1971

Reference manual to fritz laboratory matrix package (flmxpk), June 1971

C. N. Kostem

S. N. Iyengar

FLMXPK - A MATRIX PACKAGE

by
Sampath N.S. Iyengar
Celal N. Kostem

Fritz Engineering Laboratory
Department of Civil Engineering
Lehigh University
Bethlehem, Pennsylvania

September 1971

Fritz Engineering Laboratory Report No.400.4

TABLE OF CONTENTS

1. INTRODUCTION

1.1 A Brief History

The matrix package MXPAK was used in course work and research by members of the Fritz Laboratory until the GE 225 computer at Lehigh University was replaced by the CDC 6400 in the Summer of 1968. MXPAK was written in LEWIZ, a language developed at Lehigh, and was compatible for use with the GE 225 but not the CDC 6400.

To meet the immediate needs of the Laboratory, FCMXPK (FORTRAN Callable Matrix Package) was written by Mr. Edward T. Manning, Jr. and Mr. Iyengar in Fall 1968. Later, a documentation of this package was considered worthwhile in view of its highly satisfactory performance. At this stage, it became evident that many improvements and some additions were possible. The result of all such modifications is the present version, FLMXPK (Fritz Laboratory Matrix Package), which is described in the succeeding pages.

1.2 Divisions in the Text

FLMXPK contains 30 routines to perform matrix operations, as summarized in the following table.

Operation	Subprogram
Add matrices	ADD
Determinant of matrix	DETMT,MINV,SINV,SOLVE
Create diagonal matrix	DIAG
Eigenvalues of symmetric matrices	EV,IEV,GEVP
Invert matrix	MINV,SINV
Copy matrix	MOVE
Multiply matrices	MULT,PMULT,POSTM,SCMUL, TMULT,XABATC,XABTA, XABTC,XATBAC,XATBB
Print matrix	OUTE,OUTF,OUTG
Read matrix	RDCBC,RDCOLG,RDRBR, RDROWG
Simultaneous equations	SOLVE
Transpose matrix	SQTR,TRANS
Subtract matrices	SUB

Each of these routines has received individual treatment to the extent considered necessary (Chapter 2). In general, the discussion is under the following headings:

1. Function
2. Development of the Subprogram (including the mathematical background)
3. Special Features (if any)
4. Limitations (if any)
5. Additional Remarks (if any)

The limitations listed under each routine are in addition to the general limitations which are discussed in Section 4.2 of the User's Guide (Chapter 4).

Chapter 3 - "General Notes" - contains remarks which apply to the package as a whole.

The User's Guide (Chapter 4) explains briefly how each routine may actually be used. The more important limitations

of each routine are listed here also. The ready reference sheet (Chapter 5) is hopefully all that a user needs in day-to-day work after he has gained some experience in the use of this package.

Proofs of certain theorems in matrix algebra and topics related to the text are included in Appendix 1 to make the subject matter as complete in content as possible.

Finally, the subprograms themselves are listed in Appendix 2.

2. ROUTINES

2.1 SUBROUTINE ADD (A,B,C,M,N)

2.1.1 Function

The sum of matrices A and B is made available to the calling program as matrix C. C = A + B.

Matrices A,B, and C are each of size M rows by N columns.

2.1.2 Development of the Subprogram

In algebraic terms, the addition of two matrices may be expressed by $C_{I,J} = A_{I,J} + B_{I,J}$. From this definition, the following subprogram is easily written.

```
      SUBROUTINE ADD(A,B,C,M,N)
      COMMON/IYENGAR/I,J,Y(12)
      REAL A(M,N),B(M,N),C(M,N)
      DO 1000 I = 1,M
      DO 1000 J = 1,N
 1000 C(I,J)=A(I,J)+B(I,J)
      RETURN
      END
```

The use of the labeled COMMON block is discussed under "General Notes" in Chapter 3.

In the above routine, the computations required for locating the elements in the three double-subscripted arrays are time-consuming. The matrices may, however, be treated as single-subscripted arrays in the subprogram to reduce the time required for address computations.

A further advantage of this procedure is that the

operations can now be performed using a single DO-loop as indicated in the listing (Appendix 2).

The size of each matrix is M*N elements but a declaration of the type REAL A(M*N) is invalid. This difficulty is easily overcome by dimensioning each array as a vector consisting of one element only. When the subprogram is CALL-ed, the starting addresses of each array are passed to the subprogram and hence, in the subprogram, the arrays are spaced as required in spite of the arbitrary dimensioning.

2.1.3 Additional Remarks

The resultant matrix C can be stored in either of the original matrices (say A). In this case, the original matrix (A) will be destroyed.

