output unit number.

mccheps : machine precision, equals 2.2×10^{-16} for double precision arithmetic.

(b) Quantities produced by MAXVAL:

h : the total number of singular values and vectors computed including any already computed when MAXVAL was entered.

D : elements 1 to h of D contain the computed singular values.

x : the first m x h elements contain the left singular vector approximations--the first vector in the first m elements, the second in the next m elements, and so on.

Y : the first n x h elements contain the right singular vector approximations--the first vector in the first n elements, the second in the next n elements, and so on.

iecode : the error message;

= 0 : successful termination.

= 1 : $n < 2$.

= 2 : $n > m$.

= 3 : $m > 1000$.

= 4 : $g < 1$.

= 5 : $q \leq g$.

= 6 : $q > 26$.

= 7 : $q > n$.

= 8 : ~~mmax~~ is exceeded before g singular values and vectors have been computed.

3.2- Program Organization

MAXVAL is the main routine that calls all the other subroutines. It also checks the input data for inconsistencies. The main body of the subroutine begins by filling the appropriate columns of Y with

random vectors if a starting matrix is not provided. The random vectors are orthonormalized in a call to the subroutine ORTHOG. MAXVAL then calls BKLANC to carry out the block bidiagonalization of \tilde{A} and then SVBUTM to solve the singular value problem of the resulting block bidiagonal matrix $J^{(s)}$. Two calls to the subroutine ROTATE compute the matrices PX and QY. A test is then made of the relative increase in the largest singular value of $J^{(s)}$ to determine if it is necessary to call the convergence test routine CNVTST. If some but not all the desired singular values have converged, then the subroutine PCHOIC is called to choose new values for p and s for the next iteration, which begins with the first p columns of QY that have not been accepted as singular vectors as the starting matrix.

ORTHOG always reorthogonalizes the input vectors with respect to the vectors in the first h columns of the input matrix. Reorthogonalization is also carried out with respect to the previous IORTHG blocks of vectors. The resulting vectors are then orthormalized using a modified Gram-Schmidt method [1].

ORTHOG calls INPROD to compute inner products in the reorthogonalization process.

BKLANC implements the block Lanczos reduction. The banded upper triangular matrix $J^{(s)}$ is stored in columns 2 through p + 2 of the matrix C, the main diagonal being stored in the first ps elements of column 2, the upper diagonal being stored in the first ps - 1 elements of column 3, and so on.

SVBUTM is designed to solve the singular value problem of a banded upper triangular matrix. The matrix $J^{(s)}$ has been stored in the correct form in BKLANC for input into this routine. SVBUTM first calls

BIBAND to bidiagonalize $J^{(s)}$ using the algorithm Band Givens I described in section 2.5, and then SVDBI to apply the QR method to compute the singular values of the bidiagonal matrix. The routines ROTROW and ROTCOL implement Givens transformations to rotate rows and columns of $J^{(s)}$ to reduce it to a bidiagonal form--note however that an improvement here would be to implement fast Givens transformations instead. SVDBI calls DROTAT to compute the singular vectors of $J^{(s)}$.

ROTATE computes PX and QY, the left and right singular vectors of \bar{A} .

CNVTST tests the computed singular values and vectors for convergence. It tests first the largest singular value, then the second largest singular value, and so on until it finds either non-convergence or all the desired singular values.

PCHOIC computes new values for p and s if some but not all desired singular values have converged.

3.3 Numerical Properties

The user can easily modify the bounds on m and q by changing the storage allocation for the arrays C, U, V, R and T at the beginning of MAXVAL. The tests of the values of m and q must then be appropriately modified.

Our program has proved to be very efficient for large and sparse singular value problems. The convergence is very fast if the largest singular values are fairly well separated. Even in cases when the largest singular values are clustered, our program appears to be able to compute them accurately.

We have seen that the optimal choice of the block size depends on the singular value spectrum and is therefore not possible a priori. A "safe" choice appears to be choosing the block size as the number of desired singular values. The singular values thus computed are usually fairly accurate. A drawback is that sometimes this choice produces a very slow convergence rate.

We cannot overemphasize the importance of s . Storage limitations place a bound on the product ps . The two matrices X and Y require $(m+n) \times q$ storage locations, a significant amount for large m and n . Since q bounds $ps + h$, we see that the value of p uniquely determines the maximal value of s . Since s must be at least 2, the block size p will be reduced to give s the value of 2 or 3. Experiments have shown that $s = 2$ often produces intolerably slow convergence. It appears that we should always give s a value of at least 3. In fact, for a problem with a dense singular value spectrum, the best choice appears to be $p = 1$, $s = q - h$ and no reorthogonalization.

Reorthogonalization appears to be unnecessary if the singular value spectrum is dense. If the largest singular values are well separated from the rest, then complete reorthogonalization is required to keep **multiple images** of these singular values from appearing. Partial reorthogonalization, e.g. with $iorthg = 1$, is insufficient although it does produce better results than no reorthogonalization at all.

From the theorem in section 2.3, we can see that a good choice of the block size is the number of the dominating singular values. Experiments confirm the theory and we see also that it is better to

overestimate the number of dominating singular values than to under-estimate.

The use of extended precision arithmetic to accumulate inner products produces much more accurate results at an average cost of about 20% more computing time. We have, however, found its use to be unnecessary for a large value of ϵ ; we have obtained satisfactory results from 1000×999 matrices with $\epsilon = 10^{-3}$ using only double precision arithmetic.

4. Test Examples

We have chosen rectangular diagonal matrices in all but one test examples. We feel diagonal matrices are sufficiently general because we do not transform the given matrix; we obtain information about the given matrix only through the subroutine that computes the product of the matrix (or its transpose) with an input matrix. Diagonal matrices are convenient in that we know the singular value spectrum and so can study the behavior of our algorithm as a function of the block and step sizes.

We have run our program on an IBM 370/168 computer using the EXTENDED FORTRAN H compiler. Our program takes 6.95 seconds to compile.

In the examples below the following rotation is used:

$m(-n) = m \times 10^{-n}$
iter = total number of iterations
imm = total number of matrix-vector multiplications
i w = total number of vector inner products in the
orthogonalization process
exec time = execution time in seconds on the machine

Example 1

A is a 1000 X 999 matrix with diagonal elements 0.006, -0.007, 0.008, -0.009, ..., 1.000, and 2, 2, 2 and -10. With $g = 4$, $q = 12$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$, we obtain the following results.

	p = 1	p = 2	p = 3	p = 4	p = 5	p = 6
σ_1	10 + 1(-15)	10 - 4(-15)	10 - 1(-10)	10 - 2(-7)	10 - 5(-12)	10 - 4(-12)
σ_2	10 - 1(-15)	2	2 + 4(-12)	2 + 2(-9)	2 - 3(-8)	2 - 2(-8)
σ_3	2 - 6(-15)	2 - 8(-15)	2 - 1(-9)	2 - 3(-11)	2 - 1(-7)	2 - 6(-8)
σ_4		2 - 3(-9)	2 - 1(-8)	2 - 4(-10)	2 - 9(-7)	2 - 6(-7)
iter		5	3	3	5	5
imm		105	67	62	85	100
ivv		224	114	108	200	300
exec time	program fails to terminate	6.06	3.81	3.71	5.33	7.34

We see the advantage of a block algorithm in this example. The point algorithm gives a double image for the singular value 10 and then fails to terminate because it converges to a value 2.738. We obtain the fastest convergence using $p = 4$, as we expect. Note the high accuracy in the solution values with $\text{eps} = 10^{-3}$.

Example 2

A is a 1000 X YYY matrix with diagonal elements -0.005, 0.006, -0.007, 0.008, ..., 1.000, and 2, -2 and 2. We choose $g = 3$, $q = 12$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$.

	p = 1	p = 2	p = 3	p = 4
σ_1	2	$2 \cdot 2 \times 10^{-15}$	$2 \cdot 2 \times 10^{-11}$	$2 \cdot 4 \times 10^{-11}$
σ_2	2	$2 \cdot 1 \times 10^{-9}$	$2 \cdot 3 \times 10^{-11}$	$2 \cdot 6 \times 10^{-11}$
σ_3	$2 \cdot 1 \times 10^{-15}$	$2 \cdot 2 \times 10^{-8}$	$2 \cdot 3 \times 10^{-10}$	$2 \cdot 5 \times 10^{-10}$
iter	5	4	2	3
imm	115	89	52	70
i w	124	132	48	108
exec time	6.52	5.24	3.18	4.70

In this example, we see again the advantage of a properly chosen block size. Note also the better results obtained by overestimating rather than underestimating the number of dominating singular values.

Example 3

A is a 1000 X 999 matrix with diagonal elements 0.006, -0.007, 0.008, -0.009, ..., 1.000, and 2, 10, -10 and 10. We choose $g = 3$, $q = 6$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$.

	p = 1	p = 2	p = 3
σ_1	$10 \cdot 2 \times 10^{-15}$	$10 \cdot 3 \times 10^{-15}$	$10 \cdot 4 \times 10^{-12}$
σ_2	$10 \cdot 3 \times 10^{-15}$	$10 \cdot 2 \times 10^{-9}$	$10 \cdot 2 \times 10^{-11}$
σ_3	$2 \cdot 5 \times 10^{-7}$	$10 \cdot 2 \times 10^{-7}$	$10 \cdot 2 \times 10^{-8}$
iter	4	6	3
imm	42	56	37
i w	48	82	36
exec time	1.85	2.46	1.76

We see the failure of the point algorithm to obtain the third singular value 10. This example also shows how fast our algorithm can be even with very limited storage ($q = 6$) as long as the separation of the singular values is good.

Example 4

A is the same matrix as in Example 1. But we choose $g = 3$, $q = 12$, $\text{eps} = 10^{-3}$ and $p = 1$. We run our program with no, partial, and complete reorthogonalization.

	iorthg = 0	iorthg = 1	iorthg = 12
σ_1	$10+1 \times 10^{-14}$	10	10
σ_2	$10-1 \times 10^{-15}$	$10-2 \times 10^{-12}$	$2-2 \times 10^{-15}$
σ_3	$2-6 \times 10^{-15}$	$2-4 \times 10^{-15}$	$2-4 \times 10^{-15}$
iter	1	1	3
imm	31	31	71
i w	0	22	392
exec time	1.69	1.77	5.27

We see only ~~complete~~ reorthogonalization gives the correct solution. We also see that the block algorithm (Example 1) with $p = 3$ and 4 and no reorthogonalization computes four singular values correctly in ~~25%~~ less machine time.

We also run the first case ($iorthg = 0$) using extended precision arithmetic to accumulate inner products. The results are unfortunately unchanged.

