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 u_i and v_i are the left and right singular vectors corresponding to the positive singular value σ_i of A, then $\begin{pmatrix} u_i \\ v_i \end{pmatrix}$ and $\begin{pmatrix} u_i \\ -v_i \end{pmatrix}$ 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.

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

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

2.1 Restricting A to a Subspace of Interest

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

$Y_o^t Y_o = I$. 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_o and the right singular vectors of $\bar{A}$ are orthogonal to the columns of Y_o. 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 $\begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix}$.

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_o and Y_o, resp.. Define

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

Note

$$Q^t Q = \frac{1}{2} \begin{pmatrix} 2I & & \\ & 21 & \\ & & 2I \end{pmatrix} = I .$$

Consider

$$B = Q^t \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} Q$$

$$\equiv C + \Delta \quad ,$$

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

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

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

$$(X_1^t \quad Y_1^t)\begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix}\begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix}$$

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

$$= (X_1^t \quad Y_1^t)R \ .$$

Now

$$\|\Delta\| = \tfrac{1}{2}\|(Y_1^tA^tX_o + X_1^tAY_o \qquad Y_1^tA^tX_o - X_1^tAY_o)\|$$

$$= \tfrac{1}{2}\|(X_1^t \quad Y_1^t)\begin{pmatrix} 0 & A \\ \doteq A & 0 \end{pmatrix}\begin{pmatrix} X_o & X_o \\ Y_o & -Y_o \end{pmatrix}\|$$

$$= \tfrac{1}{2}(X_1^t \quad Y_1^t)R\|$$

$$\leq \tfrac{1}{2}\|(X_1^t \quad Y_1^t\ \| \quad \|R\|$$

$$= \frac{1}{\sqrt{2}}\|R\| \ .$$

If all the $\|\underset{\sim}{\xi}_i\|$ and $\|\underset{\sim}{\eta}_i\|$ were small, then $\|\Delta\|$ would be small also. For example, if

$$\|\underset{\sim}{\xi}_i\| = \epsilon_i \ ,$$

and

$$\|\underset{\sim}{\eta}_i\| = \delta_i \ ,$$

then

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

and

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

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 $\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$ approximate the $(h+1),(h+2),\ldots,g$ 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 × 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

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

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

Q_i is an $n \times p$ matrix such that $Q_i^t Q_i = I$,

P_i is an $m \times p$ matrix such that $P_i^t P_i = I$,

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

Thus,

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

and

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

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

$$\begin{pmatrix} P_1^t & 0 \\ P_2^t & 0 \\ \vdots & \vdots \\ P_s^t & 0 \\ 0 & Q_1^t \\ 0 & Q_2^t \\ \vdots & \vdots \\ 0 & Q_s^t \end{pmatrix} \begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix} \begin{pmatrix} P_1 & P_2 & \cdots & P_s & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & Q_1 & Q_2 & \cdots & Q_s \end{pmatrix} = \left(\begin{array}{cc|cc} & & A_1 & B_1^t \\ & & & \ddots \\ & & & A_{s-1}\ B_{s-1}^t \\ & & & A_s \\ \hline A_1^t & & & \\ B_1 & A_2^t & & \\ & \ddots & & \\ & B_{s-1}\ A_s^t & & \end{array}\right),$$

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

So our algorithm to generate the block bidiagonalmatrix $(\ )^s$ 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

$\begin{pmatrix} 0 \\ Q_1 \end{pmatrix}, \begin{pmatrix} P_1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ Q_2 \end{pmatrix}, \begin{pmatrix} P_2 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ Q_s \end{pmatrix}, \begin{pmatrix} P_s \\ 0 \end{pmatrix}$ form a sequence of orthonormalmatrices. Therefore $\{P_i\}$ and $\{Q_i\}$ are two sequences of orthonormal matrices.

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

to work with the original matrix A. Consider

$$\begin{pmatrix} 0 & \bar{A} \\ \bar{A}^t & 0 \end{pmatrix}$$

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

$$= \begin{pmatrix} (I-X_oX_o^t) & 0 \\ 0 & (I-Y_oY_o^t) \end{pmatrix} \begin{pmatrix} 0 & A \\ A^t & 0 \end{pmatrix} \begin{pmatrix} (I-X_oX_o^t) & 0 \\ 0 & (I-Y_oY_o^t) \end{pmatrix},$$

and the fact that $\begin{pmatrix} 0 \\ Q_1 \end{pmatrix}, \begin{pmatrix} P_1 \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ Q_s \end{pmatrix}, \begin{pmatrix} P_s \\ 0 \end{pmatrix}$ 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 × p matrix Q_1 such that $Q_1^tQ_1 = I$. Compute

$$\hat{P}_1 := AQ_1$$

and

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

Factorize $\hat{P}_1$ such that

---

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

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

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

(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

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

(2) Compute

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

and

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

Factorize $\hat{P}_i$ such that

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

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 $\sigma_1 \geq \sigma_2 > \cdots \geq \sigma_n \geq 0$ be the singular values of the $m \times n$ restricted matrix $\bar{A}$ and let $\sigma_1^{(s)} \geq \sigma_2^{(s)} > \cdots \geq \sigma_{ps}^{(s)} \geq 0$ 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 $\tilde{Q}_1^t Q_1 = I$ 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 $k = 1,2,\ldots,p$, we obtain

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

where

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

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

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

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? = 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

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

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

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

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

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

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

$$\gamma_2 = \frac{0.9 - 0.5}{0.9 + 1.0} \doteq 0.21 ,$$

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

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

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

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

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:

$$\begin{cases} \hat{Q}_i := A^t P_{i-1} - Q_{i-1} A_{i-1}^t , \\ \hat{P}_i := A Q_i - P_{i-1} B_{i-1}^t , \end{cases} \qquad 2 \le i \le s .$$

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 ps × ps 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 × p blocks which form the block diagonal of J are upper triangular and the p × 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 trans-formations. 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 Goiub 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 eigen-values, 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

$$P^{(i,j)} = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \end{array} \\ \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & c & & d & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ & & & -d & & c & \\ & & & & & & & 1 \\ & & & & & & & & \ddots \end{bmatrix} \end{array}$$

where $c^2 + d^2 = 1$ 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

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

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

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.

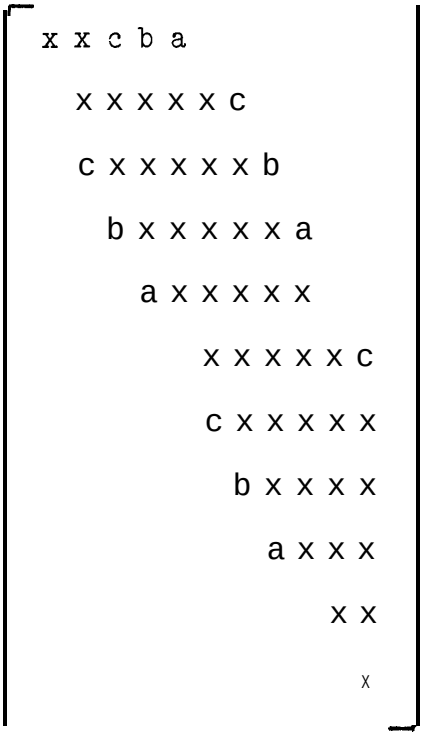
Reducing the matrix to bidiagonal form in this way requires approximately $4pt^2$ multiplications using ordinary Givens transformations, or $2pt^2$ using "fast Givens", assuming $1 \ll p \ll t$. This compares with a count of approximately $4t^3/3$ multiplications required to do the reduction by the standard 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



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 $\ell n\ p$ if vectors are required.

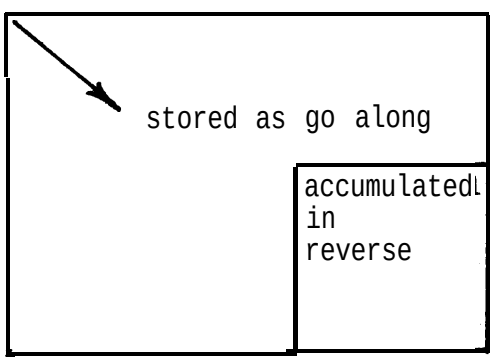
Let us consider now a method we shall call Rand Householder. This follows an idea suggested in Rutishauser [9] for the corresponding eigenvalue tridiagonal reduction problem. Recall that a Householder transformation matrix $Q^{(i,j,p)}$ can be chosen to have the property that when applied to A on the left the resulting matrix $A' = Q^{(i,j,p)}A$ has zeros in positions $i+1,\ldots,j$ of column p but is different from A only in rows $i,\ldots,j$ 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 $a_{12}, a_{13}, "_{14}$ 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 x 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) x (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 x 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) x (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 $1 << p << t$ 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.