2.2 SUBROUTINE DETMT(A,DA,N)

2.2.1 Function

The determinant of the given matrix A of size N rows by N columns is made available to the calling program as DA.

2.2.2 Development of the Subprogram

The basis of operations is to reduce the given matrix to a lower or upper triangular matrix by elementary transformations and form the product of the diagonal elements of the reduced matrix to give the determinant. In the process of such a reduction, whenever a submatrix in the upper left of the original matrix is singular, division by zero will be encountered. Row or column exchanges are necessary to overcome such a situation, if it exists, before each step of the reduction process.

This subprogram utilizes column exchanges and reduces the original matrix in its own space to an upper triangular matrix.

Assume that the matrix has the following form at the end of I-1 (I > 1) stages of the reduction process.

$$\begin{bmatrix}
A_{1,1} & A_{1,2} & \cdots & & & & A_{1,N} \\
0 & A_{2,2} & \cdots & & & & A_{2,N} \\
0 & 0 & \ddots & & & & \vdots \\
0 & 0 & \ddots & & & & \vdots \\
0 & 0 & & A_{I,I} & A_{I,I+1} & \cdots & A_{I,N} \\
0 & 0 & & A_{I+1,I} & A_{I+1,I+1} & \cdots & A_{I+1,N} \\
\vdots & \vdots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & \vdots & & \vdots \\
\vdots & \vdots & & \vdots & \vdots & & \vdots \\
0 & 0 & & A_{N,I} & A_{N,I+1} & \cdots & A_{N,N}
\end{bmatrix}$$

Except possibly for the elements in row 1, all the elements are modified by the reduction process and, therefore, do not correspond, in general, to those of the original matrix. The part product of the diagonal elements $A_{K,K}$ where K ranges from 1 through I-1 is assumed to have been computed and stored in DA.

Further processing starts with row I. Since $A_{I,I}$ may be zero, the immediate step is to search the Ith row for a non-zero element in columns I through N. If $A_{I,I}$ itself is non-zero, the process of reduction may be continued. However, it is advantageous, in the interest of accuracy, to choose $A_{I,I}$ such that it is as large as possible in absolute value, since division by $A_{I,I}$ is later involved. Hence, the Ith row is searched for the largest absolute valued element in columns I through N.

If all the elements searched are zero, the conclusion is that the matrix is singular, and hence DA has the value zero. Control is returned to the calling program in such a case.

Otherwise, the largest absolute valued element in row I is located in, say, column M where $N \geq M \geq I$. If M equals I, the largest absolute valued element is $A_{I,I}$ itself and no exchange of columns is involved. If M does not equal I, columns M and I are exchanged, and the corresponding change in the sign of the determinant is accounted for. It is useful to note that such an exchange may be limited to elements in rows I through N only, as elements in rows 1 through I - 1 do not affect the evaluation of the determinant.

For further discussion, the form of the matrix noted above remains valid, although the values of some of the elements may have changed because of column exchanges.

The part product of the diagonal elements is then modified to include $A_{I,I}$. In FORTRAN language, DA = DA*A(I,I).

The next step is the process of reduction. To reduce the elements in column I of rows I + 1 through N to zero, it is necessary to add $\frac{-A_{M,I}}{A_{I,I}}$ times the Ith row to row M, where $(I + 1) \leq M \leq N$. After such additions, the modified matrix has the following form.

$$
\begin{bmatrix}
A_{1,1} & A_{1,2} & \cdot & \cdot & \cdot & & & & & A_{1,N} \\
0 & A_{2,2} & \cdot & \cdot & \cdot & & & & & A_{2,N} \\
\cdot & 0 & \cdot & \cdot & \cdot & & & & & \cdot \\
\cdot & \cdot & 0 & \cdot & \cdot & & & & & \cdot \\
\cdot & \cdot & \cdot & 0 & \cdot & & & & & \cdot \\
\cdot & \cdot & \cdot & \cdot & 0 & A_{I,I} & \text{(Elements in this row need not be modified)} & & & \\
0 & 0 & 0 & 0 & 0 & 0 & A_{I+1,I+1} - \frac{A_{I+1,I}}{A_{I,I}} A_{I,I+1} & \cdot \cdot \cdot & & A_{I+1,N} - \frac{A_{I+1,I}}{A_{I,I}} A_{I,N} \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
0 & 0 & 0 & 0 & 0 & 0 & A_{M,I+1} - \frac{A_{M,I}}{A_{I,I}} A_{I,I+1} & \cdot \cdot \cdot & & A_{M,N} - \frac{A_{M,I}}{A_{I,I}} A_{I,N} \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 0 & \cdot & & & \cdot \\
0 & 0 & 0 & 0 & 0 & 0 & A_{N,I+1} - \frac{A_{N,I}}{A_{I,I}} A_{I,I+1} & \cdot \cdot \cdot & & A_{N,N} - \frac{A_{N,I}}{A_{I,I}} A_{I,N}
\end{bmatrix}
$$