Example 5

A is a 1000 X 999 matrix with diagonal elements 0.002, -0.003, 0.004, -0.005, ..., 1.000. We choose $g = 3$, $q = 12$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$.

	p = 1	p = 2	p = 3
σ_1	0.999992	0.999992	0.999986
σ_2	0.998960	0.998951	0.998999
σ_3	0.998036	0.998005	0.997980
iter	13	33	27
imm	305	711	609
i w	190	784	676
exec time	17.59	41.20	38.09

This is an example where a point algorithm is a good choice. The denseness of the singular value spectrum takes away the virtues of a block algorithm; the best choice is therefore to maximize s .

Example 6

A is a 314 X 80 matrix obtained from earthquake research and is of the following special form:

$$A = (A_1 | A_2),$$

where A_1 is 314 X 24 and block diagonal,

and A_2 is 314 x 56 and randomly sparse.

A_1 consists of six diagonal blocks, whose dimensions are 53 X 4, 51 X 4, 46 X 4, 58 X 4, 55 X 4 and 51 X 4. There are about 4 non-zero

elements per row in A_2 and a total of 2509 non-zero elements in A.

We store only the non-zero elements of A. We use three one-dimensional arrays IINDEX, JINDEX and A, each of length 2509, to store i, j and a_{ij} . This compact storage scheme also enables us to compute the matrix-vector products $A\tilde{x}$ and $A^t\tilde{y}$ efficiently.

Assume A is $m \times n$ and has NDATA non-zero elements. Then the following FORTRAN statements compute $\tilde{x} = Ay$:

```

DO 10 K = 1,M
    X(K) = 0.DO
10  CONTINUE
DO 20 K = 1,NDATA
    I = IINDEX(K)
    J = JINDEX(K)
    X(I) = X(I) + A(K) * Y(J)
20  CONTINUE

```

The following statements compute $\tilde{y} = A^t\tilde{x}$:

```

DO 110 K = 1,N
    Y(K) = 0.DO
110 CONTINUE
DO 120 K = 1,NDATA
    I = IINDEX(K)
    J = JINDEX(K)
    Y(J) = Y(J) + A(K) * X(I)
120 CONTINUE

```

A full singular value decomposition of A was computed using the subroutine SVD in EISPACK [10]. The demand on storage is excessive, for we need to supply at least $2 \times m \times n \times 8$ bytes (≈ 393 K bytes) if we want the singular vectors. The execution time was 23.18 seconds. The main disadvantage of SVD is its inflexibility: we always have to compute all the singular values whether or not we need all of them. Our Lanczos program, on the other hand, requires only $(m+n) \times q \times 8$ bytes (≈ 31 K bytes for $q = 10$) if we give it q vectors of

storage to compute the singular vectors. It can then compute up to $(q - 1)$ singular values and corresponding vectors. We need $2509 \times (4 + 4 + 8)$ bytes ($\doteq 40$ K bytes) to store A using our compact scheme.

The following table summarizes our results when we apply our program on A with $p = 1$, $q = 10$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$:

g	1	2	3	4	5	6	7	8	9
iter	1	2	3	5	7	9	12	18	23
imm	23	44	63	93	119	141	166	206	229
i w	0	18	50	134	230	330	474	726	886
exec time	1.66	2.21	2.73	3.61	4.38	5.08	5.92	7.23	7.9%

All our computed results agree to at least 6 significant digits with the values from SVD, agreeing with the expectation that the accuracy is $O(\text{eps}^2)$.

The 80 non-trivial singular values of A are (to 3 significant digits) 12.6, 9.53, 8.87, 8.06, 7.77, 7.59, 6.42, 5.54, 5.16, 4.49, ..., 1.28×10^{-2} , 4.45×10^{-7} , 1.91×10^{-7} , 5.93×10^{-8} and 2.48×10^{-15} . Although the largest singular values of A are quite uniformly distributed, we observe a uniform improvement in program speed when we choose the block size equal to 2, i.e. $p = 2$, $q = 10$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$:

g	1	2	3	4	5	6	7	8	9
iter	1	2	3	5	7	8	10	18	24
imm	22	43	59	88	109	121	136	190	214
i w	10	28	66	158	248	304	394	798	990
exec time	1.62	2.13	2.49	3.34	3.83	4.16	4.62	6.28	7.15

The effect of storage space on program speed is examined using both 12 and 15 vectors of storage to determine 9 singular values. The results with $p = 1$, $\text{eps} = 10^{-3}$ and $\text{iorthg} = 0$ are:

q	10	12	15
iter	23	9	5
imm	229	129	123
i w	886	530	530
exec time	7.99	5.21	4.99

The trade-offs between space and time are obvious.

References

- [1] Björck, A., 'Solving Least Squares Problems by Gram-Schmidt Orthogonalization,' BIT 7(1967), 1-21.
- [2] Chan, T. F. C., "On Computing the Singular Value Decomposition," Report STAN-CS-77-588, Stanford University (1977).
- [3] Gentleman, W. M., "Least Squares Computations by Givens Transformations Without Square Roots," JIMA 12 (1973), 329-336.
- [4] Golub, G., and Kahan, W., 'Calculating the Singular Values and Pseudo-Inverse of a Matrix,' J. SIAM Numer. Anal. 2(1965), 205-224.
- [5] Golub, G., and Luk, F., "Singular Value Decomposition: Applications and Computations," ARO Report 77-1, Transactions of the 22nd Conference of Army Mathematicians (1977).
- [6] Golub, G., and Reinsch, C., (1970, HACLA/I/10), "Singular Value Decomposition and Least Squares Solutions," Numer. Math. 14 (1970), 403-420.
- [7] Lanczos, C., Linear Differential Operators, Van Nostrand, London (1961).
- [8] Paige, C., 'The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices,' Ph.D. Thesis, University of London (1971).
- [9] Rutishauser, H., 'On Jacobi Rotation Patterns,' Proc. Sym. Applied Math. 15 (1963), 219-239.
- [10] Garbow, B. S., and Dongarra, J. J., Path Chart and Documentation for the EISPACK Package of Matrix Eigensystem Routines, Technical Memorandum No. 250, Argonne National Laboratory (August 1975).
- [11] Stewart, G. W., "The Economical Storage of Plane Rotations," Numer. Math. 25 (1976), 137-138.
- [12] Underwood, R., "An Iterative Block Lanczos Method for the Solution of Large Sparse Symmetric Eigenproblems," Ph.D. Thesis, Report STAN-CS-75-496, Stanford University (1975).
- [13] Van Loan, C., "Generalized Singular Values with Algorithms and Applications," Ph.D. Thesis, University of Michigan (1973).
- [14] Wilkinson, J. H., The Algebraic Eigenvalue Problem, Clarendon Press, Oxford (1965).

SUBROUTINE MAXVAL(M,N,Q,PINIT,G,MHAX,EPS,OP,H,D,X,Y,IORTHG,
 1 LOUT,MCHEPS,IECODE)
 INTEGER M,N,Q,PINIT,G,MHAX,H,IORTHG,LOUT,IECODE
 DOUBLE PRECISION EPS,D(Q),X(M,Q),Y(N,Q),MCHEPS
 EXTERNAL OP

CALCULATE THE LARGEST SINGULAR VALUES OF A LARGE SPARSE MATRIX

WRITTEN BY : FRANKLIN LUK
 COMPUTER SCIENCE DEPARTMENT
 STANFORD UNIVERSITY
 SBPTBRBER 1976
 LAST UPDATE : APRIL 1977

THIS SBT OPERATES USES INTEGER AND DOUBLE PRECISION ARITHMETICS

THIS SBT OPERATES INCLUDES : MAXVAL, BKLANC, ORTHOG, INPROD,
 ROTATB, CNVTST, PCEOIC, RANDOM,
 AND SVBUTH(PLUS BIBAND, ROTROW,
 ROTCOL, SVDBI, AND DROTAT).

THIS SUBROUTINE IS THE MAIN SUBROUTINE IMPLEMENTING
 THE ITERATIVE BLOCK LANCZOS METHOD FOR COMPUTING THE LARGEST
 SINGULAR VALUES AND CORRESPONDING LEFT AND RIGHT SINGULAR VECTORS
 OF AN M-BY-N MATRIX.

DESCRIPTION OF PARAMETERS :

M,N : INTEGER VARIABLES. THE NUMBER OF ROWS AND COLUMNS
 OF THE MATRIX A WHOSE SINGULAR VALUES AND VECTORS
 ARE BEING COMPUTED. IT IS ASSUMED THAT 2 ≤ M ≤ N.

Q : INTEGER VARIABLE. THE NUMBER OF VECTORS OF LENGTH M
 CONTAINED IN THE ARRAY X, AND THE NUMBER OF VECTORS
 OF LENGTH N CONTAINED IN THE ARRAY Y. THE VALUE OF Q
 SHOULD BE LESS THAN OR EQUAL TO 26, AT LEAST ONE GREATER
 THAN THE VALUE OF 6 AND LESS THAN OR EQUAL TO N.

PINIT : INTEGER VARIABLE. THE INITIAL BLOCK SIZE TO BE USED
 IN THE BLOCK LANCZOS METHOD. IF PINIT IS NEGATIVE,
 THEN -PINIT IS USED FOR THE BLOCK SIZE AND COLUMNS
 H+1, ..., H+(-PINIT) OF THE ARRAYS Y ARE ASSUMED
 TO BE INITIALIZED TO A MATRIX USED TO START THE BLOCK
 LANCZOS METHOD. IF THE SUBROUTINE TERMINATES WITH
 A VALUE OF H LESS THAN 6, THEN PINIT IS ASSIGNED
 A VALUE -P, WHERE P IS THE FINAL BLOCK SIZE CEOSBR.
 IN THIS CIRCUMSTANCE, COLUMNS H+1, ..., H+P OF Y
 WILL CONTAIN THE MOST RECENT SET OF RIGHT SINGULAR
 VECTOR APPROXIMATIONS WHICH CAN BE USED TO RESTART

C THE SUBROUTINE IF DESIRED.

C

C G : INTEGER VARIABLE. TEB UUHBBR OR SINGULAR VALUBS AND

C SINGULAR VECTORS BEING COHPUTBD. THAT IS, MAXVAL

C ATTEMPTS TO COHPUTB ACCURATE APPROXIMATIONS TO TEE

C 6 LARGEST SINGULAR VALUBS AND THEIR CORRBSPONDIUG

C LBPT AID RIGHT SINGULAR VBCTORS OF TEE MATRIX A. TEE

C THE VALUB OP G SEOULD BE POSITIVE AND LESS TEAR Q.

C

C MMAX : INTEGER VARIABLE. TEB MAXIMUM NUMBER OP MATRIX-VECTOR

C PRODUCTS $A \cdot X$ AND $TRANSPOSE(A) \cdot X$, WHERE X IS AN APPRO-

C PRIATE VBCTOR, TEAT ARE ALLOUDB DURING ONE CALL OF

C THIS SUBROUTINE TO COMPLETE ITS TASK OP COMPUTING

C G SINGULAR VALUBS AND VBCTORS. UNLESS THE PROBLEM

C INDICATES OTHERWISE, MMAX SEOULD BB GIVEN A VERY

C LARGE VALUB.