Method	Reduction Without Vectors	Accumulating the Vectors	Total When Vectors Desired	Total Using Fast Givens
Golub-Reinsch	$\sum_{i=1}^{t-2} 2(2)(t-i)^2 \sim \frac{4t^3}{3}$	$\sum_{i=1}^{t-2} 2(2)(t-i)^2 \sim \frac{4t^3}{3}$	$\frac{8t^3}{3}$	---
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	$\sum_{k=2}^{p} \sum_{i=1}^{t-2} \frac{t-(i+k)+1}{k} (8)(k+1) \sim 4pt^2$	$\sum_{k=2}^{p} \sum_{i=1}^{t-2} \frac{t-i}{k} (8)t \sim 4t^3 \sum_{k=2}^{p} \frac{1}{k} \sim 4(\ell n\ p-\frac{1}{2})t^3$	$4(\ell n\ p-\frac{1}{2})t^3$	$2(\ell n\ p-\frac{1}{2})t^3$
Band Householder	$\sum_{i=1}^{t-2} \frac{t-i}{p} (2) \sum_{k=1}^{p-1} 2(k)(k+p) \sim \frac{5}{3} p^2t^2$	$\sum_{i=1}^{t-2} \frac{t-i}{p} (2) \sum_{k=1}^{p} 2(k)t \sim pt^3$	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 $$P = (P_1, P_2, \ldots, P_s) ,$$

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

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 $\underset{\sim}{x}$ of unit length, if

$$\|A\underset{\sim}{x} - \mu\underset{\sim}{x}\| = \delta$$

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

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

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

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

$$= \|A\underset{\sim}{v} - \sigma_i^{(s)}\underset{\sim}{u}\|^2 + \|A^t\underset{\sim}{u} - \sigma_i^{(s)}\underset{\sim}{v}\|^2 \ .$$

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

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

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 $\bar{A}$ and not of A. Thus if we compute the residuals with respect to A and not to 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

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

then

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

where mcheps = 2.20×10^{-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

$$\left(\|A\underset{\sim}{v}_i - \sigma_1^{(s)}\underset{\sim}{u}_i\|^2 + \|A^t\underset{\sim}{u}_i - \sigma_i^{(s)}\underset{\sim}{v}_i\|^2 \right)^{1/2} \leq \hat{\tau}_{ii}\sigma_i^{(s)}$$

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_o, the step size is s_o, and the bound on $p_o s_o$ is q_o. Assume that g singular values are to be computed and $g_o (1 \leq g_o < g)$ 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 $p_o \geq g$, then

$$Pl := p_o - g_o \ ,$$

and

$$s_1 := \left\lfloor \frac{q_o - g_o}{p_1} \right\rfloor \ .$$

(Here $\lfloor a \rfloor$ denotes the integer part of α.)

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 $p_o < g$, 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,

$$p_1 := \min(p_o, g-g_o) \ ,$$

$$s_1 := \left\lfloor \frac{q_o - g_o}{p_1} \right\rfloor .$$

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

$$p_1 := \left\lfloor \frac{q_o - g_o}{2} \right\rfloor \ ,$$

$$s_1 := \left\lfloor \frac{q_o - g_o}{p_1} \right\rfloor \ .$$

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_o columns of QY are usually better approximations than Q_1 to the p_o right singular vectors corresponding to the p_o largest singular values of A. If $g_o = 0$, then those p_o columns of QY will serve as a good starting matrix for another Lanczos iteration. If $g_o > 0$, then the $(g_o+1),\ldots,(g_o+p_1)$-th columns will be chosen as the starting matrix for the next iteration. We have seen that the $(g_o+1),\ldots,p_o$-th columns of QY are usually good approximations to the $(g_o+1),\ldots,p_o$-th right singular vectors of A. Our experimental results show that the other columns are usually rich in the direction of the $(p_o+1),\ldots,p_1$-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 X 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 $A\underset{\sim}{x}$ and $A^t\underset{\sim}{x}$ 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 × p matrix and V is an n × 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 X q.

Y : an array of length at least n × 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.

mcheps : 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 $\bar{A}$ 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 [l].

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 underestimate.

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 eps; we have obtained satisfactory results from 1000 x 999 matrices with eps = 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 X 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, eps = 10^{-3} and 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 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, eps = 10^{-3} and iorthg = 0.

	p = 1	p = 2	p = 3	p = 4
σ_1	2	$2\text{-}2 \times 10^{-15}$	$2\text{-}2 \times 10^{-11}$	$2\text{-}4 \times 10^{-11}$
σ_2	2	$2\text{-}1 \times 10^{-9}$	$2\text{-}3 \times 10^{-11}$	$2\text{-}6 \times 10^{-11}$
σ_3	$2\text{-}1 \times 10^{-15}$	$2\text{-}2 \times 10^{-8}$	$2\text{-}3 \times 10^{-10}$	$2\text{-}5 \times 10^{-10}$
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, eps = 10^{-3} and iorthg = 0.

	p = 1	p = 2	p = 3
σ_1	$10\text{-}2 \times 10^{-15}$	$10\text{-}3 \times 10^{-15}$	$10\text{-}4 \times 10^{-12}$
σ_2	$10\text{-}3 \times 10^{-15}$	$10\text{-}2 \times 10^{-9}$	$10\text{-}2 \times 10^{-11}$
σ_3	$2\text{-}5 \times 10^{-7}$	$10\text{-}2 \times 10^{-7}$	$10\text{-}2 \times 10^{-8}$
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 $g = 3$, $q = 12$, eps = 10^{-3} and $p = 1$. We run our program with no, partial, and complete reorthogonalization.

	iorthg = 0	iorthg = 1	iorthg = 12
σ_1	$10+1\times10^{-14}$	10	10
σ_2	$10-1\times10^{-15}$	$10-2\times10^{-12}$	$2-2\times10^{-15}$
σ_3	$2-6\times10^{-15}$	$2-4\times10^{-15}$	$2-4\times10^{-15}$
iter	1	1	3
imm	31	31	71
iw	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 $0.002,-0.003,0.004,-0.005,\ldots,1.000$. We choose g = 3, $q=12, eps=10^{-3}$ and iorthg = 0.

	p = 1	p = 2	p = 3
σ_1	0.999992	[illegible]	0. 999986
σ_2	0.998960	0. 998951	.998999
σ_3	0.998036	0. 998005	.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 $A\underset{\sim}{x}$ and $A^t\underset{\sim}{y}$ efficiently.

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

```
            DO 10 K = 1,M
                 X(K) = 0.D0
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 $\underset{\sim}{y} = A^t\underset{\sim}{x}$:

```
            DO 110 K = 1,N
                 Y(K) = 0.D0
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 ($\doteq$ 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 ($\doteq$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, eps = 10^{-3} and 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 $O(eps^2)$.