Since the purpose here is to evaluate only the determinant, better speed can be achieved by avoiding generation in the machine of fresh values for some of the elements. Some of these values are known (zeroes below $A_{I,I}$ in column I) and others do not affect the final product (elements to the right of $A_{I,I}$ in row I).

The general formula, then, in the region of interest, is simply

$$A'_{M,J} = A_{M,J} - \frac{A_{M,I}}{A_{I,I}} A_{I,J} ,$$

where both M and J range in values from I + 1 through N, and the prime* denotes the fresh values computed for the corresponding elements.

The elements are evaluated row by row and $\frac{A_{M,I}}{A_{I,I}}$ is recognized as a constant when elements in row M are processed.

When the search and reduction process explained above is repeated for a total of N-1 cycles, the matrix is (upper) triangulated and all that remains is to modify the part product of the diagonal elements obtained thus far by multiplying it by $A_{N,N}$ to give the final value of the determinant DA.

*In the subprogram, it suffices to redefine $A_{M,J}$ as the quantity on the right-hand side of the equation.

A special situation arises if N is 1. The normal reduction process in this case would imply operations in stages 1 through 0, which is meaningless. However, the determinant is simply $A_{1,1}$ of the original matrix (1 by 1). Hence, very soon after the subroutine is entered, N is checked for equality with 1 and if the test is true, the determinant is set to $A_{1,1}$ and control is returned to the calling program.

2.2.3 Features

The operations involved are carried out within the space of the given matrix itself. Those operations which do not affect the final result are avoided altogether to ensure better speed of performance.

2.2.4 Limitations

The original matrix A is destroyed in the process of evaluating the determinant.

It is conceivable that the determinant of a matrix which is singular may not be identically zero when this subprogram is used, because of machine errors in using floating-point arithmetic. A check of the diagonal element $A_{I,I}$ for its absolute value at each stage of the reduction process (after column exchanges, if any) with a number like 1.0×10^{-6} could have been employed to recognize the singularity of the matrix, but this has

an obvious disadvantage. The determinant is a function of the values of the elements in the original matrix as well as their positions. Hence, a purely arbitrary number like 1.0×10^{-6} is inadequate for such a test. In most engineering applications, the user has a feel for the value of the determinant based on the trend of a set of calculations and is normally able to recognize a singular matrix when the determinant is "small".

2.2.5 Additional Remarks

If the value of DA is to be printed, it is better to output this quantity in E-format (or G-format) than in F-format since, generally, its magnitude is unknown.

This is the only subprogram (in this package) that can also be written as a FUNCTION subprogram [FUNCTION DA(A,N)]. Were it so written, it would have to be "referenced" instead of being "CALL-ed" by a calling program. In its present form, no exception is necessary to the general rule of CALL-ing any of the subprograms.

2.3 SUBROUTINE DIAG(A,DA,N)

2.3.1 Function

Matrix A of dimension N rows by N columns is generated as follows. All the diagonal elements have the same value DA, and the off-diagonal elements are all zero (Diagonal matrix).

2.3.2 Development of the Subprogram

Treating the matrix as a vector of N*N elements in the subprogram, a null array is created in the first DO-loop (Appendix 2). Next, the diagonal elements are each assigned the value DA in the second DO-loop.

2.3.3 Additional Remarks

DA should be defined as a FORTRAN real number in the calling program.

2.4 SUBROUTINE EV(A,S,N) and ENTRY IEV

2.4.1 Function

Eigenvalues and eigenvectors of the symmetric matrix A are computed.

2.4.2 Development of the Subprogram

A. Theory and a Practical Approach

The procedure presented here is based upon the mathematical discussion of the problem outlined by Greenstadt[1].

If λ is a scalar and X is a (non-zero) column vector such that $AX = \lambda X$, vector X is known as the eigenvector corresponding to the eigenvalue λ. Since matrix A is of size N (N rows by N columns), it has N eigenvalues and N eigenvectors. Since matrix A is also symmetric, its eigenvalues are all real but not necessarily distinct. However, N distinct eigenvectors can be found. In fact, it can be shown that these eigenvectors are also orthogonal; that is, mutually perpendicular to each other.