C

C EPS : DOUBLE PRBCISIOU VARIABLE. BPS SHOULD COUTAIU

C A VALUE INDICATING THE RELATIVE PRECISION TO WHICH

C MAXVAL WILL ATTEMPT TO COHPUTB TEB SINGULAR VALUBS

C AND VECTORS OP A. FOR SINGULAR VALUBS LESS IN MODULUS

C THAN 1, BPS UILL BE AUABSOLUTE TOLBRAUCE.

C

C OP : SUBROUTIPB NAME. TEE ACTUAL ARGUMENT CORRESPONDING

C TO OP SEOULD BB TEB NAME OP A SUBROUTINE USED TO

C DEFINE TEB MATRIX A. THIS SUBROUTINE SHOULD HAVE

C SIX ARGUMENTS M, N, P, U, V, AND ORIG, SAY, WHERE

C A IS AU M-BY-N ARRAY, U IS AU X-BY-P ARRAY,

C V IS AU U-BY-P ARRAY, AND ORIG IS A LOGICAL VARIABLE.

C THE STATEMENT

C CALL OP (M,N,P,U,V,.TRUE.)

C SEOULD RESULT IN THE ARRAY $A \cdot V$ BEING COHPUTBD AND

C STORED IN U. THE STATEMENT

C CALL OP (M,N,P,U,V,.FALSE.)

C SEOULD RESULT IN THB ARRAY $TRANSPOSE(A) \cdot U$ BEING

C COHPUTED AND STORBD IN V.

C

C H : INTEGER VARIABLE. H GIVES THB NUMBER OF SINGULAR

C VALUBS AND LEFT AND RIGHT SINGULAR VBCTORS ALRBADY

C COMPUTED. THUS, INITIALLY, H SHOULD BB ZERO.

C IF H IS GREATER THAN ZERO, THEN ELEMENTS OUB THROUGH

C H OF THB VICTOR D COUTAIU APPROXIMATIONS TO THE H

C LARGEST SINGULAR VALUBS OP A, COLUMNS ORB THROUGH H

C OF THB ARRAYS X AND Y CONTAIN APPROXIMATIONS TO TEE

C CORRESPONDING LEFT AND RIGHT SINGULAR VBCTORS,

C AT EXIT, H CONTAINS A VALUB EQUAL TO THE TOTAL NUMBER

C OF SINGULAR VALUBS AND LEFT AND RIGHT SINGULAR VBCTORS

C COMPUTED INCLUDING ANY ALREADY COHPUTBD WHEN MAXVAL

C WAS ENTERED. THUS, AT EXIT, THE FIRST HELEMENTS OF D

C AND THE FIRST H COLUMNS OF X AND Y WILL CONTAIN

C APPROXIMATIONS TO THE H LARGEST SINGULAR VALUBS OF A AND

C THEIR CORRESPONDING LBPT AND RIGHT SINGULAR VECTORS.

C

C D : DOUBLE PRECISION ARRAY. D CONTAINS THB COHPUTBD SINGULAR

C VALUBS. D SHOULD BB AU ONE-DIMENSIONAL ARRAY WITH AT

C LEAST 6 ELEMENTS.

C

C X : DOUBLE PRBCISIOR ARRAY. X CONTAINS THB COEPUTBD LEFT

C SINGULAR VECTORS. X SEOULD BE AN ARRAY CONTAINING AT

C LBAST M*Q ELEMENTS. X IS USED ROT ONLY TO STORB THE LEFT


```

C
C      C (Q*(Q2+3)) , U (Q*Q) , V (Q*Q) , R (Q2*Q2) , T (H)
C
DOUBLE PRECISION C (416) , U (676) , V (676) , R (169) , T (1000)
DOUBLE PRECISION DBLE
C
C      ISEED IS SBBD FOR RANDOM NUMBER GENERATOR
C
DATA ISEED/99991/
C
C      CHECK THAT THE INITIAL VALUES OF THE SUBROUTINE PARAMETERS ARE IN RANGE.
C
      I? (U.LT.2) 60 TO 901
      IF (H.LT.H) GO TO 902
      IF (H.GT.1000) 60 TO 903
      IF (G.LT.1) 60 TO 904
      IF (Q.LB.6) 60 TO 905
      IF (Q.GT.26) GO TO 906
      IF (Q.GT.H) GO TO 907
C
C      INITIALIZE THE SINGULAR VALUES TO VERY LARGE NEGATIVE NUMBERS.
C
      DO 110 I = 1,G
        D(I) = -1.010
110 CONTINUE
C
C      CEOOSB INITIAL VALUES FOR THE BLOCK SIZE P, THE NUMBER S
C      OF STBPS THAT THE BLOCK LANCZOS METHOD IS CARRIED OUT, AND
C      CEOOSB AN INITIAL II-BY-P ORTHONORMAL MATRIX 11 TO START
C      THE BLOCK LANCZOS METHOD.
C
      P = PINIT
      I? (P.LT.0) P = -P
      S = (Q-H)/P
      IF (S.GE.2) GO TO 120
      S = 2
      P = (Q-H)/2
C
120 I? (PINIT.LT.0) GO TO 200
C
      INSERT RANDOM VECTORS INTO COLUMNS H+1 THROUGH H+P OF THE ARRAY Y.
C
      CALL RANDOM(N,Q,P,H,Y,ISEED)
C
      SET CONSTANTS FOR LATER CONVERGENCE TESTS.
C
200 ERRBND = BPS + 10.DO*DBLE(FLOAT(H+H)) *HCHEPS
      ERRC = 0.DO
      ITBR = 0
      INH = 0
      IVV = 0
C
C      THE MAIN BODY OF THE SUBROUTINE STARTS HERE. INH
C      COUNTS THE NUMBER OF MATRIX-VECTOR PRODUCTS COMPUTED.
C      IVV COUNTS THE NUMBER OF VECTOR INNER PRODUCTS PERFORMED
C      IN THE ORTHOGONALIZATION ROUTINE. ERRC MEASURES THE
C      ACCUMULATED ERROR IN THE SINGULAR VALUES AND VECTORS.
C

```

```

300 IF (H.GE.G) GO TO 900
    IF (IMH.GT.HMAX) GO TO 908
    ITER = ITER+1
    PS = P+S
    PP3 = P+3
    WRITE (LOUT,6010) ITER,P,S
6010 FORMAT(14H • ** ITERATION,I4/5X,4H P =,I3,5X,4H S =,I3)
C
C     USE RANDOM VECTORS TO RESTART THE LANCZOS ALGORITHM IF
C     LINEAR INDEPENDENCE HAS BEEN LOST.
C
DO 310 I= 1,P
    IPH = I+H
    IF (D(IPH).GT.0.D0) GO TO 310
    PHI = P-I
    CALL RANDOM(N,Q,PHI+1,IPH-1,Y,ISEED)
    GO TO 320
310 CONTINUE
C
C     ORTHONORMALIZE COLUMNS H+1 THROUGH H+P OF TEB ARRAY Y.
C
320 CALL ORTHOG(N,Q,H,H,P,R,Y,IORTHG,IVV,LOUT,NCHEPS)
C
C     BKLANC CARRIES OUT THE BLOCK LANCZOS METHOD AND
C     RETURNS THE RESULTING BAUBD UPPER TRIANGULAR MATRIX MS
C     IN C, THE M-BY-PS ORTHONORMAL MATRIX XS IN X AND THE
C     N-BY-PS ORTHONORMAL MATRIX YS IN Y. THE INITIAL
C     N-BY-P ORTHONORMAL MATRIX YI IS ASSUMED TO BE STORED
C     IN COLUMNS H+1 THROUGH H+P OF Y.
C
CALL BKLANC(N,H,Q,PP3,H,P,S,OP,C,X,Y,R,IORTHG,IVV,LOUT,NCHEPS)
IMH = IMH + P*(2*S-1)
C
C     SVBUTH SOLVES THE SINGULAR VALUE PROBLEM FOR THE PS-BY-PS
C     ARRAY MS, RETURNING THE SINGULAR VALUES IN THE SBCOBD COLUMN
C     OF C AND THE RIGHT SINGULAR VECTORS IN THE FIRST P*S COLUMNS
C     OF U, AND THE P LEFT SINGULAR VECTORS CORRESPONDING TO THE
C     P LARGEST SINGULAR VALUES IN THE FIRST P COLUMNS OF V.
C
CALL SVBUTH(Q,PS,P,PP3,C,PS,PS,U,V,NCHEPS,IERR)
IF (IERR.EQ.0) GO TO 330
WRITE(LOUT,6020) IERR
6020 FORMAT(5X,39H *** ERROR IN SUBROUTINE SVBUTH. IERR =,I3,4H ***)
330 QP1 = Q+1
    QPPS = Q+PS
    WRITE(LOUT,6030) (C(I),I=QP1,QPPS)
6030 FORMAT(5X,20H SINGULAR VALUES . . .,6 (/5X,1P5D24.15))
C
C     ROTATE COMPUTES THE LEFT AND RIGHT SINGULAR VECTORS
C     OF THE RESTRICTED MATRIX USING XS STORED IN X, AND YS
C     STORED IN Y.
C
CALL ROTATE(N,Q,H,PS,PS,U,X,T)
CALL ROTATE(N,Q,H,PS,PS,V,Y,T)
C
C     TEST IF RELATIVE INCREASE OF COMPUTED SINGULAR VALUES EXCEEDS
C     THE USER-SET PRECISION BOUND.
C
NCONV = 0
I? (ITER.EQ.1) GO TO 340

```

```

      IF ( ( C ( Q+1 ) - D ( H+1 ) ) / C ( Q+1 ) .GT. BPS ) GO TO 400
C
C      CNVTST DETERMINES HOW MANY OF THE SINGULAR VALUES
C      AND LEFT AND RIGHT SINGULAR VECTORS HAVE CONVERGED.
C      THE NUMBER THAT HAVE CONVERGED IS STORED IN NCONV.
C      IF NCONV=0, THEN NONE HAS CONVERGED.
C
340 CALL CNVTST ( H, N, Q, H, G, ERRBND, ERRC, OP, C, X, Y, NCONV, LOUT, T )
      INH = INH + ( NCONV+1 ) * 2
400 CONTINUE
C
      DO 410 I = 1, PS
          IPH = I+H
          IPQ = I+Q
          D ( IPH ) = C ( IPQ )
410 CONTINUE
2
C      PCHOIC CHOOSES NEW VALUES FOR P AND S, THE BLOCK
C      SIZE AND THE NUMBER OF STEPS FOR THE BLOCK LANCZOS
C      SUBPROGRAM, RBSP..
C
      IF ( NCONV.EQ.0 .OR. NCONV.EQ.G-H ) GO TO 420
      CALL PCHOIC ( Q, H, G, NCONV, P, S )
420 WRITE ( LOUT, 6040 ) INH, IVV, NCONV
6040 FORMAT ( 5X, 6H INH =, I5, 5X, 6H IVV =, I5, 5X, 8H NCONV =, I3 )
      II = H+NCONV
C
      GO TO 300
C
C      THIS IS THE END OF THE MAIN BODY OF THE SUBROUTINE.
C      NOW SET THE VALUE OF THE IBCODB AND EXIT.
C
900 IECODE = 0
      RBTURR
901 IECODE = 1
      RETURN
902 IECODE = 2
      RETURN
903 IECODB = 3
      RETURN
904 IECODE = 4
      RETURN
905 IBCODB = 5
      RETURN
906 IBCODB = 6
      RETURN
907 IBCODB = 7
      RETURN
908 IBCODB = 8
      PINIT = -P
      RETURN
C
      END