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 X 10 -7 , 1.91 x 10-7 , 5.93 X 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, eps = 10^{-3} and 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, eps = 10^{-3} and 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,MMAX,EPS,OP,H,D,X,Y,IORTHG,
     1                  LOUT,MCHEPS ,IECODE)
      INTEGER M,N,Q,PINIT,G,MMAX,H,IORTHG,LOUT,IECODE
      DOUBLE PRECISION EPS,D(Q),X(M,Q),Y(N,Q),MCHEPS
      EXTERNAL OP
C
C
C     ------------------------------------------------------------------
C
C     CALCULATE TEE LARGEST SINGULAR VALUES OF A LARGE SPARSE MATRIX
C
C     WRITTEN BY :  FRANKLIN LUK
C                   COMPUTER SCIBBCE DEPARTMENT
C                   STANFORD UNIVERSITY
C                   SBPTBRBER 1976
C     LAST UPDATE : APRIL     1977
C
C     ------------------------------------------------------------------
C
C     TBIS SBT OP ROUTINES USES INTEGER AND DOUBLB PRECISION ARITHMETICS
C
C     ------------------------------------------------------------------
C
C     THIS SBT OP ROUTINES INCLUDES : MAXVAL, BKLANC, ORTHOG, INPROD,
C                                     ROTATB, CNVTST, PCEOIC, RANDOM,
C                                     AND SVBUTH( PLUS BIBAND, ROTROW,
C                                     ROTCOL, SVDBI, AND DROTAT ).
C
C     ------------------------------------------------------------------
C
C
C        THIS SUBROUTINE IS THE MAIN SUBROUTINE IMPLEMENTING
C     THE ITERATIVE BLOCK LANCZOS METHOD FOR COMPUTING THE LARGEST
C     SINGULAR VALUES AND CORRESPONDING LEFT AND RIGHT SINGULAR VECTORS
C     OF AI M-BY-N MATRIX.
C
C     DESCRIPTION OP PARAMETERS :
C
C     M,N :     INTEGER VARIABLES. TEE NUMBER OF ROWS AND COLUMNS
C               OP TEB MATRIX A WHOSE SINGULAR VALUES AND VECTORS
C               ARE BEING COMPUTED. IT IS ASSUMED TRAT 2 .LE. N .LB. M.
C
C     Q :       INTEGER VARIABLE. THE NUMBER OF VECTORS OP LENGTH M
C               CONTAINED II TEE ARRAY X, AND THE NUMBER OF VBCTORS
C               OF LENGTH N CONTAINED IN THE ARRAY Y. THE VALUB OP Q
C               SHOULD BE LESS THAN OR EQUAL TO 26, AT LBAST ONE GREATER
C               TEAM TEE VALUE OF 6 AND LBSS TEAR OR EQUAL TO N.
C
C     PINIT :   INTEGER VARIABLE. THE INITIAL BLOCK SIZE TO BE USED
C               IN THB BLOCK LANCZOS METHOD. I? PINIT IS NEGATIVE,
C               THEN -PINIT IS USED FOR TEB BLOCK SIZE AND COLUMNS
C               M+1, . . . . M+(-PINIT) OF THE ARRAYS Y ARE ASSUMED
C               TO BE INITIALIZED TO A MATRIX USED TO START TEB BLOCK
C               LANCZOS HBTHOD. IF THE SUBROUTINE TERMINATES WITH
C               A VALUE OF M LBSS THAI 6, THEN PINIT IS ASSIGNED
C               A VALUE -P, WHERE P IS THB FINAL BLOCK SIZE CEOSBR.
C               IN THIS CIRCUMSTANCE, COLUMNS M+1, . . . , M+P OF Y
C               WILL CONTAIN THB HOST RECENT SET OF RIGET SINGULAR
C               VICTOR APPROXIMATIONS WHICH CAN BE USED TO RESTART
```

```
C               THE SUBROUTINE IF DESIRED.
C
C     G :       INTEGER VARIABLE. TEB UUHBBR OR SIRGULAR VALUBS AND
C               SINGULAR VECTORS BEING COHPUTBD. THAT IS, MAXVAL
C               ATTEMPTS TO COHPUTB ACCURATE APPROXIMATIONS TO TEE
C               6 LARGEST SIRGULAR VALUES 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*X AND TRANSPOSE(A)*X, WHERE X IS AN APPRO-
C               PRIATE VBCTOR, TEAT ARE ALLOUBD DURING ONE CALL OF
C               THIS SUBROUTINE TO COMPLETE ITS TASK OP COMPUTING
C               G SINGULAR VALUES 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 SIRGULAR VALUBS
C               AND VECTORS OP A. FOR SINGULAR VALUES LESS IN MODULUS
C               THAN 1, BPS UILL BE AU ABSOLUTE TOLBRAUCE.
C
C     OP :      SUBROUTIPB NAME. TEE ACTUAL ARGUMENT CORRESPONDING
C               TO OP SEOULD BB TEB NAME OF 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*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) *U BEING
C               COHPUTED AND STORBD IN V.
C
C     M :       INTEGER VARIABLE. M GIVES THB NUMBER OF SIRGULAR
C               VALUES AND LEFT AND RIGHT SINGULAR VBCTORS ALRBADY
C               COMPUTED. THUS, INITIALLY, M SHOULD BB ZERO.
C               IF M IS GREATER THAN ZERO, THEN ELEMENTS OUB THROUGH
C               M OF THB VICTOR D COUTAIU APPROXIMATIONS TO THE M
C               LARGEST SIGULAR VALUBS OF A, COLUMNS ORB THROUGH M
C               OF THB ARRAYS X AND Y CONTAIN APPROXIMATIONS TO TEE
C               CORRESPONDING LEFT AND RIGHT SIRGULAR VBCTORS,
C               AT EXIT, M CONTAINS A VALUB EQUAL TO THE TOTAL NUMBER
C               OF SINGULAR VALUBS AND LEFT AND RIGHT SIRGULAR VBCTORS
C               COMPUTED INCLUDINGANY ALREADY COHPUTBD WHEN MAXVAL
C               WAS ENTERED. THUS, AT EXIT, THE FIRST MELEMENTSOF D
C               AND THE FIRST M COLUMNS OF X AND Y WILL CONTAIN
C               APPROXIMATIONS TO THEM LARGEST SINGULAR VALUES OF A AND
C               THEIR CORRESPONDING LBPT AND RIGHT SINGULAR VECTORS.
C
C     D :       DOUBLE PRECISION ARRAY. D CONTAINS THB COHPUTBD SINGULAR
C               VALUES. 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               SIRGULAR VECTORS. X SEOULD BE AN ARRAY CONTAINING AT
C               LBAST M*Q ELEMENTS. X IS USED ROT ONLY TO STORB THE LEFT
```

```
C             SIUGULAR VECTORS COMPUTED BY MAXVAL, BUT ALSO AS
C             WORKING STORAGE FOR TEE BLOCK LANCZOS METHOD. AT EXIT,
C             THB PIRST M*H ELEMENTS OF X CONTAIN THE LEFT SINGULAR
C             VECTOR APPROXIMATIONS -- THE FIRST VECTOR IN TEB FIRST
C             M ELEMENTS, THE SBCOUD IN THE SECOND a ELEMENTS, ETC.
C
C     Y :     DOUBLE PRBCISIOU ARRAY. Y CONTAINS THB COMPUTED RIGHT
C             SINGULAR VECTORS. Y SHOULD BE AN ARRAY CONTAINING AT
C             LBAST N*Q ELEMENTS. Y IS USED HOT ONLY TO STORE THE
C             RIGHT SINGULAR VECTORS COMPUTED BY MAXVAL, BUT ALSO AS
C             WORKING STORAGE FOR TEB BLOCK LANCZOS METHOD. AT EXIT,
C             TEE FIRST N*H ELEMENTS OF Y CONTAIN TEB RIGHT SINGULAR
C             VECTOR APPROXIMATIONS -- THB FIRST VECTOR IN THE FIRST
C             N ELEMENTS, THE SECOND IN THE SBCOUD N ELEMENTS, ETC.
C
C     IORTHG : INTEGER VARIABLE. ITS VALUB IS THB NUMBER OF IMMEDIATELY
C             PRECEDING BLOCKS 0? VECTORS WITH RBSPBCT TO WHICH
C             REORTHOGONALIZATION OP THB PRBSBUT BLOCK 0? VBCTORS
C             IS CARRIED OUT.
C
C     LOUT :  INTEGER VARIABLE. OUTPUT UNIT NUMBER.
C
C     MCHEPS : DOUBLE PRECISION VARIABLB. THE MACHINE PRBCISIOU.
C
C     IECODE : INTEGER VARIABLB. TEE VALUE OF IBCODE IUDICATBS
C             WHETHER MAXVAL TERMINATED SUCCESSFULLY, AND I? HOT,
C             TEB REASON WHY.
C                IECODE=0 : SUCCESSFUL TERMINATION.
C                IECODE=1 : THE VALUB 0? N IS LESS THAN TPO.
C                IECODE=2 : TEE VALUB 0? N IS GRBATBR THAN TEB VALUE
C                           OF N.
C                IECODE=3 : THE VALUE OF N IS GRBATBR THAN 1000.
C                IECODE=4 : THE VALUB 0? G IS LBSS THAN ORB.
C                IECODE=5 : THE VALUE 0? Q IS LBSS THAN OR EQUAL TO G.
C                IECODE=6 : THE VALUB OP Q IS GREATER THAN 26.
C                IECODE=7 : THE VALUE 0? Q BXCBBDS N.
C                IECODE=8 : THE VALUB OF MMAX WAS BXCBBDBD BEFORE
C                           G SIUGULAR VALUES AND LEFT AND RIGHT
C                           SINGULAR VECTORS UBRB COMPUTED.
C
C        UOTB TEAT THE SUBROUTINE HAS BEEN DESIGNED TO ALLOW INITIAL
C     APPROXIMATIONS TO THB RIGHT SINGULAR VECTORS CORRES-
C     PONDING TO THB LARGEST SINGULAR VALUES TO BE UTILIZED
C     ( IF THEY UBRB KNOWN ) BY STORING THBH IN Y AND ASSIGNING
C     PINIT MINUS THE VALUE 0? THEIR NUMBER. FURTHERMORE, IT
C     HAS ALSO BEEN DESIGNED TO ALLOW RESTARTING I? IT STOPS WITH
C     IECODE=8. THUS, THE USER OF THIS PROGRAM CAN RESTART IT AFTER
C     EXAMINING ANY PARTIAL RESULTS WITHOUT LOSS OF PREVIOUS WORK.
C
C
      INTEGER I,IERR,IMM,IPH,IPQ,ISEED,ITER,IVV,MCONV,P,PHI,PS,PP3
      INTEGER QPPS,QP1,S
      REAL FLOAT
      DOUBLE PRBCISIOU ERRBND,ERRC
C
C        THE MINIMUM LENGTHS 0? THE LOCAL ARRAYS ARE AS FOLLOWS. THESE
C     COULD BE CHANGED BY THB USER I? NECESSARY BY CHANGING TEB MAXIMUM
C     VALUES 0? Q OR N WHICH AT PRBSBUT ARE 26 AND 1000 ( TEB TESTS
C     BBLOU SHOULD ALSO BE MODIFIED ).
C        LBT Q2 DBUOTB TEB INTEGER PART 0? Q/2, THEN
```