If the eigenvalues represent the diagonal elements of a diagonal matrix D, and the corresponding eigenvectors (normalized) represent the columns of a matrix S, the equation AS = SD represents a generalization of $AX = \lambda X$. (Recall that postmultiplication of matrix S by a diagonal matrix D is equivalent to multiplication of each column of matrix S by the corresponding diagonal

element of matrix D). Since the X vectors are orthonormal, S is an orthonormal matrix and hence $S^T = S^{-1}$. The equation AS = SD can, therefore, also be written as $S^TAS = D$.

In mathematical terms, if S is an orthonormal matrix such that the transform (S^TAS) of matrix A is a diagonal matrix, the diagonal elements of the diagonal matrix are the eigenvalues of matrix A and the columns of matrix S are the corresponding eigenvectors. The requirement is, therefore, to find the matrix S. In the case of a 2 x 2 matrix, it can be found almost directly, as shown below.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{1,2} & A_{2,2} \end{bmatrix}$$

Matrix A is symmetric.

$$\text{Assume } S = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Matrix S is orthonormal, and the value of θ is to be found. The transform of A is given by S^TAS which is symmetric.

$$S^TAS = \begin{bmatrix} A_{1,1}\cos^2\theta + A_{2,2}\sin^2\theta - 2A_{1,2}\cos\theta\sin\theta & (A_{1,1}-A_{2,2})\cos\theta\sin\theta + A_{1,2}(\cos^2\theta-\sin^2\theta) \\ (A_{1,1}-A_{2,2})\cos\theta\sin\theta + A_{1,2}(\cos^2\theta-\sin^2\theta) & A_{1,1}\sin^2\theta + A_{2,2}\cos^2\theta + 2A_{1,2}\sin\theta\cos\theta \end{bmatrix}$$

Hence, A will be diagonalized if θ satisfies the relation

$$(A_{1,1}-A_{2,2})\sin 2\theta = -2A_{1,2}\cos 2\theta$$

$$\text{or} \quad \tan 2\theta = \frac{-2A_{1,2}}{A_{1,1}-A_{2,2}} \qquad (1)$$

A simple application in Civil Engineering is the evaluation of principal stresses in an element which is subject to normal and shearing stresses.

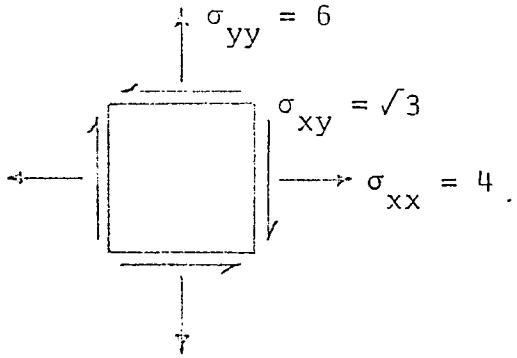


Figure 1(a)

The stress tensor for the loading shown in Figure 1(a) is

$$\begin{bmatrix} 4 & \sqrt{3} \\ \sqrt{3} & 6 \end{bmatrix}$$

From Equation (1), $\tan 2\theta = \frac{-2\sqrt{3}}{4-6} = \sqrt{3}$ or $\theta = 30°$.

Hence, the principal stresses are given by the diagonal elements of the following matrix:

$$\begin{bmatrix} 3.0 & 0.0 \\ 0.0 & 7.0 \end{bmatrix}$$

The orientations of the principal planes are depicted in Figures 1(b), 1(c), and 1(d).

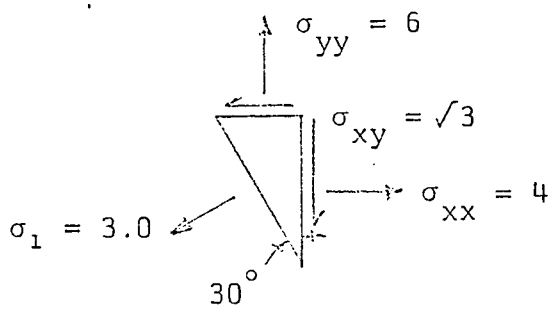


Fig. 1(b)

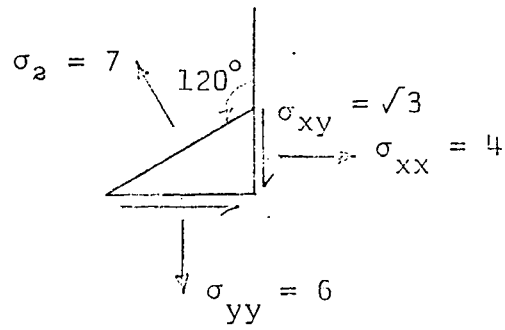


Fig. 1(c)