```

```

SUBROUTINE BKLANC(M,N,Q,PP3,H,P,S,OP,C,X,Y,R,IORTHG,IVV,
1      LOUT,MCHEPS)
INTEGER M,N,Q,PP3,H,P,S,IORTHG,IVV,ICUT
DOUBLE PRECISION C(Q,PP3),X(M,Q),Y(N,Q),R(P,P),MCHEPS

C
C      THIS SUBROUTINE IMPLEMENTS THE BLOCK LANCZOS
C      METHOD WITH REORTHOGONALIZATION. BKLANC COMPUTES
C      A PS-BY-PS ( PS=P*S ) BANDED UPPER TRIANGULAR
C      MATRIX MS WHICH IT STORES IN COLUMNS 2 THROUGH P+2
C      OF THE Q-BY-P+1 MATRIX C ( THE DIAGONAL BEING STORED
C      IN THE FIRST PS LOCATIONS OF COLUMN 2, THE NEXT
C      SUPERDIAGONAL BEING STORED IN THE FIRST PS-1 LOCATIONS
C      OF COLUMN 3, AND SO ON ), AND A PS-BY-PS ORTHOGONAL
C      MATRIX XS WHICH IT STORES IN COLUMNS H+1 THROUGH H+PS
C      OF THE E-BY-Q ARRAY X, AND A PS-BY-PS ORTHOGONAL
C      MATRIX YS WHICH IT STORES IN COLUMNS H+1 THROUGH H+PS
C      OF THE N-BY-Q ARRAY Y.
C      MS CAN ALSO BE REGARDED AS A BLOCK UPPER TRIANGULAR
C      MATRIX WITH P-BY-P UPPER TRIANGULAR MATRICES R(1), . . . .
C      R(S) ON ITS DIAGONAL AND P-BY-P LOWER TRIANGULAR
C      MATRICES T(2)', . . . . T(S)' ALONG ITS UPPER DIAGONAL.
C      XS IS COMPOSED OF S PS-BY-P ORTHONORMAL MATRICES
C      X(1), ..., X(S).
C      YS IS COMPOSED OF S PS-BY-P ORTHONORMAL MATRICES
C      Y(1), ..., Y(S), WHERE Y(1) IS GIVEN AND SHOULD BE
C      STORED IN COLUMNS H+1 THROUGH H+P OF Y.
C      OP IS THE NAME OF AN EXTERNAL SUBROUTINE USED TO
C      DEFINE THE MATRIX A.
C
INTEGER I,I1,I2,J,JMP,J1,J2,K,K1,L,LL,LLMP,IU
DOUBLE PRECISION T

C
C      L = 1
C
C      LL = H+1
C      LU = H+P
C
C      COMPUTE X(1) = A*Y(1)
C
CALL OP(M,N,P,X(1,LL),Y(1,LL),.TRUE.)
C
C      FACTORIZE X(1) := X(1)*R(1)
C
CALL ORTHOG(M,Q,H,H,P,R,X,IORTHG,IVV,LOUT,MCHEPS)
C
C      STORE R(1) IN C
C
DO 120 J = 1,P
C
DO 110 I = 1,J
J1 = J-I+2
C(I,J1) = R(I,J)
110 CONTINUE
C
120 CONTINUE
C
C      L = L + 2
C
IF (S.LT.2) GO TO 900

```

```

DO 600 L = 2,S
  LL = H+ (L-1) *P+1
  LU = H+L*P
  I1 = (L-2) *P
  I2 = I1+P

C
C
C
  COHPUTB A'*X(L-1)

  LLMP = LL-P
  CALL OP (H,N,P,X(1,LLMP),Y(1,LL),.FALSE.)

C
DO 230 K = LL,LU

C
C
C
  COMPUTE Y(L-1)*R(L-1)'

  K1 = K-LL+1

DO 220 I = 1,N
  T = 0.DO

C
DO 210 J = K,LU
  JMP = J-P
  J1 = J-LL+1
  T = T + Y(I,JMP)*R(K1,J1)
210  CONTINUE

C
C
C
  COHPUTB Y(L) = A'*X(L-1) - Y(L-1) * B(L-1)'

  Y(I,K) = Y(I,K)-T
220  CONTINUE

C
230  CONTINUE

C
C
C
  FACTORIZE Y(L) := Y(L)*T(L)

  CALL ORTHOG (N,Q,H,LL-1,P,R,Y,IORTHG,IVV,IOUT,HCHEFS)

C
C
C
  STORB T(L)' IN C

DC 320 J = 1,P
  J1 = J+I1

C
DO 310 I = 1,J
  J2 = P-J+I+2
  C(J1,J2) = R(I,J)
310  CONTINUE

C
320  CONTINUE

C
C
C
  CCOMPUTE A*Y(L)

  CALL OP (H,N,P,X(1,LL),Y(1,LL),.TRUE.)

C
DO 430 A = LL,LU

C
C
C
  COMPUTE X(L-1)*T(L)'

  K1 = K-LL+1

C
DO 420 I = 1,M

```

```

      T = 0.DO
C
      DO 410 J = K,LU
        JMP = J-P
        31 = J-11+1
        T = T + X(I,JMP)*R(K1,J1)
410    CONTINUE
C
      COMPUTE X(L) = A*Y(L) - X(I-1)*T(L) '
C
      X(I,K) = X(I,K)-T
420    CONTINUE
C
430    CCNTINUE
C
      FACTORIZE X(L) := X(L)*R(L)
C
      CALL ORTHOG (M,Q,H,LL-1,P,R,X,IORTHG,IVV,IOUT,MCHKEFS)
C
      STORE R(L) IN C
C
      DO 520 J = 1,P
C
        DO 510 I = 1,J
          11 = I+12
          J1 = J-142
          C(I1,J1) = R(I,J)
510    CONTINUE
C
520    CCNTINUE
C
600    CCNTINUE
C
900    CCNTINUE
      RETURN
      END

```

```

SUEROUTIUE ORTHOG(N,Q,H,L,P,R,X,IORTHG,IVV,LOUT,MCHEFS)
INTEGER N,Q,H,L,P,IORTHG,IVV,LOUT
DCUBLB PRECISION B (P,P),X(N,Q),MCHEFS

```

```

C
C      OBTEOG REORTHOGONALIZES THE N-BY-P MATRIX Z STORED IN
C      COLUMNS L+1 THROUGH L+P OF THE N-BY-Q ARRAY X WITH
C      RESPECT TO THE VECTORS STORED IN COLUMNS 1 THROUGH H
C      AND COLUMNS (L-IORTHG+P+1) THROUGH L OF THE MATRIX X
C      USING GRAM-SCHMIDT ORTHOGONALIZATION. THE MODIFIED
C      GRAM-SCHMIDT METHOD IS USED TO FACTORIZE THE RESULTING
C      MATRIX INTO THE PRODUCT OF AN N-BY-F ORTHONORMAL MATRIX
C      STORED IN COLUMNS L+1 THROUGH L+P OF X, AND
C      AP-BY-P UPPER TRIANGULAR ARRAY R.
C

```

```

INTEGER I,IH1,IP1,J,K,KH1,L1,LP1,LPP
INTEGER MAX0
DOUBLE PRECISION SUM
DOUBLE PRECISION DSQRT
IF (P.EQ.0) RETURN
LP1 = L+1
LPP = L+P

```

```

C
C      IF (H.EQ.0) GO TO 200
C

```

```

DC 130 I = LP1,LPP

```

```

C      DO 120 K = 1,H
C          CALL INPROD(N,X(1,I),X(1,K),SUM)
C

```

```

C          DO 110 J = 1,N
C              X(J,I) = X(J,I) - SUM*X(J,K)
110          CONTINUE
C

```

```

C      120 CONTINUE
C

```

```

C      130 CCNTINUE
C

```

```

C      IVV = IVV + H*P
C

```

```

C      200 IF (IORTHG.EQ.0) GO TO 300
C          IF (L.EQ.H) GO TO 300
C          L1 = MAX0(L-P*IORTHG+1, H+1)
C

```

```

DC 230 I = LP1,LPP

```

```

C      DO 220 K = L1,L
C          CALL INPROD(N,X(1,I),X(1,K),SUM)
C

```

```

C          DO 210 J = 1,N
C              X(J,I) = X(J,I) - SUM*X(J,K)
210          CONTINUE
C

```

```

C      220 CCNTINUE
C

```

```

C      230 CCNTINUE
C

```

```

C      IVV = IVV + (L-L1+1)*P
C

```

```

C      300 CCNTINUE

```



```

C      DC 400 I = IP1,LPP
      SUM = 0.DO
C
      DO 310 J = 1,N
        sun = SUM + X(J,I)**2
310    CCNTINUE
C
      IML = I-1
      IF (SUM.GT.MCHEPS) GO TO 330
C
      WRITE(LOUT,6010)
6010    FORMAT(5X,47H *** WARNING • LINEAR INDEPENDENCE MAY BE LOST,
1        24H. VECTOR SET TO ZERO *** )
      R(IML,IML) = 0.DO
      DO 320 J = 1,N
        X(J,I) = 0.DO
320    CCNTINUE
      GO TO 400
C
330    SUM = DSQRT(SUM)
      R(IML,IML) = SUM
      sun = 1.DO/SUM
      DO 340 J = 1,N
        X(J,I) = SUM*X(J,I)
340    CCNTINUE
C
350    IF1 = I+1
      IF (IP1.GT.LPP) GO TO 400
C
      DO 370 K = IP1,LPP
        CALL INPROD(N,X(1,I),X(1,K),SUM)
        KML = K-L
        R(IML,KML) = SUM
C
        DO 360 J = 1,N
          X(J,K) = X(J,K) - SUM*X(J,I)
360    CONTINUE
C
330    CONTINUE
C
400    CCNTINUE
C
      IVV = IVV + (P-1)*P/2
      RETURN
      END