```
C
C             C(Q*(Q2+3)),U(Q*Q),V(Q*Q),R(Q2*Q2),T(M)
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 THB SUBROUTIUB PARA-
C     METERS ARE IN RANGE.
C
      I? (U.LT.2) 60 TO 901
      IF (M.LT.M) GO TO 902
      IF (M.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.M) 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 COUTINUE
C
C        CEOOSB INITIAL VALUES FOR THB BLOCK SIZE P, THE NUMBER S
C     OF STBPS TEAT TUB BLOCK LANCZOS METHOD IS CARRIED OUT, AND
C     CEOOSB AH INITIAL II-BY-P ORTHONORMAL MATRIX 11 TO START
C     TEE BLOCR LANCZOS METHOD.
C
      P = PINIT
      I?  (P.LT.0) P =-P
      S = (Q-M)/P
      IF (S.GE.2) GO TO 120
      s - 2
      P = (Q-M)/2
C
  120 I? (PINIT.LT.0) GO TO 200
C
C     INSERT RANDOM VECTORS INTO COLUMNS M+1 THROUGHM+P OF TEE ARRAY Y.
C
      CALL RANDOM(M,Q,P,M,Y,ISEED)
C
C     SET CONSTANTS FOR LATER CONVERGENCE TESTS.
C
  200 ERRBND = BPS + 10.DO*DBLE(FLOAT(M+M))*MCHEPS
      ERRC  =0.DO
      ITBR = 0
      IMM = 0
      IVV = 0
C
C        THE MAIN BODY OF THE SUBROUTIUB STARTS HERE. IMM
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 VBCTORS.
C
```

```
  300 IF (H.GE.G) GO TO 900
      IF (IMM.GT.MMAX) 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 BEBP LOST.
C
      DO 310 I= l,P
         IPH = I+H
         IF (D(IPH).GT.0.D0) GO TO 310
         PMI = P-I
         CALL RANDOM(N,Q,PMI+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,MCHEPS)
C
C        BKLANC CARRIES OUT TEE BLOCK LANCZOS METHOD AND
C     RETURNS TEE RESULTING BAUDBD UPPER TRIANGULAR MATRIX MS
C     IN C, THE N-BY-PS ORTHONORMAL MATRIX XS IN X AND THB
C     N-BY-PS ORTHONORMAL MATRIX YS IN Y. THB INITIAL
C     N-BY-P ORTHONORMAL MATRIX Yl IS ASSUMED TO BE STORED
C     IN COLUMNS H+1 THROUGH H+P OF Y.
C
      CALL BKLANC(N,M,Q,PP3,H,P,S,OP,C,X,Y,R,IORTHG,IVV,LOUT,MCHEPS)
      IMM = IMM + P*(2*S-1)
C
C        SVBUTH SOLVES THE SINGULAR VALUE PROBLBH FOR THE PS-BY-PS
C     ARRAY MS, RETURNING THB SINGULAR VALUES IN THE SBCOBD COLUMN
C     OF C AND THB RIGHT SINGULAR VECTORS IN THB FIRST P*S COLUMNS
C     OP U, AND TEB P LEFT SINGULAR VECTORS CORRESPONDING TO THB
C     P LARGEST SINGULAR VALUES IN TEE FIRST P COLUMNS OF V.
C
      CALL SVBUTH(Q,PS,P,PP3,C,PS,PS,U,V,MCHEPS,IERR)
      IF (IERR.EQ.0) 60 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 COHPUTBS THE LEFT AND RIGHT SINGULAR VECTORS
C     OF THB 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 I? RELATIVE INCREASE OF COMPUTED SINGULAR VALUES EXCEEDS
C     TEE 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 BABY OF THB SINGULAR VALUES
C     AND LEPT 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)
      IMM = IMM + (NCONV+1)*2
  400 CONTINUE
C
      DO 410 I=  l,PS
         IPH = I+H
         IPQ = I+Q
         D(IPH) = C(IPQ)
  410 CONTINUE
2
C        PCHOIC CHOOSES NEW VALUBS FOR P AND S, THE BLOCK
C     SIZE AID 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) IMM,IVV,NCONV
 6040  FORMAT(5X,6H IMM =,I5,5X,6HIVV =,I5,5X,8H NCONV =,I3)
      ii = H+NCONV
C
      GO TO 300
C
c        THIS IS TEE ENDOPTHEMAIN BODY OF THE SUBROUTINE.
C     NOW SET THE VALUE OP THB IBCODB ANDEXIT.
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,LOUT
      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, TEE NEXT
C     SUPERDIAGONAL BEING STORED IN TBE FIRST PS-1 LOCATIONS
C     OF COLUMN 3, AND SO ON ), AND A PS-BY-PS ORTHOGONAL
C     MATRIX XS WHICH IT STORES IN COLUMNSH+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 TBB N-BY-Q ARRAY Y.
C        MS CAN ALSO BE REGARDED AS A BLOCK UPPER BIAGONAL
C     MATRIX WITH P-BY-F UPPER TRIANGULAR MATRICES R(l), . . . .
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 CORPOSED 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 OP AN EXTERNAL SUBROUTINE USED TO
C     DEFINE TEE MATRIX A.
C
      INTEGER I,I1,I2,J,JMP,J1,J2,K,K1,L,LL,LLMP,LU
      DOUBLE PRECISION T
C
C     L = 1
C
      LL = H+1
      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 = l,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 .GE. 2
C
      IP (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        COMPUTE A'*X(L-1)
C
         LLMP = LL-P
         CALL OP(M,N,P,X(1,LLMP),Y(1,LL),.FALSE.)
C
         DO 230 K = LL,LU
C
C           COMPUTE Y(L-1)*R(L-1)'
C
            K1 = K-LL+1
C
            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              COMPUTE Y(L) = A'*X(L-1) - Y(L-1) * B(L-1)'
C
               Y(I,K) = Y (I,K)-T
  220       CONTINUE
C
  230    CONTINUE
C
C        FACTORIZE Y(L) := Y(L)*T(L)
C
         CALL ORTHOG (N,Q,M,LL-1,P,R,Y,IORTHG,IVV,IOUT,MCHEPS)
C
C        STORE T(L)' IN C
C
         DO 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        COMPUTE A*Y(L)
C
         CALL OP(M,N,P,X(1,LL),Y(1,LL),.TRUE.)
C
         DO 430 A = LL,LU
C
C           COMPUTE X (L-1)*T(L)'
C
            K1 = K-LL+1
C
            DO 420 I =1,M
```