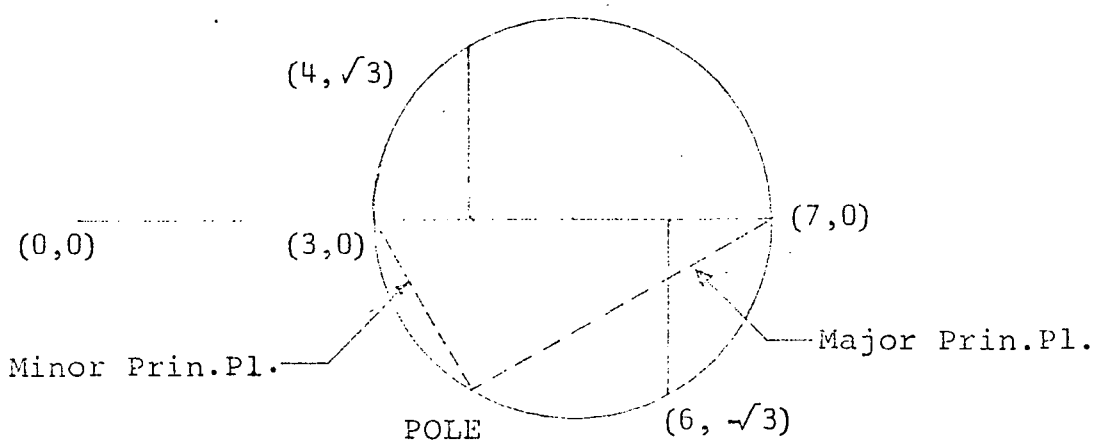


Fig. 1(d)

When matrix A is larger than 2 x 2 it is not possible to find S directly, since there is more than one off-diagonal element to be annihilated. [Whenever an off-diagonal element is referred to below, it is assumed to be in the upper half of the matrix (subscripts P,Q where $P < Q$). By symmetry, there is a corresponding (Q,P) element identical in value in the lower half. Both elements are affected in the same way throughout.]

However, it is possible to annihilate a selected off-diagonal element $A_{P,Q}$ if matrix A is transformed by R, where R is the Identity matrix modified with regard to the following elements only.

$$R_{P,P} = \cos\theta \qquad R_{P,Q} = \sin\theta$$

$$R_{Q,P} = -\sin\theta \qquad R_{Q,Q} = \cos\theta$$

Matrix R is orthonormal and $\tan 2\theta = \dfrac{-A_{P,Q}}{0.5\,(A_{P,P}-A_{Q,Q})}$

(See Appendix 1 for the values of $\sin\theta$ and $\cos\theta$ from this equation). In the transform R^TAR represented by matrix B, $B_{P,Q}$ (and $B_{Q,P}$) will be zero. We now need to continue this process to annihilate some other off-diagonal element of matrix B by a transform using another orthonormal matrix, say T. Let $C = T^TBT = T^TR^TART = (RT)^TA(RT)$. Since R and T are both orthonormal, RT is orthonormal (Appendix 1). In other words, C is the transform of A by RT, although it was shown to be derived in two steps using elementary orthonormal transformation matrices R

and T in turn.

With the transform of B to C, usually it happens that $B_{P,Q}$, which was zero, will not retain that value. The annihilation of $A_{P,Q}$ is therefore undone.

Naturally, therefore, the question arises whether a process such as this is convergent. The answer in the affirmative was first provided by Jacobi. (The proof is not presented here.) To achieve rapid convergence, he also proposed selection of that off-diagonal element which has the largest absolute value, for annihilation at the subsequent step of the process.

The selection of the off-diagonal element may be modified slightly when using the high-speed electronic computer(1). The procedure is as follows.

The square root of the sum of the squares of the off-diagonal elements in the upper half of matrix A is calculated first. (Note that squaring each element emphasizes the importance of the elements having larger absolute values. The principle is analogous to that used in Least Squares Method for Curve Fitting.) This is called the initial threshold (THRI in the subprogram). If diagonalization of A can be continued indefinitely to the point where all off-diagonal elements are zero, the final threshold will be zero. For practical reasons, it is necessary to terminate the program when the absolute value of each

off-diagonal element is smaller than a specified value (THRF in the subprogram). Since the magnitudes of the original off-diagonal elements will differ with each problem, it is further necessary to relate THRF to these elements in some way. This subprogram assumes that THRF is one-millionth of THRI. (Greater accuracy can be achieved as discussed later under ENTRY IEV.)