```

```
SUBROUTINE INPROD(N,U,V,SUM)
  INTEGER N
  DOUBLE PRECISION U(N),V(N),SUM
```

```
C
C      INPROD COMPUTES THE INNER PRODUCT OF 2 VECTORS U AND V,
C      EACH OF LENGTH N, AND STORES THE RESULT IN S.
C
```

```
  INTEGER I
  SUM = 0.D0
```

```
C
C      DO 110 I = 1,N
C          SUM = SUM + U(I)*V(I)
110 CONTINUE
```

```
C
C      RETURN
C      END
```

```

SUBROUTINE ROTATE(N,Q,H,PS,L,U,X,T)
INTEGER N,Q,H,PS,L
DOUBLE PRECISION U(Q,L),X(N,Q),T(Q)

```

```

C
C      ROTATE COMPUTES THE FIRST L COLUMNS OF THE MATRIX
C      XS*QS, WHERE XS IS AN N-BY-PS ORTHONORMAL MATRIX STORED
C      IN COLUMNS H+1 THROUGH H+PS OF THE N-BY-Q ARRAY X AND
C      QS IS A PS-BY-PS ORTHONORMAL MATRIX WHOSE FIRST L COLUMNS
C      ARE STORED IN COLUMNS 1 THROUGH L OF THE ARRAY U. THE
C      RESULT IS STORED IN COLUMNS H+1 THROUGH H+L OF X
C      OVERWRITING PART OF XS.
C

```

```

INTEGER I,J,JPH,K,KPH
DOUBLE PRECISION SUM

```

```

C
DC 200 I = 1,N

```

```

C
C      COMPUTE THE I-TH ROW OF XS*QS
C

```

```

DO 110 K = 1,L
    SUM = 0.D0

```

```

C
DO 105 J = 1,PS
    JPH = J+H
    sun = SUM + X(I,JPH)*U(J,K)
105    CONTINUE

```

```

C
T(K) = sun

```

```

110    CCNTINUE

```

```

C
DO 120 K = 1,L
    KPH = K+H
    X(I,KPH) = T(K)
120    CONTINUE

```

```

C
200    CCNTINUB

```

```

C
EBTUFN
END

```

```

SUBROUTINE CNVTST(M,N,Q,H,G,ERRBND,ERRC,OF,C,X,Y,NCCNV,
1      LOUT,T)
  INTEGER M,N,Q,H,G,NCCNV,LOUT
  DOUBLE PRECISION ERBND,ERRC
  DOUBLE PRECISION C(Q,2),X(M,Q),Y(M,Q),T(M)

```

```

C      CRVTST DETERMINES WHICH OF THE P COMPUTED SINGULAR
C      VALUES STORED IN THE SECOND COLUMN OF C HAVE CCNVERGED.

```

```

C      THE RESIDUAL RESIDU OF THE (H+I)-TH SINGULAR VALUE
C      IS COMPUTED BY

```

```

C      RBSIDU = DSQRT( 2NORM( A*Y(H+I) - X(H+I)*C(I,2) ) **2
C
C              + 2NORM( A*X(H+I) - Y(H+I)*C(I,2) ) **2 ).

```

```

C      BRRC IS A MEASURE OF THE ACCUMULATED ERROR IN THE
C      H PREVIOUSLY COMPUTED SINGULAR VALUES AND LEFT AND RIGHT
C      SINGULAR VECTORS.

```

```

C      WE DECIDE THE (H+I)-TH SINGULAR VALUE HAS CCNVERGBD
C      IF

```

```

C      RBSIDU .LE. B*ERRBND + ERRC,

```

```

C      WHERE B EQUALS C(I,2) IF THE LATTER IS GREATER THAN 1,
C      AND 1 OTHERWISE. HENCE WE DO A RELATIVE ERROR TEST IF THE
C      CCPUTBD SINGULAR VALUE IS GREATER THAN 1, AND AN ABSOLUTE
C      ERROR TEST CTHBRUISB.

```

```

C      THE CONVERGENCE TEST IS PERFORMED IN ORDER ON THE (H+1)-TH,
C      (H+2)-TH, . . . COMPUTED SINGULAR VALUES. AS SOON AS A COMPUTED
C      VALUE FAILS THE TEST, RETURN IS MADE TO THE CALLING ROUTINE.

```

```

C      NCONV IS THE NUMBER THAT HAS CCNVERGED. IF NCONV=0,
C      THEN NONE HAS CONVBEGBD.

```

```

C      INTEGER I,IPH,K,L,PT
C      DOUBLE PRECISION RESIDU,B,SUM
C      DOUBLE PRECISION DSQRT

```

```

C      SUB = 0.0
C      FT = G-H

```

```

C      DO 200 I = 1,PT
C      K = I
C      IF (C(I,2).EQ.0.0) GO TO 300
C      IPH = I+H
C      CALL OP(M,N,1,T,Y(1,IPH),.TRUE.)

```

```

C      RBSIDU = 0.0
C      DO 110 L = 1,M
C      B = T(L) - C(I,2)*X(L,IPH)
C      RBSIDU = RBSIDU + B**2

```

```

110    CONTINUE

```

```

C      CALL OP(M,N,1,X(1,IPH),T,.FALSE.)

```

```

C      DO 120 L = 1,M
C      B = T(L) - C(I,2)*Y(L,IPH)
C      RESIDU = RESIDU + B**2

```

```

120    CONTINUE

```

```

C      TEST FOR CONVERGENCE

```

```

RESIDU = DSQRT(RBSIDU)
B = C(I,2)
IF (B.LT.1.D0) B = 1.D0
IF (RESIDU.LE.B*ERRBND+ERRC) GO TC 130
C
WRITE(LOUT,6010) K,RESIDU
6010 FORMAT(5X,4H K =,I4,5X,9H RBSIDU =,1PD15.5,
1      36H      • ** COMPUTED VALUE REJECTED ***)
GO TO 300
C
130 WRITE(LOUT,6020) K,RESIDU
6020 FORMAT(5X,4H K =,I4,5X,9H RBSIDU =,1PD15.5,
2      36H      *** COMPUTED VALUE ACCEFTED ***)
SUM = SUM + RESIDU**2
IF (I.EQ.PT) K =K+1
200 CONTINUE
C
300 NCONV = K-1
IF (K.EQ.1) RETURN
C
BRRC = DSQRT(ERRC**2+SUM)
RETURN
END

```

```

SUBROUTINE PCHOIC(Q,H,G,NCONV,P,S)
INTEGER Q,H,G,NCCNV,P,S

```

```

      BASED ON THE VALUES OF Q, H, G AND NCCNV,
      PCHOIC CHOOSES NEW VALUES FOR P AND S, THE ELCK SIZE
      AND NUMBER OF STEPS FOR THE ELCK LANCZOS METHOD.

```

```

      THE STRATEGY IS : IF THE PREVIOUS ELCK SIZE IS
      GREATER THAN THE NUMBER OF SINGULAR VALUES TO BE
      COMPUTED, THEN THE NEW BLOCK SIZE EQUALS 'IRE PREVIOUS
      ELCK SIZE MINUS 'IRE NUMBER OF SINGULAR VALUES THAT
      HAVE CONVERGED IN THE CURRENT ITERATION, OTHERWISE
      THE NEW BLOCK SIZE IS CHOSEN TO BE THE SMALLER OF THE
      TWO VALUES : 1) THE PREVIOUS ELCK SIZE, AND 2) THE
      NUMBER OF SINGULAR VALUES TO BE COMPUTED. S IS CHOSEN
      AS LARGE AS POSSIBLE SUBJECT TO STORAGE CONSTRAINT,
      BUT ITS VALUE IS ALWAYS AT LEAST 2.

```

```

      H IS THE NUMBER OF SINGULAR VALUES AND LEFT AND RIGHT
      SINGULAR VECTORS THAT HAVE ALREADY BEEN COMPUTED AND G
      IS THE REQUIRED NUMBER. NCCNV IS THE NUMBER OF SINGULAR
      VALUES AND LEFT AND RIGHT SINGULAR VECTORS THAT HAVE
      CONVERGED IN THE CURRENT ITERATION.

```

```

      INTEGER HT,PT

```

```

      HT = H+ NCCNV
      IF (P.LE.G-H) GO TO 110

```

```

      F = F - NCONV
      S = (Q-HT)/F
      RETURN

```

```

110 PT = G - HT
      IF (P.GT.PT) P = PT
      S = (Q-HT)/P
      IF (S.GE.2) RETURN
      E = (Q-HT)/2
      S = (Q-HT)/P

```

```

      RETURN
      END

```

```
SUEROUTINE RANDCM(N,Q,P,H,X,ISEED)
INTEGER N,Q,P,H,ISEED
DCUBLE PRECISION X(N,Q)
```

C
C
C
C
C

```
    RANDCM COMPUTES AND STORES A SEQUENCE OF P*N PSEUDO-
    RANDCM INTEGERS ( VALUE BETWEEN 0 AND 2147483647 ) IN
    COLUMNS H+1 THBOUGH H+P OF THE N-EY-Q ARRAY X.
```

```
INTEGER I,L,LPH
```

```
DC 130 L = 1,P
```

```
    LPH = L+H
```

```
DO 120 I = 1,N
```

```
    ISBBD = ISEED*314159269 + 453806245
```

C
C
C

```
    THE STATEHBNT NUHBBR 110 IS TG PREVENT UNWANTED,
    OPTIMIZATICN BY TEE COMPILER.
```

```
110    IF (ISEED.LT.0) ISBBD = ISEED + 2147483647 +1
        X(I,LPH) = ISBBD
```

```
120    CONTINUE
```

C

```
130 CCNTINUB
```

```
    RETURN
```

```
END
```

SUBROUTINE SVBUTH (NDIM, N, H, MP3, C, NO, NV, U, V, NCHEPS, IERR)

```

INTEGER NDIM, N, M, MP3, NU, NV, IERR
DOUBLE PRECISION C(NDIM,MP3), U(NDIM,NU), V(NDIM,NV), MCHEPS

```

CALCULATE THE SINGULAR VALUE DECOMPOSITION OF A BANDED UPPER TRIANGULAR MATRIX

YRITBN BY: **M. L. OVERTON**
COMPUTER SCIENCE DEPARTMENT
STAMFORD UNIVERSITY
JANUARY 1976

LAST UPDTB: **JANUARY 1976**

THIS ROUTINE COMPUTES TEE SINGULAR VALUE DECOMPOSITION OF A REAL
N*N MATRIK A, I. E. IT COMPUTES MATRICES U, S AND V SUCH TEAT

$$A = U * S * V^T,$$

WHERE

U IS AN N*N MATRIX AND $U^T * U = I$, ($U^T =$ TRAUSPOSB
OF U) ,

V IS AN $N \times N$ MATRIX AND $VT * V = I$, ($VT = \text{TRANPOSE OF } V$),

AND S IS AN $N \times N$ DIAGONAL MATRIX.