```
                T = 0.DO
C
                DO 410 J = K,LU
                   JMP = J-P
                   31 = J-LI+1
                   T = T + X(I,JMP)*R(K1,J1)
  410           CONTINUE
C
C               COMPUTE X(L) = A*Y(L) - X (L-1)*T(L)'
C
                X(I,K) = X(I,K)-T
  420        CONTINUE
C
  430     CCNTINUE
C
C         FACTORIZE X(L) := X(L)*R(L)
C
          CALL ORTHOG (M,Q,H,LL-1,P,R,X,IORTHG,IVV,IOUT,MCHEPS)
C
C         STORE R(L) IN C
c
          DO 520 J = 1,P
C
             DO 510 I = 1,J
                11 = I+I2
                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,IOUT,MCHEPS)
      INTEGER N,Q,H,L,P,IORTHG,IVV,LOUT
      DCUBLB PRECISION R(P,P),X(N,Q),MCHEPS
C
C        OBTEOG REORTHOGONALIZES TEE N-BY-P MATRIX Z STORED IN
C     COLUMNS L+1 THROUGH L+P OF TEE 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 OP TEE MATRIX X
C     USING GRAM-SCHMIDT ORTHOGONALIZATION. THE MODIFIED
C     GRAM-SCHMIDT METHOD IS USED TO FACTORIZE THE RESULTING
C     MATRIX INTO TEE PRODUCT OF AN N-BY-F ORTHONORMAL MATRIX
C     XORTH STOBED IN COLUHNS L+1 THROUGH L+P OP X, AND
C     A P-BY-P UPPER TRIANGULAR ARRAY R.
C
      INTEGER I,IML,IP1,J,K,KML,L1,LP1,LPP
      INTEGER MAX0
      DOUBLE PRECISION SUM
      DOUBLE PRECISION DSQRT
      IF (P.EQ.0) RETURN
      LP1 = L+1
      LPP = L+P
C
      IP(H.EQ.0) GO TO 200
C
      DC 130 I = LP1,LPP
C
         DO 120 K = 1,H
            CALL INPROD(N,X(1,I),X(1,K),SUM)
C
            DO 110 J = l,N
               X(J,I) = X(J,I) - SUM*X(J,K)
  110       CONTINUE
C
  120    CONTINUE
C
  130 CCNTINUE
C
      IVV = IVV + H*P
C
  200 IF (IORTHG.EQ.0) GO TO 300
      IF (L.EQ.H) GO TO 300
      L1 = MAX0( L-P*IORTHG+1, H+1 )
C
      DC 230 I = LP1,LPP
C
         DO 220 K = L1,L
            CALL INPROD(N,X(1,I),X(1,K),SUM)
C
            DO 210 J = 1,N
               X(J,I) = X(J,I) - SUM*X(J,K)
  210       CONTINUE
C
  220    CCNTINUE
C
  230 CCNTINUE
C
      IVV = IVV + (L-L1+1)*P
C
  300 CCNTINUE
```

```
C
      DC 400 I = LP1,LPP
         SUM = 0.DO
C
         DO 310 J = 1,N
            sun = SUM + X(J,I)**2
  310    CGNTINUE
C
         IML = I-L
         IF (SUM.GT.MCHEPS) GO TO 330
C
         WRITE(LOUT,6010)
 6010    FORMAT(5X,47H *** WARNING •  LINEAR INDEPENDENCEMAY BE LOST,
     1           24H. VECTOR SET TO ZERC ***)
         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
         IP (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 TEE RESULT IN S.
C
      INTEGER I
      SUM = 0.D0
C
      DC 110 I =1,N
         SUM = SUM + U(I)*V(I)
  110 CONTINUE
C
      RETURN
      END
```

```
      SUBRCUTINE ROTATE(N,Q,H,PS,L,U,X,T)
      INTEGER N,Q,H,PS,I
      DOUBLE PRECISION U(Q,L),X(N,Q),T(Q)
C
C        RCTATE COMPUTES TEE PIRST L COLUMNS OF THE MATRIX
C     XS*QS, WHERE XS IS AN N-BY-PS ORTHCNORMAL MATRIX STORED
C     IN CCLUMNS H+1 THROUGH H+PS OP TEE N-BY-Q ARRAY X AND
C     QS IS A PS-BY-PS CRTHONORMAL MATRIX WHOSE FIRST L COLUMNS
C     ARE STORED IN COLUMNS 1 THROUGH L CP TEE ARRAY U. TRB
C     RESULT IS STORED IN CCLUKNS H+1 THECUGH H+L OF X
C     CVERWRITING PART OF XS.
C
      INTEGER I,J,JPH,K,KPH
      DOUBLE PRECISION SUM
C
      DC 200 I = 1,N
C
C        CCMPUTE TEE II-TH ROW OF XS*QS
C
         DO  110 K= 1,L
            SUM = 0.DO
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,OP,C,X,Y,NCONV,
     1                  LOUT,T)
      INTEGER M,N,Q,H,G,NCONV,LOUT
      DOUBLE PRECISION ERRBND,ERRC
      DOUBLE PRECISION C (Q,2),X(M,Q),Y(M,Q),T(M)
C
C        CRVTST DETERMINES WHICH OF TEE P COMPUTED SINGULAR
C     VALUES STORED IN THE SECOND COLUMN OF C HAVE CONVERGED.
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
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 TEE (H+I)-TH SINGULAR VALUE HAS CONVERGED
C     IF
C        RBSIDU  .LE.  B*ERRBND + ERRC,
C
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     COMPUTED SINGULAR VALUE IS GREATER THAN 1, AND AN ABSOLUTE
C     ERROR TEST CTHBRUISB.
C        TEE CONVERGENCE TEST IS PERFORMED IN ORDER CN TEE (H+1)-TH,
C     (H+2)-TH, . . . COMPUTED SINGULAR VALUES. AS SOON AS A COMPUTED
C     VALUE FAILS THE TEST, RETURN IS RACE TO TEE CALLING ROUTINE.
C        NCONV IS THE NUMBER THAT HAS CONVERGED. IF NCONV=0,
C     TEEN NONE HAS CONVBEGBD.
C
      INTEGER I,IPH,K,L,PT
      DOUBLE PRBCISION RESIDU,B,SUM
      DOUBLB PRECISION DSQRT
C
      SUB = 0.DO
      FT = G-H
C
      DO 200 I = 1,PT
         K = I
         IF (C(I,2).EQ.0.DO) GO TO 300
         IPH = I+H
         CALL OP(M,N,1,T,Y(1,IPH),.TRUE.)
C
         RBSIDU = 0.DO
         DO 110 L = 1,M
            B = T(L)  - C(I,2)*X(L,IPH)
            RBSIDU = RESIDU + B**2
  110    CONTINUE
C
         CALL OP (M,N,1,X(1,IPH) ,T,.FALSE.)
C
         DO  120 L =1,N
            B = T(L) - C (I,2)*Y (L,IPH)
            RESIDU = RESIDU + B**2
  120    CONTINUE
C
C        TEST FOR CONVERGENCE
C
```

```
      RESIDU = DSQRT(RBSIDU)
      B = C(I,2)
      IF (B.LT.1.D0) B = 1.D0
      IF (RESIDU.LE.B*ERRBND+ERRC) GO TO 130
C
      WRITE(LOUT,6010) K,RESIDU
 6010 FORMAT(5X,4H K =,I4,5X,9H RBSIDU =,1PD15.5,
     1       36H   • ** COHPUTBD 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,NCONV,P,S
C
C        BASED ON THE VALUES OF Q, H, G AND NCONV,
C     PCHOIC CHOOSES NEW VALUES FOR P AND S, THE BLOCK SIZE
C     AND NUMBER OF STEPS FOR THE BLOCK LANCZOS METHOD.
C        THE STRATEGY IS : IF THE PREVIOUS BLOCK SIZE IS
C     GREATER THAN THE NUMBER OF SINGULAR VALUES TO BE
C     COMPUTED, THEN THE NEW BLOCK SIZE EQUALS THE PREVIOUS
C     BLOCK SIZE MINUS THE NUMBER OF SINGULAR VALUES THAT
C     HAVE CONVERGED IN THE CURRENT ITERATION, OTHERWISE
C     THE NEW BLOCK SIZE IS CHOSEN TO BE THE SMALLER OF THE
C     TWO VALUES : 1) THE PREVIOUS BLOCK SIZE, AND 2) THE
C     NUMBER OF SINGULAR VALUES TO BE COMPUTED. S IS CHOSEN
C     AS LARGE AS POSSIBLE SUBJECT TO STORAGE CONSTRAINT,
C     BUT ITS VALUE IS ALWAYS AT LEAST 2.
C        H IS THE NUMBER OF SINGULAR VALUES AND LEFT AND RIGHT
C     SINGULAR VECTORS THAT HAVE ALREADY BEEN COMPUTED AND G
C     IS THE REQUIRED NUMBER. NCONV IS THE NUMBER OF SINGULAR
C     VALUES AND LEFT AND RIGHT SINGULAR VECTORS THAT HAVE
C     CONVERGED IN THE CURRENT ITERATION.
C
      INTEGER HT,PT
C
      HT = H + NCONV
      IF (P.LE.G-H) GO TO 110
C
      P = P - NCONV
      S = (Q-HT)/P
      RETURN
C
  110 PT = G - HT
      IF (P.GT.PT) P = PT
      S = (Q-HT)/P
      IF (S.GE.2) RETURN
      P = (Q-HT)/2
      S = (Q-HT)/P
C
      RETURN
      END
```

```
      SUEROUTINE RANDOM(N,Q,P,H,X,ISEED)
      INTEGER N,Q,P,H,ISEED
      DOUBLE PRECISION X(N,Q)
C
C        RANDOM COMPUTES AND STORES A SEQUENCE OF P*N PSEUDO-
C     RANDOM INTEGERS ( VALUE BETWEEN 0 AND 2147483647 ) IN
C     COLUMNS H+1 THROUGH H+P OF THE N-BY-Q ARRAY X.
C
      INTEGER I,L,LPH
      DO 130 L = 1,P
         LPH = L+H