Having set the value of THRF, the value of THRI is reduced by dividing it by a number M at least equal to N, the size of the matrix. In this subprogram, N itself is the number chosen. Then, there is at least one off-diagonal element which is larger in absolute value than this new value of THRI[1].

Any off-diagonal element which exceeds THRI in absolute value is a "candidate" for annihilation. The choice is made systematically. If the first off-diagonal element ($A_{1,2}$) is a candidate, it is annihilated. (Else, $A_{1,3}$ is considered.)

Assuming $A_{1,2}$ was annihilated in the previous step, the next off-diagonal element considered is $B_{1,3}$. (Note that A is transformed by now to B.) The next in order is $C_{2,3}$, etc. (column by column).

At the conclusion of the first sweep, there is no guarantee that every off-diagonal element will be less than THRI in absolute value, since annihilation of an

element in a particular step may have been undone by succeeding transformations. This is no serious problem since one or more repeat sweeps will assure that THRI is larger than every off-diagonal element in absolute value, by virtue of convergence of the process. The necessity or otherwise of a repeat sweep is indicated by the value of the logical variable IND in the subprogram.

THRI is now further reduced by dividing it by M and the process repeated all over again. When THRI, so modified successively, becomes less than THRF, the diagonalization is terminated.

B. Programming Details

The procedure calls for many matrix multiplications. Although several names have been used in the earlier discussion for the sake of clarity, the subprogram actually uses only matrices A and S. The original matrix A is continuously modified so that on return to the calling program, it is diagonalized and has, for its diagonal elements, the eigenvalues of the original matrix A. Matrix S is initially defined to be the Identity matrix and is also modified at each step so that finally it will store, in its columns, the eigenvectors of the original matrix A.

Assuming calculations at stage J have been completed, the modifications required for matrices A and S at stage J + 1 will now be noted. Let R be the elementary

orthonormal transformation matrix during this stage. For convenience, again, let $B = R^T AR$ and $T = SR$.

Since matrices A and B are symmetric and R contains a number of zero elements, it is reasonable to suspect that regular matrix multiplication is not warranted. A long-hand matrix multiplication shows that the following equations are true. These may, however, also be derived. ($A_{P,Q}$ is the element to be annihilated.)

$$B_{P,K} = A_{P,K} \cos\theta - A_{Q,K} \sin\theta \quad \text{where } K \neq P \text{ or } Q \tag{2}$$

$$B_{Q,K} = A_{P,K} \sin\theta + A_{Q,K} \cos\theta \quad \text{where } K \neq P \text{ or } Q \tag{3}$$

$$B_{I,P} = A_{I,P} \cos\theta - A_{I,Q} \sin\theta \quad \text{where } I \neq P \text{ or } Q \tag{4}$$

$$B_{I,Q} = A_{I,P} \sin\theta + A_{I,Q} \cos\theta \quad \text{where } I \neq P \text{ or } Q \tag{5}$$

$$B_{I,K} = A_{I,K} \quad \text{where I and K are both different from P and Q} \tag{6}$$

$$B_{P,P} = A_{P,P} \cos^2\theta + A_{Q,Q} \sin^2\theta - 2A_{P,Q} \sin\theta\cos\theta \tag{7}$$

$$B_{Q,Q} = A_{P,P} \sin^2\theta + A_{Q,Q} \cos^2\theta + 2A_{P,Q} \sin\theta\cos\theta \tag{8}$$

$$B_{P,Q} = B_{Q,P} = (A_{P,P} - A_{Q,Q}) \cos\theta \sin\theta + A_{P,Q}(\cos^2\theta - \sin^2\theta) \tag{9}$$

$$T_{I,K} = S_{I,K} \quad \text{where } K \neq P \text{ or } Q \tag{10}$$

$$T_{I,P} = S_{I,P} \cos\theta - S_{I,Q} \sin\theta \tag{11}$$

$$T_{I,Q} = S_{I,P} \sin\theta + S_{I,Q} \cos\theta \tag{12}$$

This list of equations enables us to recognize a few features which, when incorporated in the program, speed up operations considerably.

The matrix B is the same as matrix A, except for rows P and Q and columns P and Q. Further, by virtue of symmetry, rows P and Q are the same as columns P and Q, respectively. The following scheme to generate matrix B (the new A) can therefore be adopted.

1. Calculate a new value for each element of column P. Modify the corresponding element in row P to have the same value.

   Do the same for column Q and row Q in the same DO-loop. Equations 4 and 5 are to be used.