THE CALCULATION IS PERFORMED IN TWO STEPS:

1. **REDUCE THE BANDED UPPER TRIANGULAR MATRIX TO AN UPPER BIDIAGONAL MATRIX USING GIVENS TRANSFORMATIONS. THIS IS DONE BY SUBROUTINE BIBAND.**

THE METHOD USBD IS SIMILAR TO THE METHOD USED FOR TRIDIAGONALIZING A SYMMETRIC BANDED MATRIX, DESCRIBED IN H. RUTISHAUSER, ON JACOBI ROTATION PATTERNS, PROC. OF SYMP. IN APPLIED MATH., VOL. XV, EXPERIMENTAL ARITH., HIGH SPEED COMPUTING, AND MATH. (1963). FOR FURTHER DETAILS SEE COMMENTS AT BEGINNING OF THE SUBROUTINE.

2. DIAGONALIZE THE BIDIAGONAL MATRIX TO OBTAIN THE SINGULAR VALUES. THIS IS DONE BY SUBROUTINE SVDBI.

TEE METHOD USED IS A VARIANT OF THE QR ALGORITHM,
DESCRIBED IN: GOLUB AND REINSCH, SINGULAR VALUE DECOMPOSITION
AND LBAST SQUARES SOLUTION, NUMER. BATH. 14, 403-420(1970),
SECTION 1.3.

TEE ROUTINE IS IN DOUBLE PRECISION

THE SPEED OF **THIS ROUTINE** COULD BE IMPROVED BY **IMPLEMENTING**

THE SPEED OF **THIS ROUTINE** COULD BE IMPROVED BY **IMPLEMENTING**

PAST GIVENS TRANSFORMATION

ADDITIONAL SUBROUTINES REQUIRED: BIBAUD, WITH ROTROW AND ROTCOL
SVDBI, WITH DROTAT

THE FORMAL PARAMETERS ARE:

NDIM - THE QUANTITY USED TO DECLARE THE FIRST DIMENSION OF THE
ARRAYS C,U,V (NDIM .GE. N)

M - THE ORDER OF THE BAUDBD UPPER TRIANGULAR MATRIX A

M - THE UUMBBR OF SUPERDIAGONALS IN THE MATRIX A:
A(I,J) = 0 FOR J .GT. I+M AND J .LT. I

MP3 - THE NUMBER OF COLUMNS IN THE ARRAY C. MUST BE SET TO M+3.

C - AN NDIM * MP3 ARRAY WHICH HOLDS THE NONZERO ELEMENTS OF
OF A.
THE DIAGONAL IS STORED IN THE FIRST M ELEMENTS OF
COLUMN 2, THE NEXT SUPERDIAGONAL IN THE FIRST U-1
ELEMENTS OF COLUMN 3, AND SO ON UP TO THE LAST
NONZERO SUPERDIAGONAL BEING STORED IN THE FIRST U-M
ELEMENTS OF COLUMN M+2. COLUMNS 1 AND M+3 ARE ARBITRARY.
THUS:

A(I,J) = C(I, J-I+2), I .LE. J .LB. I+M.
THE ROUTINE RETURNS THE DIAGONAL OF THE MATRIX S,
I. E. THE SINGULAR VALUES OF A, IN DESCENDING
ORDER, IN COLUMN 2 OF C - THUS THE
SINGULAR VALUES WILL BE:
C(1,2) .GE. C(2,2) .GE.GE. C(M,2)

NU, NV - INTEGER VARIABLES. THE UUMBBR OF COLUMNS IN THE
ARRAYS U AND V. SET NU TO M IF THE MATRIX U IS DESIRED,
OR SET NU TO 1 IF U IS NOT DESIRED. SET NV TO M
IF THE MATRIX V IS DESIRED, OR SET NV TO 1 IF V
IS NOT DESIRED,

U - REAL NDIM * NU ARRAY. IF NU = M, THE MATRIX U IS COMPUTED
AND STORED IN THE ARRAY U.

V - REAL NDIM * NV ARRAY. IF NV = M, THE MATRIX V IS COMPUTED
AND STORED IN THE ARRAY V.

IERR - ERROR FLAG. THE ERROR CODES RETURNED HAVE THE FOLLOWING
MEANINGS:

IBRR = 0: NORMAL RETURN
IERR = 2: ERROR - MP3 DOES NOT EQUAL M+3.
IERR = 3: ERROR - MU IS NOT SET TO M OR 1.
IERR = 4: ERROR - NV IS NOT SET TO M OR 1.
IERR = 5: ERROR - M IS GREATER THAN NDIM.

LOGICAL WITHU, WITHV

```

      INTEGER I,NH1,NHI
C
C CHECK INPUT PARAMETERS
      IERR = 0
      IF (MP3.NE.M+3) 60 TO 102
      IF (NU.NE.1.AND. NU.NE.N) GO TO 103
      IF (UU.BQ.1) WITHU = .FALSE.
      IF (NU.EQ.N) WITHU = .TRUE.
      IF (NV.NE.1 .AND. NV.NE.N) 60 TO 104
      IF (NV.EQ.1) WITHV = .FALSE.
      IF (NV.EQ.N) UITEV = .TRUE.
      IF (N.GT.NDIN) 60 TO 105
C
C TURN OFF UNDERFLOW
      CALL ERRSET(208,256,-1,1,1,0)
C
C BIDIAGONALIZB
      CALL BIBAND (NDIN,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V)
C
C THE SUPERDIAGONAL COLUMN MUST BE SHIFTED DOWN ONE ELEMENT IN C
C BBPORE CALLING SUBROUTINE SVDBI
      NH1=N-1
      DO 20 I=1,NH1
          NHI = N-I
20      C(NHI+1,3) = C(NHI,3)
      C(1,3) = 0.D0
C
C DIAGONALIZB
      CALL SVDBI(NDIN,N,C(1,2),C(1,3),NU,NV,WITHU,WITHV,U,V,MCHEPS)
      RETURU
C
C SET ERROR FLAGS
102 IBRR = 2
      RETURU
103 IBRR = 3
      RETURU
104 IERR = 4
      RETURN
105 IERR = 5
      RETURN
      BUD

```

```

C      SUBROUTINE BIEAND (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V)
C
C***** START OF BIEAND *****
C
C      INTEGER NDIM, N, M, MP3, NU, NV
C      LOGICAL WITHU, WITHV
C      DOUBLE PRECISION C (NDIM,MP3), U (NDIM,NU), V (NDIM,NV)
C
C      -----
C
C      REDUCE A BANDED UPPER TRIANGULAR MATRIX TO A BIDIAGONAL MATRIX
C      BY GIVENS TRANSFORMATIONS, PRESERVING THE SINGULAR VALUES.
C
C      WRITTEN BY:      M. L. OVEFTON
C                      COMPUTER SCIENCE DEPARTMENT
C                      STANFORD UNIVERSITY
C                      JANUARY 1976
C      LAST UPDATE:    JANUARY 1976
C
C      -----
C
C      THE ROUTINE IS IN DOUBLE PRECISION
C
C      -----
C
C      ADDITIONAL SUBROUTINES REQUIRED:  ROTROW AND ROTCOL
C
C      -----
C
C      THIS SUBROUTINE COMPUTES THE MATRICES U,J AND V SUCH THAT
C
C      WHERE
C
C      
$$A = U * J * V^T,$$

C
C      U IS AN N*N MATRIX AND  $U^T * U = I$ , ( $U^T$  = TRANSPOSE
C      OF U),
C      V IS AN N*N MATRIX AND  $V^T * V = I$ , ( $V^T$  = TRANSPOSE
C      OF V),
C      ADD J IS AN N*N UPPER BIDIAGONAL MATRIX.
C
C      THE METHOD USED IS SIMILAR TO THE METHOD USED FOR
C      TRIDIAGONALIZING A SYMMETRIC BANDED MATRIX, DESCRIBED IN
C      H. RUTISHAUSER, ON JACOBI ROTATION PATTERNS, PROC. OF STEP.
C      IN APPLIED MATH., VOL.XV, EXPERIMENTAL MATH., HIGH SPEED
C      COMPUTING, AND MATH. (1963).
C
C      -----
C
C      THE FORMAL PARAMETERS ARE:
C
C      NDIM - THE QUANTITY USED TO DECLARE THE FIRST DIMENSION OF THE
C      ARRAY C (NDIM .GE. N)
C
C      N - THE ORDER OF THE BANDED UPPER TRIANGULAR MATRIX A
C
C      M - THE NUMBER OF SUPERDIAGONALS IN THE MATRIX A:
C       $A(I,J) = 0$  FOR  $J .GT. I+M$  AND  $J .LT. I$ 
C
C      MP3 - THE NUMBER OF COLUMNS IN THE ARRAY C. MUST BE SET TO M+3.

```

C - AN **NDIM * HP3** ARRAY WHICH HOLDS THE NONZERO ELEMENTS OF
 OF A.
 TAB DIAGONAL IS STORED IN THE FIRST **N** ELEMENTS OF
 COLUMN 2, THE NEXT **SUPERDIAGONAL** IN THE FIRST **N-1**
 ELEMENTS OF COLUMN 3, AND SO ON UP TO THE LAST
NONZERO SUPERDIAGONAL BEING STORED IN THE FIRST **N-H**
 ELEMENTS OF COLUMN **M+2**. COLUMNS 1 AND **M+3** ARE ARBITRARY.
 THUS:

$$A(I,J) = C(I, J-I+2), \quad I \leq J \leq I+M.$$

THE ROUTINE RETURNS THE **BIDIAGONAL MATRIX J** WITH THE
 DIAGONAL IN THE FIRST **N** ELEMENTS OF COLUMN 2 OF C AND
 THE SUPERDIAGONAL IN THE FIRST **N-1** ELEMENTS OF
 COLUMN 3 OF C.

NU, NV - INTEGER VARIABLES. THE NUMBER OF COLUMNS IN THE
 ARRAYS **U** AND **V**. SET **NU** TO **N** IF **UITHU = .TRUE.**, OR SET
NU TO 1 OTHERWISE. SIMILARLY SET **NV** TO **N** OR 1.

WITHU, WITHV - LOGICAL VARIABLES. IF **UITHU = .TRUE.**, THEN
 THE MATRIX **U** IS COMPUTED AND STORED IN THE ARRAY **U**.
 IF **WITHV = .TRUE.**, THEN THE MATRIX **V** IS COMPUTED AND
 STORED IN THE ARRAY **V**.

U - REAL **NDIM * NU** ARRAY.

V - REAL **NDIM * NV** ARRAY.