         DO 120 I = 1,N
            ISEED = ISEED*314159269 + 453806245
C
C              THE STATEMENT NUMBER 110 IS TO PREVENT UNWANTED,
            OPTIMIZATION BY THE COMPILER.
C
  110       IF (ISEED.LT.0) ISEED = ISEED + 2147483647 +1
            X(I,LPH) = ISEED
  120    CONTINUE
C
  130 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SVBUTM (NDIM, N, M, MP3, C, NO, NV, U, V, MCHEPS, IERR)
C
C****************************** START OF SVBUTM ************************
C
C
      INTEGER NDIM, N, M, MP3, NU, NV, IERR
      DOUBLE PRECISION C(NDIM,MP3), U(NDIM,NU), V(NDIM,NV), MCHEPS
C
C     ------------------------------------------------------------------
C
C     CALCULATE THE SINGULAR VALUE DECOMPOSITION OF A BANDED UPPER
C                          TRIANGULAR MATRIX
C
C     YRITTEN BY:        M. L. OVERTON
C                        COMPUTER SCIENCE DEPARTMENT
C                        STANFORD UNIVERSITY
C                        JANUARY 1976
C     LAST UPDATE:       JANUARY 1976
C
C     ------------------------------------------------------------------
C
C     THIS ROUTINE COMPUTES THE SINGULAR VALUE DECOMPOSITION OF A REAL
C     N*N MATRIX A, I. E. IT COMPUTES MATRICES U, S AND V SUCH THAT
C
C                A = U * S * VT ,
C     WHERE
C          U IS AN N*N MATRIX AND UT * U = I, (UT = TRANSPOSE
C                                                 OF U),
C          V IS AN N*N MATRIX AND VT * V = I, (VT = TRANSPOSE
C                                                 OF V),
C        AND   S IS AN N*N DIAGONAL MATRIX.
C
C     THE CALCULATION IS PERFORMED IN TWO STEPS:
C
C       1. REDUCE THE BANDED UPPER TRIANGULAR MATRIX TO AN UPPER
C          BIDIAGONAL MATRIX USING GIVENS TRANSFORMATIONS.  THIS IS
C          DONE BY SUBROUTINE BIBAND.
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 SYMP.
C          IN APPLIED MATH., VOL.XV, EXPERIMENTAL ARITH., HIGH SPEED
C          COMPUTING, AND MATH. (1963). FOR FURTHER DETAILS SEE
C          COMMENTS AT BEGINNING OF THE SUBROUTINE.
C
C       2. DIAGONALIZE THE BIDIAGONAL MATRIX TO OBTAIN THE SINGULAR
C          VALUES.  THIS IS DONE BY SUBROUTINE SVDBI.
C                THE METHOD USED IS A VARIANT OF THE QR ALGORITHM,
C          DESCRIBED IN: GOLUB AND REINSCH,SINGULAR VALUE DECOMPOSITION
C          AND LEAST SQUARES SOLUTION, NUMER. MATH. 14, 403-420(1970),
C          SECTION 1.3.
C
C     ------------------------------------------------------------------
C
C     THE ROUTINE IS IN DOUBLE PRECISION
C
C     ------------------------------------------------------------------
C
C     THE SPEED OF THIS ROUTINE COULD BE IMPROVED BY IMPLEMENTING
C
```

```
C     PAST GIVENS TRANSPORMATIOUS
C
C
C     ------------------------------------------------------------------
C     ADDITIONAL SUBROUTINES REQUIRED:  BIBAUD, WITH ROTROW AND ROTCOL
C
C                                       SVDBI,  WITH DROTAT
C
C     ------------------------------------------------------------------
C
C     THE FORMAL PARAMETERS ABE:
C
C      NDIM - TEB QUANTITY USED TO DECLARE THE FIRST DIMENSION OF TEE
C             ARRAYS  C,U,V (NDIM .GE.N)
C
C      N    - TEE ORDER OF THE BAUDBD UPPER TRIANGULAR MATRIX A
C
C      M    - THE UUMBBR OF SUPERDIAGONALS IN TEB MATRIX A:
C                  A(I,J) = 0 FOR J  .GT.  I+M AID J  .LT.  I
C
C      MP3  - THE NUMBER OF COLUMUS IN TRB ARRAY C.  MUST BB SET TO M+3.
C
C      C    - AU NDIM * MP3 ARRAY WHICH HOLDS THE NONZERO ELEMENTS OF
C             OF A.
C             THE DIAGONAL IS STORBD IN TEE FIRST N BLBMBNTS OF
C             COLUMU 2, TEE NEXT SUPBRDIAGOUAL IN THB FIRST U-1
C             ELEMENTS OF COLUMU 3, AUD SO ON UP TO TEB LAST
C             NONZERO SUPERDIAGOUAL BEING STORED IN THB FIRST U-M
C             ELEMENTS OF COLUMU M+2.  COLUMUS 1 AUD M+3 ARE ARBITRARY.
C             THUS:
C                     A(I,J) =C(I,J-I+2),  I .LE. J .LB. I+M.
C             TEIB ROUTINE RETURNS TEE DIAGONAL OF TEE MATRIX S,
C             I. E. THE SINGULAR VALUES OF A, IN DESCENDING
C             ORDER, IN COLUMN 2 OF C - THUS THE
C             SINGULAR VALUES WILL BE:
C                   C(1,2)  .GE.  C(2,2)  .GE.  . ⌧  .GE.  C(N,2)
C
C     NU, NV- INTEGER VARIABLES.  TEE UUMBBR OF COLUMUS IN TEE
C             ARRAYS U AUD V.  SET NU TO N IP TEE MATRIX U IS DESIRED,
C             OR SET NU TO 1 I? U IS UOT DESIRED. SET NV TO N
C             IF THE MATRIX V IS DBSIRBD, OR SBT NV TO 1 I? V
C             IS ROT DESIRED,
C
C     U     - REAL NDIM * NU ARRAY.  IF NU = N, TEB MATRIX U IS COMPUTED
C             AND STORED IN THE ARRAY U.
C
C     V     - REAL NDIM *NV ARRAY.  IF NV = N, THE MATRIX V IS COMPUTED
C             AND STORED INTHE ARRAY V.
C
C     IERR  - ERROR PLA6.  TEE ERROR CODES RETURNED HAVE THE FOLLOWING
C             MEANINGS:
C                 IBRR = 0:  NORMAL RETURN
C                 IERR = 2:  ERROR - MP3 DOBS UOT BQUAL M+3.
C                 IERR = 3:  ERROR - MU IS UOT SET TO N OR 1.
C                 IERR = 4:  ERROR - NV IS UOT SET TO N OR 1.
C                 IERR = 5:  ERROR - N IS GREATER THAN NDIM.
C
C     ------------------------------------------------------------------
C
C
      LOGICAL WITHU, WITHV
```

```
      INTEGER I,MM1,MMI
C
C     CHECK INPUT PARAMETERS
      IERR = 0
      IF (MP3.NE.M+3) 60 TO 102
      IF (MU.NE.1.AND. MU.NE.M) GO TO 103
      IF (UU.BQ.l) WITHU = .FALSE.
      IF (MU.EQ.M) WITHU = .TRUE.
      IF (MV.NE.1 .AND. MV.NE.M) 60 TO 104
      IF (MV.EQ.1) WITHV = .FALSE.
      IF (MV.EQ.M) UITEV = .TRUE.
      IF (M.GT.MDIM)60 TO  105
C
C     TURN OFF UNDERFLOW
C      CALL ERRSET(208,256,-1,1,1,0)
C
C     BIDIAGOUALIZB
      CALL BIBAND (MDIM,M,M,MP3,C,MU,MV,WITHU,WITHV,U,V)
C
C     THE SUPERDIAGONAL COLUMN MUST BE SHIFTED DOWN ORE ELEMENT IN C
C     BBPORE CALLING SUBROUTINE SVDBI
      MM1=M-1
      DO 20 I=1,MM1
            MMI = M-I
   20       C(MMI+1,3) = C(MMI,3)
      C (1,3)= O.D O
C
C     DIAGOUALIZB
      CALL SVDBI(MDIM,M,C (1,2),C(1,3),MU,MV,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
```