2. As a consequence of step 1, we have some fictitious values for (the pivotal) elements $B_{P,P}$, $B_{P,Q}$, $B_{Q,P}$ and $B_{Q,Q}$ as these elements also have been computed using either Equation 4 or Equation 5. However, this "wasteful" set of calculations (the values will be discarded) has saved any checking of subscripts that would otherwise be involved in Step 1. The elements of the pivotal set are now evaluated according to the Equations 7, 8, and 9. The "mistake" is thus corrected.

Turning next to matrix T, it is seen that, again, only columns P and Q of matrix T are different from those of matrix S. The calculations needed can be easily combined with those in Step 1 above.

C. ENTRY IEV

It was remarked under Part A in the above discussion that the accuracy of calculations need not be confined to the degree where THRF equals one-millionth of the initial threshold value. The following explains this feature.

The diagonalization of matrix A was shown to be an iterative process. Since a limit to this process was required to be set for practical reasons, THRF was set to the value stated. The fraction, one-millionth, was arbitrarily chosen assuming that the requirements of the user are thereby met. (Reference 1 uses one-billionth in a numerical example.) If, however, a user is interested in greater accuracy, all that is necessary is to return to the subprogram from the calling program and thus continue the process which is iterative. In other words, the subprogram should be CALL-ed again. However, one snag of this procedure must be avoided.

If the subprogram is CALL-ed again in the form CALL EV(A,S,N), the S-matrix will get reinitialized to the Identity matrix. So, although the eigenvalues will be improved, the eigenvectors will all have wrong values. The ENTRY statement is useful to avoid this difficulty.

The statement ENTRY IEV in the subprogram allows an alternate point of entry (the next executable statement after the ENTRY statement). Hence, if the CALL statement

400.4

is CALL IEV(A,S,N), the initialization of the matrix S will be bypassed. Thus, matrix S also will be improved.

In the subprogram, the positioning of the ENTRY statement is such that calculations for THRI are also bypassed. This was deliberately done for the following reasons.

With the first use of the subprogram, the calculations for the initial threshold will be made and the diagonalization procedure assures us that every off-diagonal element is less than one-millionth of this quantity in absolute value. The variables THRI and THRF in the subprogram, defined prior to the ENTRY statement, represent these quantities. The variable THRI changes in value during the execution of the subprogram and when the diagonalization is terminated, it is less than THRF. If now, THRF is reduced to one-millionth of its own value, we will have set the final threshold for a subsequent use of this subprogram and be ready for the improvement procedure, without having to calculate a new initial threshold. (See the redefinition of THRF just prior to the RETURN Statement.) The accuracy prescribed for the second run would, therefore, be 10^{-12} times the initial threshold value calculated for the original matrix A. This process is also iterative and hence in general terms, the accuracy is 10^{-6n} times the initial threshold value, where n is the number of times this subprogram is CALL-ed.

The advantages are these: 1. The user has a measure of the accuracy based on the number of CALL statements. 2. The calculations for initial threshold are limited to the first set. 3. These calculations can be combined with the initialization of the S matrix (to the Identity matrix) in the same DO-loop nest.

D. CALL-ing Procedure

The following rules apply to the CALL statements for the reasons stated.

1. CALL EV(A,S,N) must be used the first time around. This permits matrix S to be initialized and the initial threshold to be evaluated.

If the accuracy prescribed is considered good enough, the problem is solved on return to the calling program.

2. CALL IEV(A,S,N) must not be used the first time around, since matrix S and the initial threshold will not be properly initialized.

3. CALL IEV(A,S,N) may be used as often as necessary subsequent to CALL EV(A,S,N).

Trail runs have shown that the eigenvalues will not differ appreciably when improvement is attempted. This is because the eigenvalues found in the initial run are already very close to the exact values.

The eigenvectors show some improvement with each subsequent CALL IEV(A,S,N). In engineering problems, one improvement cycle may be considered the maximum necessary.

(A good check on the final results is to form the matrix product SAS^T, where S is the matrix of eigenvectors and A is the matrix of eigenvalues, and compare this with the original matrix A. Since the original matrix A is destroyed when this subprogram is used, it needs to be saved or printed earlier for comparison if this check is deemed necessary.)

A repeated number of improvement cycles indicates fastidiousness with respect to accuracy uncalled for in engineering applications. Further, there is a good chance of creating an underflow in the machine (extremely small quantities in absolute value other than zero itself cannot be handled by the machine).

For obvious reasons, matrices A and S should not be affected in any way in the calling program between two successive CALL-s to this subprogram.