 INTEGER **NM2, I, J, K, J0, JOFF, KK**

INITIALIZE U, V

IF (.NOT. **WITHU**) GO TO 81

DC 80 **I=1, N**

DO 70 **J=1, N**

70 **U(I, J) = 0. D0**

U(I, I) = 1. D0

80 CONTINUE

81 CONTINUE

IF (.NOT. **WITHV**) GO TO 101

DO 100 **I=1, N**

DO 90 **J=1, N**

90 **V(I, J) = 0. D0**

V(I, I) = 1. D0

100 CONTINUE

101 CONTINUE

HANDLE **DBGBNBBATB** CASE

IF (**M.LT.2.OR.N.LT.3**) RETURN

NM2=N-2

ZERO WORKING SPACE ON LEFT AND RIGHT SIDES OF C

DC 120 **I=1, N**

C(I, 1) = 0. D0

C(I, NM2) = 0. D0

120 CONTINUE

PASS DOWN THE ROWS OF A

```

DC 400 I=1,NM2
C      LOOK AT TXE ELEHENTS OUTSIDE THE BIDIAGONAL PART
C      FOR K PROH M STEP -1 UNTIL 2...
      DO 300 KK=2,M
        K=M+2-KK
C      THE POLLOUING LOOP FIRST ANNIHILATES THE CHOSEN ELEMENT
C      BY A COLUMN ROTATION UITH JOFF=K. THIS CREATES A NEW
C      BLBHNT TO BE ZEROED BY A ROW ROTATION WHICH CREATES A
C      NBU ORE TO BE ZEROED EY A COLUMN ROTATION WITH JOFF=M+1
C      AND SO ON UNTIL THE ELEMENT IS CHASED OFF THE MATRIX.
        JO=I+K
        JOFF=K
        IF (JO.GT.N) GO TO 201
        DO 200 J=JO,N,M
          ROTATE COLUMNS TO ANNIHILATE ELEMENT
          CALL ROTCOL (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V
            ,J,JOFF)
            JOFF=M+1
C          ELEMENT CREATED BELCW DIAGONAL - ZERO IT AND
C          CREATE ANOTNER ABOVE EY ROTATING RCWS
          CALL ROTROW (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V
            ,J)
            200      CONTINUE
        201      CONTINUE
        300      CONTINUE
        400      CONTINUE
      FETUFN
      END

```

```

C      SUERCUTINE ROTBOW (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V,I)
C      APPLY TO HATRIX A ON THE LBPT SIDE AGIVENS TRANSFORMATION
C      TO ROTATE ROWS I AND I-1 SUCH TEAT TEE SUBDIAGONAL ELEMENT A (I,I-1)
C      IS ANNIHILATED
C
C      RECALL THAT A IS STORED IN C UITH
C      A (I,J)=C (I,J-I+2)   I .LE. J .LE. I+M
C
C      INTEGER NDIM,N,M,MP3,NU,NV,I
C      DCUBLE PRECISION C (NDIM,MP3),U (NDIM,NU),V (NDIM,NV)
C      LCGICAL WITHU,WITHV
C      INTEGER K,MP1
C      DCUBLE PRECISION X,Y,Z,COST,SINT,TEMP,S,DABS,DSQRT
C
C      X=C (I-1,2)
C      Y=C (I,1)
C      IF Y IS ZERO THEN THERE IS NOTHING TC DO
C      IF (Y.EQ.0.D0) RETURN
C      PERFORM Z=SQRT (X*X+Y*Y) ; COSTT=X/Z; SINT=Y/Z WITH SCALING TO
C      PREVENT UNDERFLOW
C      S=DABS (X) +DABS (Y)
C      CCST=X/S
C      SINT=Y/S
C      Z=DSQRT (COST*COST+SINT*SINT)
C      CCST=COST/Z
C      SINT=SINT/Z
C      C (I-1,2)=Z*S
C      C (I,1)=0.D0
C      MP1=M+1
C      DO 100 K=1,MP1
C          IF (I-1+K.GT.N) GO TO 100
C          TEMP=C (I-1,K+2)
C          C (I-1,K+2)=COST*TEMP + SINT *C (I,K+1)
C          C (I,K+1)=-SINT*TEMP + COST*C (I,K+1)
100      CCNTINUE
C
C      UPDATE U (ACCUMULATE TRANSFORMATIONS) - BUST UPDATE U ON THE
C      RIGHT BECAUSE U IS WANTED, NOT U TRANSPOSED
C      IF (.NOT. WITHU) RETURN
C      DC 200 K=1,N
C          TEMP=U (K,I-1)
C          U (K,I-1)=COST*TEMP + SINT*U (K,I)
C          U (K,I)=-SINT*TEMP + COST*U (K,I)
200      CONTINUE
C      RETURN
C      END
C

```

```

C      SUERCUTINE ROTCOL (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V,J,JOFF)
C      APPLY TO MATRIX A ON THE RIGHT SIDE A GIVENS TRANSFORMATION TO
C      ROTATE COLUMNS J AND J-1 SUCH THAT THE ELEMENT A(J-JOFF,J) (IN THE
C      UPPER TRIANGLE) IS ANNIHILATED.
C
C      RECALL THAT A IS STORED IN C WITH
C      A(I,J)=C(I,J-I+2)  I .LE. J .LE. I+M
C
C      INTEGER NDIM,N,M,MP3,NU,NV,J,JOFF
C      DOUBLE PRECISION C(NDIM,MP3),U(NDIM,NU),V(NDIM,NV)
C      LOGICAL WITHU,WITHV
C      INTEGER I,IPK,K,JMIP1,JMIP2,JK1,JK2
C      DCUBLB PRECISION X,Y,Z,COST,SINT,TEMP,S,DABS,DSQRT
C
C      I=J-JOFF
C      JMIP1=J-I+1
C      JMIP2=J-I+2
C      X=C(I,JMIP1)
C      Y=C(I,JMIP2)
C
C      IF Y IS ZERO THERE IS NOTHING TO CO
C      IF (Y.EQ.0.D0) RETURN
C      PERFORM Z=SQRT(X*X+Y*Y); COST=X/Z; SINT=Y/Z WITH SCALING TO
C      PREVENT UNDERFLOW
C      S=DABS(X)+DABS(Y)
C      CCST=X/S
C      SINT=Y/S
C      Z=DSQRT(COST*COST+SINT*SINT)
C      CCST=COST/Z
C      SINT=SINT/Z
C      C(I,JMIP1)=Z*S
C      C(I,JMIP2)=0.D0
C      DO 100 K=1,JOFF
C          JK1=JMIP1-K
C          JK2=JMIP2-K
C          IPK = I+K
C          TEMP=C(IPK,JK1)
C          C(IPK,JK1)=COST*TEMP + SINT*C(IPK,JK2)
C          C(IPK,JK2)=-SINT*TEMP + COST*C(IPK,JK2)
100      CONTINUE
C
C      UPDATE V (ACCUMULATE TRANSFORMATIONS)
C      MUST UPDATE V ON THE RIGHT SINCE V IS DESIRED, NOT V TRANSPOSED
C      IF (.NOT. WITHV) RETURN
C      DO 200 K=1,N
C          TBHP=V(K,J-1)
C          V(K,J-1)=COST*TEMP + SINT*V(K,J)
C          V(K,J)=-SINT*TEMP + COST*V(K,J)
200      CONTINUE
C      RETURN
C
C ***** END OF BIEAND *****
C      END

```

```

C      SUERCUTINE SVDEI (NDIM, N, S, T, NU, NV, WITHU, UITHV, U, V, ETA )
C ***** START OF SVCBI *****
C
C      INTEGER          NDIM, N, NU, NV
C      DCUBLE PRECISION S(N), T(N), U(NDIM,NU), V(NDIM,NV), ETA
C      LCGICAL          WITHU, WITHV
C
C      -----
C
C      THIS IS ESSENTIALLY THE SECOND HALF OF SUERCUTINE DSVD,
C      A SINGULAR VALUE DECOMPOSITION ROUTINE IN THE CSD LIBRARY.
C
C      THE ROUTINE IS IN DCUBLE PRECISION.
C
C      CSVD ORIGINAL PROGRAMMER: R. C. SINGLETON
C      DSVD 360 VERSION BY:      3. G. LEWIS
C      CSVD LAST REVISION:      JANUARY 1974
C      SVCBI EXTRACTED BY:      H. L. OVERTON
C      SVCBI EXTRACTED IN:      AUGUST 1975
C      SVCBI LAST REVISION:     SEPTEMBER 1975
C
C      -----
C
C      ADDITIONAL SUBROUTINE NEEDED:  DECIAT
C
C      -----
C
C      THIS SUBROUTINE COMPUTES THE SINGULAR VALUE DECOMPOSITION
C      OF A REAL BIDIAGONAL N*N MATRIX Z, I.E. IT COMPUTES MATRICES
C      P, S AND Q SUCH THAT
C
C          J = F * S * QT ,
C
C      WHERE
C
C          P IS AN N*N MATRIX AND FT * P = I, (PT = TRANSPOSE
C                                           OF P) .
C          Q IS AN N*N MATRIX AND QT * Q = I, (QT = TRANSPOSE
C                                           OF Q) .
C
C      AND S IS AN N*N DIAGONAL MATRIX.
C
C      THE METHOD USED IS A VARIANT OF THE QR ALGORITHM.
C      REFERENCE: GOLUB AND REINSCH, SINGULAR VALUE DECOMPOSITION
C      AND LEAST SQUARES SOLUTION, NUMER. MATH. 14, 403-420 (1970),
C      SECTION 1.3.
C
C      DESCRIPTION OF PARAMETERS:
C
C      S = REAL N*1 ARRAY.  ON ENTRY S CONTAINS THE MAIN DIAGONAL OF Z.
C      -THE ROUTINE REPLACES THIS BY THE DIAGONAL OF THE MATRIX S,
C      I.E., THE SINGULAR VALUES OF J IN DESCENDING ORDER.
C
C      T = REAL N*1 ARRAY.  ON ENTRY T CONTAINS THE SUPERDIAGONAL OF J
C      IN ELEMENTS 2,...,N; THE FIRST ELEMENT IS ARBITRARY.
C      THE ARRAY IS DESTROYED BY THE ROUTINE.
C
C      N = INTEGER VARIABLE.  THE NUMBER OF ELEMENTS IN ARRAYS S AND T,
C      I.E. THE ORDER OF THE BIDIAGONAL MATRIX J.