```
      SUBROUTINE BIBAND(NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V)
C
C****************************** START OF BIBAND ************************
C
C
      INTEGER NDIM, N, M, MP3, NU, NV
      LOGICAL WITHU, WITHV
      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. OVERTON
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                    A = U * J * VT ,
C     WHERE
C          U IS AN N*N MATRIX AND UT * U = I, (UT = TRANSPOSE
C                                              OF U),
C          V IS AN N*N MATRIX AND VT * V = I, (VT = 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 SYMP.
C     IN APPLIED MATH., VOL. XV, EXPERIMENTAL ARITH., 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
C       C     - AN NDIM * HP3 ARRAY WHICH HCLDS THE NCNZERO ELEMENTS OF
C               OF A.
C               TAB DIAGONAL IS STORED IN TRE FIRST NELEMENTS OF
C               COLUHN 2, THE NEXT SUPERDIAGCNAL IN THE PIRST N-l
C               ELEMENTS CF COLUHN 3, AND SO CNUF TO THE LAST
C               NCNZERO SUPERDIAGONAL BBIN6 STORED INTHE FIRST N-H
C               BLBHBNTS OF COLUHN M+2.  CCLUMNS 1 AND M+3 ARE ARBITRARY.
C               THUS:
C                       A(I,J)=C(I,J-I+2),  I .LE. J .LE.I+M.
C               THE ROUTINE RETURNS TEE BIDIAGCNAI HATRIX J WITH THE
C               DIAGONAL IN THE FIRST N ELEMENTS OF CCLUMN 2 OF C AND
C               THE SUPERDIAGONAL IN TRB FIRST N-l ELBWENTS CF
C               COLUMN 3 OF C.
C
C       NU,NV- INTEGER VARIABLES.  THE NUMBER OF COLUMNS IN THE
C               ARRAYS U AND V.  SBT NU TO N IF UITHU = .TRUE., OR SET
C               NU TO 1 CTHERWISE.  SIHILARLY SET NW TC N OR 1.
C
C       WITHU,WITHV - LOGICAL VARIABLES.  IF UITHU = .TRUE., THEN
C               THE HATRIX U IS COHPUTBD AND STORED IN THE ARRAY U.
C               IP WITHV = .TRUE., THEN THE HATRIXV IS COHPUTED  AND
C               STORED IN THE ARRAY V.
C
C       U     - REAL NDIH * NU ARRAY.
C
C       V     - REAL NDIH * NV ARRAY.
C
C     ------------------------------------------------------------------
C
      INTEGER NM2,I,J,K,J0,JOFF,KK
C
C
C     INITIALIZE U,V
      IV (.NOT. WITHU) GO TO 81
      DC 80 I=l,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)=l.DO
  100       CONTINUB
  101 CCNTINUB
C
C     HANDLB DBGBNBBATB CASE
      IV (M.LT.2.OR.N.LT.3) BETURN
      NM2=N-2
C
C     ZEBO WORKING SPACE ON LEFT AND RIGHT SIDES OF C
      DC  120 I=1,N
            C(I,1)=0.D0
            C(I,MP3)=0.D0
  120       CONTINUE
C
C     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 PIRST ANNIHILATES THE CHOSEN ELEMENT
C                 BY A COLUMN ROTATION UITH JOFF=K.  THIS CREATES A NEW
C                 BLBHBNT TO BE ZEROED BY A BOW BOTATION 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.
                   J0=I+K
                   JOFF=K
                   IF (J0.GT.N) GO TO 201
                   DO 200 J=J0,N,M
C                       ROTATE COLUMNS TO ANNIHILATE ELEMENT
                         CALL ROTCOL(NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V
     2                     ,J,JOFF)
                         JOFF=M+1
C                       ELEMENT CREATED BELOW DIAGONAL - ZERO IT AND
C                       CREATE ANOTNER ABOVE EY ROTATING ROWS
                         CALL ROTROW(NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V
     2                     ,J)
  200                    CONTINUE
  201                CONTINUE
  300                CONTINUE
  400          CONTINUE
       FETUFN
       END
```

```
      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-l 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
      INTEGER NDIM,N,M,MP3,NU,NV,I
      DCUBLE PRECISION C(NDIM,MP3),U(NDIM,NU),V(NDIM,NV)
      LCGICAL WITHU,WITHV
      INTEGER K,MP1
      DCUBLE PRECISION X,Y,Z,COST,SINT,TEMP,S,DABS,DSQRT
C
      X=C(I-1,2)
      Y=C(I,1)
C     IF Y IS ZERO THEN THERE IS NOTHING TC DO
      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
      S=DABS(X)+DABS(Y)
      CCST=X/S
      SINT=Y/S
      Z=DSQRT(COST*COST+SINT*SINT)
      CCST=COST/Z
      SINT=SINT/Z
      C(I-1,2)=Z*S
      C(I,1)=0.D0
      MP1=M+1
      DO 100 K=1,MP1
            IF(I-1+K.GT.N)GO TO 100
            TEMP=C(I-1,K+2)
            C(I-1,K+2)=COST*TEMP + SINT *C(I,K+1)
            C(I,K+1)=-SINT*TEMP + COST*C(I,K+1)
  100       CCNTINUE
C
C     UPLATE U (ACCUMULATE TRANSFORMATIONS) - BUST UPCATE U ON THE
C     RIGHT BECAUSE U IS WANTED, NOT U TRANSFOSEC
      IF (.NOT. UITHU) RETURN
      DC 200 K=1,N
            TEMP=U(K,I-1)
            U(K,I-1) =COST*TEMP + SINT*U(K,I)
            U(K,I)=-SINT*TEMP + COST*U(K,I)
  200       CONTINUE
      RETUFN
      END
C
```

```
      SUERCUTINE ROTCOL (NDIM,N,M,MP3,C,NU,NV,WITHU,WITHV,U,V,J,JOFF)
C     APPLY TO MATRIX A CN 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
      INTEGER NDIM,N,M,MP3,NU,NV,J,JOFF
      DOUBLE PRECISION C(NDIM,MP3),U(NDIM,NU),V(NDIM,NV)
      LCGICAL WITHU,WITHV
      INTEGER I,IPK,K,JMIP1,JMIP2,JK1,JK2
      DCUBLB PRECISION X,Y,Z,COST,SINT,TEMP,S,DABS,DSQRT
C
      I=J-JOFF
      JMIP1=J-I+1
      JMIP2=J-I+2
      X=C(I,JMIP1)
      Y=C(I,JMIP2)
C     IF Y IS ZERO THERE IS NOTHING TO CO
      IF(Y.EQ.0.D0) RETURN
C     PERFORM Z=SQRT(X*X+Y*Y); COSTT=X/Z; SINT=Y/Z UITH SCALING TO
C     PREVENT UNDERFLOW
      S=DABS(X)+DABS(Y)
      CCST=X/S
      SINT=Y/S
      Z=DSQRT(COST*COST+SINT*SINT)
      CCST=COST/Z
      SINT=SINT/Z
      C(I,JMIP1)=Z*S
      C(I,JMIP2)=0.D0
      DO 100 K=1,JOFF
            JK1=JMIP1-K
            JK2=JMIP2-K
            IPK = I+K
            TEMP=C(IPK,JK1)
            C(IPK,JK1)=COST*TEMP + SINT*C(IPK,JK2)
            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
      IP (.NOT. WITHV) RETURN
      DO 200 K=1,N
            TBHP=V(K,J-1)
            V(K,J-1)=COST*TEMP + SINT*V(K,J)
            V(K,J)=-SINT*TEMP + COST*V(K,J)
  200 CCNTINUE
      RETURN
C
C******************************** END OF BIBAND ****************************
      END
```

```
      SUBROUTINE SVDEI (NDIM, N, S, T, NU, NV, WITHU, UITHV, U, V, ETA )
C
C*********************************** START OF SVDBI ***************************
C
C
      INTEGER           NDIM, N, NU, NV
      DOUBLE PRECISION S(N), T(N), U(NDIM,NU), V(NDIM,NV), ETA
      LOGICAL           WITHU, WITHV
C
C     ------------------------------------------------------------------
C
C     THIS IS ESSENTIALLY THE SECOND HALF OF SUBROUTINE DSVD,
C     A SINGULAR VALUE DECOMPOSITION ROUTINE IN THE CSD LIBRARY.
C
C     THE ROUTINE IS IN DOUBLE PRECISION.
C
C     CSVD ORIGINAL PROGRAMMER: R. C. SINGLETON
C     DSVD 360 VERSION BY:      B. G. LEWIS
C     DSVD LAST REVISION:       JANUARY 1974
C     SVDBI EXTRACTED BY:       H. L. OVERTON
C     SVDBI EXTRACTED ON:       AUGUST 1975
C     SVDBI LAST REVISION:      SEPTEMBER 1975
C
C     ------------------------------------------------------------------
C
C     ADDITIONAL SUBROUTINE NEEDED:  DPYTHA
C
C     ------------------------------------------------------------------
C
C
C     THIS SUBROUTINE COMPUTES THE SINGULAR VALUE DECOMPOSITION
C     OF A REAL BIDIAGONAL N*N MATRIX B, I.E. IT COMPUTES MATRICES
C     P, S AND Q SUCH THAT
C
C                  J = P * S * QT ,
C     WHERE
C               P IS AN N*N MATRIX AND PT * P = I, (PT = TRANSPOSE
C                                                  OF P) .
C               Q IS AN N*N MATRIX AND QT * Q = I, (QT = TRANSPOSE
C                                                  OF Q) .
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 B.
C         ON RETURN 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         SIMILARIY SET NV TC N OR 1.
C
C     UITHU, WITHV = LOGICAL VARIABLBS.  IF WITHU = .TRUE., THEN
C         THE MATRIX U SUPPLIED IN THE ARRAY U IS POSTMULTIPLIED
C         BY THE MATRIX P.
C         IF WITHV = .TRUE., TRBN THE MATRIX V SUPPLIED IN TEE
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 SUEROUTINE
C     BY BUSINGER AND GOLUB, ALGORITHM 358:  SINGULAR VALUE
C     DECOMPOSITION OF A CCMPLEX MATRIX, COMM. ACM, V. 12,
C     NO. 10, PP. 564 - 365 (OCT. 1969).
C     WITH REVISIONS BY RC SINGLETON, HAY 1972.
C     ------------------------------------------------------------
C
      DOUBLE PRECISION B,W, CS, SN,  F,  X, BPS,  6,  Y
      DCUBLE PRECISION H, Q
      DOUBIE PRECISION DSQRT, DABS, DMAX1
      INTEGER          I, J, K, L, L1
C
C
      S(1) =0.DO
C
C
C    THIS CALCULATION OF EPS IS TAKEN FROH THE RIDDLE OF THE FIRST HALF
-C    OF DSVD
C
      EPS = 0.DO
      DO 50 K=1,N
   50 EFS = DMAX1(EPS,DABS(S(K)) + DABS(T(K)))
C     TCLBEANCB FOR NEGLIGIBLE ELEMENTS
  100 BPS = BPS * ETA
C
C    THE REST OF THE PROGRAM IS TEE SECOND HALF OF DSVD
C
C     QR DIAGONALIZATICN
      K = N
C
C     TEST FOR SPLIT
  230    L = K
  240       I? (DABS(T(t)) .LE. EPS) GOTO 290
            L = L - 1
            IF (DABS(S(L)) .GT. BPS) GOTO 240
C
C     CANCELLATION
         CS = 0.ODO
         SB = 1.0DO
         L1 = L
         L = L + 1
         DO 280 I =L,K
            F = SN * T(1)
            T(I) = ♦ • T(1)
            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 CONVERGENCE
  290  W = S(K)
         IF (L .EQ. K) GOTO 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     QR STEP
         CS = 1.0D0
         SN = 1.0D0
         L1 = L + 1
         DO 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
            IF (WITHU) CALL DROTAT (U(1,I-1), U(1,I), CS, SN, N)
  350       CONTINUE
C
         T (L) = 0.0D0
         T(K) = F
       - S (K) = X
         GOTO 230
C
C     CONVERGENCE
  360    IF(W.GE.0.0D0) GOTO 380
         S(K) = -W
         IF (.NOT.WITHV) GOTO 380
         DO 370 J = 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 = -1.0D0
         DO 390 I = K,N
            IF (S(I) .LT. G) GOTO 390
            G = S(I)
            J = I
  390       CONTINUE
         IF (J .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)
  400       V(I,K) = Q
  410    IF (.NOT.WITHU) GOTO 430
         DO 420 I = 1,N
            Q = U(I,J)
            U(I,J) = U(I,K)
  420       U(I,K) = Q
  430    CONTINUE
  450    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE DROTAT (X, Y, CS, SN, N)
      INTEGER           N
      DOUBLE PRECISICN  CS,  SN,  X(N), Y(N)
C
C
      DOUBLE PRECISICN XX
      INTEGER          J
C
C
      DC 10 J = l,N
         XX = X(J)
         X(J) = XX*CS + Y(J)*SN
   10    Y(J) = Y(J)*CS - XX*SN
         RETURN
C
C*********************************** END OF SVIBI ****************************
C
C*********************************** END OF SVBUTM ***************************
C
      E ND
```

```
C      +----------------------+
C      + SAMPLE MAIN PROGRAM +
C      +----------------------+
C
C      PEPPEP LENGTHS OF MATPICES :
C
C         D (Q),X(M*Q),Y(N*Q)
C
       DOUBLE PRECISION D(20),X(8000),Y(2000),EPS
       INTEGER I,IECODE,IORTHG,H,MMAX,M,N,G,PINIT,Q
       EXTERNAL AX
C
       COMMON A(3000),IINDEX(3000),JINDEX(3000),NDATA
       LOUBLE PRECISIGN A
       INTEGER IINDEX,JINDEX,NDATA
       INTEGER K,KP1,KP5,KPJ,NCARD,NDATA
C
C      IOUT IS OUTPUT UNIT NUMBER
C      MCHEPS IS MACHINE PRECISION
C
       INTEGER LOUT
       DOUBLE PRECISION MCHEPS
       DATA LOUT/6/
       DATA MCHEPS/2.22D-16/
C
C      NCARD IS NUMBER OF DATA CARDS TO BE READ
C
       READ (5,5010) M,N,NCARD
 5010  FORMAT(3I5)
C
       K = 0
       DO 10 I = 1,NCARD
          KP1 = K+1
          KP5 = K+5
          READ (5,5020) (IINDEX (L) ,JINDEX (L),A(L),L=KP1,KP5)
 5020     FORMAT(5(2I3,F10.6))
          K = K+5
    10 CONTINUE
C
C      NDATA IS NUMBER OF NON-ZERO ELEMENTS IN A
C      IINDEX = 0 SIGNIFIES END OF DATA INPUT
C
       NDATA = K
       K = K-5
       DO 15 J = 1,5
          KBJ = K+J
          IF (IINDEX(KPJ).GT.0) GO TO 15
          NDATA = KFJ-1
          GO TO 17
    15 CONTINUE
C
    17 CONTINUE
       Q = 10
       EINIT = 2
       G = 9
       MMAX = 2000
       EPS = 1.D-3
       M = 0
       IORTHG = 0
```

```
C
      WRITE(LOUT,6010) M,N,Q,PINIT,G,MMAX,EPS,H,IORTHG
 6010 FORMAT(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 =,1PD10.3,5X,4H H =,I4,5X,9H IORTHG =,I4)
C
      CALL MAXVAL(M,N,Q,PINIT,G,MMAX,EPS,AX,H,D,X,Y,IORTHG,
     1            LOUT,RCHEPS,IECODE)
C
      WRITE(LOUT,6020)
 6020 FORMAT (35H *****  USING BLOCK LANCZOS  ***** )
      WRITE(LOUT,6030) H,IECODE
 6030  FORMAT(8H ** H =,I4,13H **  IECODE =,I4)
      IF (H.EQ.0) STOP
      WRITE(LOUT,6040) (D(I),I=1,H)
 6040 FORMAT(20H SINGULAR VALUES . ../5H **  ,6(1PD25.15/5X))
      STOP
      END
```

```
      SUBROUTINE AX(M,N,P,U,V,ORIG)
      INTEGER M,N,P
      DOUBLE PRECISION U(M,P),V(N,P)
      LOGICAL ORIG
C
C        AX COMPUTES X = A*Y IF ORIG IS TRUE, AND Y = A'*X
C     IF ORIG IS FALSE.  X IS STORED IN U AND Y IS STORED IN V.
C
      COMMON A(3000),IINDEX(3000),JINDEX(3000),NDATA
      DOUBLE PRECISION A
      INTEGER IINDEX,JINDEX,NDATA
C
      INTEGER I,J,K,L
C
      IF (.NOT.ORIG) GO TO 100
C
C     COMPUTE X = A*Y
C
      DO 20 K = 1,P
C
         DO 10 L = 1,M
            U(L,K) = 0.D0
   10    CONTINUE
C
   20 CONTINUE
C
      DO 40 L = 1,NDATA
         I = IINDEX(L)
         J = JINDEX(L)
C
         DO 30 K = 1,P
            U(I,K) = U(I,K) + A(L)*V(J,K)
   30    CONTINUE
C
   40 CONTINUE
C
      RETURN
C
C     COMPUTE Y = A'*X
C
  100 CONTINUE
C
      DO 120 K = 1,P
C
         DO 110 L = 1,N
            V(L,K) = 0.D0
  110    CONTINUE
C
  120 CONTINUE
C
      DO 140 L = 1,NDATA
         I = IINDEX(L)
         J = JINDEX(L)
C
         DO 130 K = 1,P
            V(J,K) = V(J,K) + A(L)*U(I,K)
  130    CONTINUE
C
  140 CONTINUE
```

```
      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.