```



```

C
C      NU, NV = INTEGER VARIABLES.  TBB  NUMBER OF COLUMNS IN THE
C      ARRAYS U AND V.  SET NU TO N IF UITHU = .TRUE., '1 OTRBRUISB.
C      SIMILARLY SET NV TO N OR 1.
C
C      UITHU, WITHV = LOGICAL VARIABLES.  IF WITHU = .TRUE., THEN
C      THE MATRIX U SUPPLIED IN THE ARRAY U IS POSTMULTIPLIED
C      BY THE MATRIX P.
C      IF WITHV = .TRUE., THEN THE MATRIX V SUPPLIED IN THE
C      ARRAY V IS POSTMULTIPLIED BY THE MATRIX Q.
C
C      U = BBAL NDIH * NU ARRAY.
C
C      V = REAL NDIH * NV ARRAY.
C
C      SUBROUTINE DSVD IS A REAL VERSION OF A FORTRAN SUBROUTINE
C      BY BUSINGER AND GOLUB, ALGORITHM 358:  SINGULAR VALUE
C      DECOMPOSITION OF A COMPLEX MATRIX, COMM. ACM, V. 12,
C      NO. 10, PP. 564 - 365 (OCT. 1969).
C      WITH REVISIONS BY RC SINGLETON, MAY 1972.
C      -----
C
C      DOUBLE PRECISION B,W,CS,SN,  F,  X,BPS,  6,  Y
C      DOUBLE PRECISION H, Q
C      DOUBLE PRECISION DSQRT, DABS, DMAX1
C      INTEGER          I, J, K, L, L1
C
C
C      S(1) = 0.D0
C
C
C      THIS CALCULATION OF EPS IS TAKEN FROM THE RIDDLE OF THE FIRST HALF
C      OF DSVD
C
C      EPS = 0.D0
C      DO 50 K=1,N
C 50  EPS = DMAX1(EPS,DABS(S(K)) + DABS(T(K)))
C      TCLBEANCB FOR NEGLIGIBLE ELEMENTS
C 100 BPS = BPS * ETA
C
C      THE REST OF THE PROGRAM IS THE SECOND HALF OF DSVD
C
C      QR DIAGONALIZATION
C      K = N
C
C      TEST FOR SPLIT
C 230  L = K
C 240  IF (DABS(T(L)) .LE. EPS) GOTO 290
C      L = L - 1
C      IF (DABS(S(L)) .GT. BPS) GOTO 240
C
C      CANCELLATION
C      CS = 0.0D0
C      SB = 1.0D0
C      L1 = L
C      L = L + 1
C      DO 280 I = L, K
C      F = SN * T(1)
C      T(I) = 0.0D0
C      IF (DABS(F) .LE. EPS) GOTO 290

```

```

      H = S(1)
      W = DSQRT (F*F + H*H)
      S(X) = W
      CS = H / W
      SN = -F / W
      IF (WITHU) CALL DBOTAT (U(1,L1), U(1,I), CS, SN, N)
280    CONTINUE
C
C    TEST FOR CCONVERGENCE
290    W = S(K)
      IF (L.EQ. K) GOTC 360
C
C    ORIGIN SHIFT
      X = S(L)
      Y = S(K-1)
      G = T(K-1)
      H = T(K)
      F = ((Y - W) * (Y + W) + (G - H) * (G + H)) / (2.0D0*H*Y)
      G = DSQRT (F*F + 1.0D0)
      IF (F.LT. 0.0D0) G = -G
      F = ((X - W) * (X + W) + (Y / (F + G) - H) * H) / X
C
C    QF STEP
      CS = 1.0D0
      SN = 1.0D0
      L1 = L + 1
      DC 350 I = L1,K
        G = T(I)
        Y = S(I)
        H = SN * G
        G = CS * G
        W = DSQRT (H*H + F*F)
        T (I-1) = W
        CS = F / W
        SN = H / W
        F = X*CS + G*SN
        G = G*CS - X*SN
        H = Y * SN
        Y = Y * CS
        IF (WITHV) CALL DROTAT (V(1,I-1), V(1,I), CS, SN, N)
        W = DSQRT (H*H + F*F)
        S (I-1) = W
        CS = F / W
        SW = H / W
        P = CS*G + SN*Y
        X = CS*Y - SN*G
        I? (WITHU) CALL DRCTAT (U(1,I-1), U(1,I), CS, SN, N)
350    CONTINUE
C
      T (L) = 0.0D0
      T (K) = F
      S (K) = X
      GCTO 230
C
C    CCONVERGENCE
36C    IF (W.GE. 0.0D0) GOTO 380
      S(K) = -W
      IF (.NOT.WITHV) GOTO 380
      DO 370 3 = 1,N
370    V (J,K) = -V (J,K)

```

```

380      K = K - 1
      IF (K . NE. 0) GO TO 230
C
C      SORT SINGULAR VALUES
DO 450 K = 1,N
      G = -1,ODO
      DO 390 I = K,N
          IF (S (I) .LT. G) GOTO 390
          G = S (I)
          J = I
390      CONTINUE
      IF (3 .EQ. K) GOTO 450
      S (J) = S (K)
      S (K) = G
      IF (.NOT.WITHV) GOTO 410
      DO 400 I = 1,N
          Q = V (I,J)
          V (I,J) = V (I,K)
          V (I,K) = Q
400      IF (.NOT.WITHU) GOTO 430
410      DC 420 I = 1,N
          Q = U (I,J)
          U (I,J) = U (I,K)
          U (I,K) = Q
420      CONTINUE
430      CCNTINUE
450      CCNTINUE
C
      RETURN
      END

```

```

SUBROUTINE DROTAT (X, Y, CS, SN, N)
INTEGER          N
DOUBLE PRECISION CS, SN, X(N), Y(N)

```

C
C

```

DOUBLE PRECISION XX
INTEGER          J

```

C
C

```

DC 10 J = 1,N
    XX = X(J)
    X(J) = XX*CS + Y(J)*SN
10    Y(J) = Y(J)*CS - XX*SN
    RETURN

```

C
C
C
C
C

```

***** END OF SVIBI *****

```

```

***** END OF SVEUTH *****

```

```

END

```

```

C      +-----+
C      +  SAMPLE MAIN PROGRAM  +
C      +-----+
C
C      EFFCPEP LENGTHS OF MATRICES :
C
C      D (Q) ,X (M*Q) ,Y (N*Q)
C
C      DCUBLE PRECISICN D (20) ,X (8000) ,Y (2000) ,EPS
C      INTEGER I,IICODE,IORTHG,H,MMAX,M,N,G,FINIT,Q
C      EXTENAL AX
C
C      CCMCN A (3000) ,IINDEX (3000) ,JINDEX (3000) ,NCATA
C      LCUBLE PRECISIGN A
C      INTEGER IINDEX,JINDEX,NCATA
C      INTEGER K,KF1,KP5,KPJ,NCARD,NCATA
C
C      ICUT IS COUTPUT UNIT NUMBER
C      MCHEPS IS MACHINE PRECISICN
C
C      INTEGER LOUT
C      DCUBLE PRECISICN MCHEPS
C      DATA LOUT/6/
C      DATA MCHEPS/2.22D-16/
C
C      NCARD IS NUMBEF OF DATA CARDS TO BE READ
C
C      READ (5,5010) M,N,NCARD
501C  FORMAT (3I5)
C
C      K = 0
C      GO 1C I = 1,NCARD
C      KF1 = K+1
C      KP5 = K+5
C      READ (5,5020) (IINDEX (L) ,JINDEX (I) ,A (I) ,L=KF1,KP5)
502C  FORMAT (5 (2I3,F10.6) )
C      K = K+5
C      10 CCNTINUE
C
C      NCATA IS NUMBEF CF NON-ZERO ELEMENTS IN A
C      IINDEX = 0 SIGNIFIES END OF DATA INPUT
C
C      NCATA = K
C      K = K-5
C      DC 15 J = 1,5
C      KEJ = K+J
C      IF (IINDEX (KPJ) .GT.0) GO TO 15
C      NCATA = KFJ-1
C      GC TO 17
C      15 CCNTINUE
C
C      17 CCNTINUE
C      Q = 10
C      EINIT = 2
C      G = 9
C      MMAX = 2000
C      EES = 1.D-3
C      H = 0
C      ICFTBG = 0

```

```

C      WRITE(LCUT,6010) M,N,Q,PINIT,G,MMAX,EPS,H,IOPTHG
6010  FCIMAT(24H INITIAL PARAMETERS ... /5X,
1      4H M =,I4,5X,4H N =,I4,5X,4H Q =,I4,5X,
2      8H PINIT =,I4,5X,4H G =,I4/5X,7H MMAX =,I5,5X,
3      6H EPS =,1FD10.3,5X,4H H =,I4,5X,9H IORTHG =,I4)
C
      CALL MAXVAL(M,N,C,PINIT,G,MMAX,EPS,AX,H,E,X,Y,IORTHG,
1      LOUT,PCBEPS,IECCDE)
C
      WRITE(LOUT,6020)
602C  FCFHAT (35H ***** USING ELOCK LANCZOS ***** )
      WRITE(LOUT,6030) H,IECODE
6030  FCFMAT(8H ** H =,I4,13H ** IECODE =,I4)
      IF (H.EQ.0) STOP
      WRITE(LOUT,6040) (I(I),I=1,H)
6040  FCFMAT(20H SINGULAR VALUES . .. /5H ** ,6(1PD25.15/5X) )
      STCP
      END

```

```

SUBROUTINE AX(M,N,P,U,V,CRIG)
  INTEGER M,N,P
  LOGICAL PBECISICN U(M,P),V(N,P)
  LOGICAL ORIG

C
C      AX COMPUTES  $X = A*Y$  IF CRIG IS TRUE, AND  $Y = A'*X$ 
C      IF ORIG IS FALSE. X IS STORED IN U AND Y IS STORED IN V.
C
CCEHCN A(3000),IINDEX(3000),JINDEX(3000),NCATA
  LOGICAL PBECISICN A
  INTEGER IINDEX,JINDEX,NCATA
C
  INTEGER I,J,K,L
C
  IF (.NOT.ORIG) GO TO 100
C
  CCMPUTE X = A*Y
C
  DO 20 K = 1,P
C
    DO 10 L = 1,M
      U(L,K) = 0.00
    10 CCNTINUE
C
  20 CCNTINUE
C
    DO 40 L = 1,NCATA
      I = IINDEX(L)
      J = JINDEX(L)
C
      DO 30 K = 1,P
        U(I,K) = U(I,K) + A(L)*V(J,K)
      30 CCNTINUE
C
  40 CCNTINUE
C
  RETURN
C
  CCMPUTE Y = A'*X
C
  100 CCNTINUE
C
  DO 120 K = 1,P
C
    DO 110 L = 1,N
      V(L,K) = 0.00
    110 CCNTINUE
C
  120 CCNTINUE
C
    DO 140 L = 1,NCATA
      I = IINDEX(L)
      J = JINDEX(L)
C
      DO 130 K = 1,P
        V(J,K) = V(J,K) + A(L)*U(I,K)
      130 CCNTINUE
C
  140 CCNTINUE

```

RETURN
END

A BLOCK LANCZOS METHOD TO COMPUTE THE SINGULAR VALUES AND
CORRESPONDING SINGULAR VECTORS OF A MATRIX

Gene H. Golub, Franklin T. Luk, and Michael L. Overton^{*}
Stanford University

Key Words and Phrases: Block Lanczos method, singular values, singular vectors, large sparse matrix.

CR Categories: 5.14

Language: FORTRAN

Description: This algorithm is complement to [1], where we describe the theory and development of the block Lanczos algorithm.

References:

[1] Golub, G., Luk, F., and Over-ton, M., "A Block Lanczos Method to Compute the Singular Values and Corresponding Singular Vectors of a Matrix," submitted to ACM Trans. Math. Software.

Algorithm

^{*}Research supported in part under Army Research Grant DAHC04-75-G-0195 and in part under National Science Foundation Grant MSC75-13497-A01.