4. CALL EV(A,S,N) must not be used except for the first time. The reason, as already stated, is that such a CALL would destroy the S-matrix previously found by reinitializing it to the Identity matrix.

E. Special Case of 1 x 1 Matrix

If N equals 1, there are no off-diagonal elements. For this case, the eigenvalue is the element itself and the eigenvector is just 1.0. In the subprogram, therefore, N is checked for equality with 1. If N = 1, the relations mentioned are established directly.

2.4.3 Features

Memory requirements are minimal, and speed is ensured by avoiding full-scale matrix multiplications.

An improvement of the eigenvalues and eigenvectors is easily possible, at the user's option, by one or more additional CALL statements.

2.4.4 Limitations

The matrix whose eigenvalues and eigenvectors are to be found must be symmetric. It will be destroyed in the subprogram.

2.5 SUBROUTINE GEVP(A,B,S,T,N)

2.5.1 Function

Eigenvalues and eigenvectors of the symmetric matrix A where $AX = \lambda BX$ are computed (General Eigenvalue Problem). In this equation, X is the eigenvector corresponding to the eigenvalue λ which is a scalar quantity. B is a symmetric matrix which is also positive definite.

2.5.2 Development of the Subprogram

In terms of matrices, the above equation may be rewritten in the form

$$A\,\Phi = B\,\Phi\,L$$

where Φ represents the matrix of eigenvectors; that is,

$$\Phi = \left[X_1\ X_2\ \ldots X_N\right] \quad \text{and}$$

$X_1, X_2, \ldots, X_N$ are the N column vectors corresponding to $\lambda_1, \lambda_2, \ldots, \lambda_N$ which are the N eigenvalues. L is the matrix of eigenvalues and is a diagonal matrix.

At first sight, it would appear that the solution may be obtained directly by writing

$$B^{-1}A\,\Phi = \Phi\,L \quad \text{or} \quad C\,\Phi = \Phi\,L$$

and finding the eigenvalues and eigenvectors of the matrix C, by the use of another subprogram. This would be in order provided the subprogram used can handle non-symmetric matrices. This package does not contain such

a subprogram.

Since B is symmetric, B^{-1} is also symmetric (Appendix 1). The product $C = B^{-1}A$ is, however, not necessarily symmetric, although both B^{-1} and A are symmetric (Appendix 1).

The problem needs therefore a modification of the matrices such that a symmetric matrix will result and SUBROUTINE EV(A,S,N) may be used.

The matrix B can be expressed in the form $B = SDS^T$ where D and S are the matrices of eigenvalues and eigenvectors respectively of matrix B (Section 2.4.2 D).

Hence, $A\Phi = B\Phi L = SDS^T\Phi L$ or $S^TA\Phi = DS^T\Phi L$

Since matrix B is positive-definite, its matrix of eigenvalues (diagonal matrix) will have, for its diagonal elements, only positive terms and hence it is possible to express matrix D as the product of two (diagonal) matrices G and G, where

$$G_{I,I} = \sqrt{D_{I,I}} \text{ and } G_{I,J} = 0.0 \text{ for } I \neq J,$$

without the necessity of dealing with imaginary numbers.

Hence, $S^TA\Phi = DS^T\Phi L = GGS^T\Phi L$

$$\text{or } G^{-1}S^TA\Phi = GS^T\Phi L$$

which can be written as

$$G^{-1}S^TA(SG^{-1}GS^T)\Phi = GS^T\Phi L$$

Using the notations, $\Psi = GS^T\Phi$, $F = SG^{-1}$ and hence

$$F^T = (G^{-1})^T (S^T) = G^{-1} S^T,$$

$$F^T A F \Psi = \Psi L$$

Since $(F^T A F)^T = F^T A F$ by virtue of symmetry of A, $F^T A F$ is symmetric. Letting $F^T A F = C$, $C\Psi = \Psi L$, where C is symmetric.

This equation is hence in a form suitable for the use of SUBROUTINE EV(A,S,N).

The matrix of eigenvalues, L, obtained in such a manner needs no modification as it is the same L in the original equation $A\Phi = B\Phi L$.

From $\Psi = GS^T\Phi$, the matrix of eigenvectors Φ is given by $\Phi = SG^{-1}\Psi$.

The procedure, in brief, is to modify matrix A and obtain matrix C first. The eigenvalues of matrix C are the required eigenvalues. The eigenvectors of matrix C (matrix Ψ) are to be modified by premultiplication by the matrix SG^{-1}.

The importance of generating SG^{-1} and saving it for all modifications at further stages should be noted. The product SG^{-1} is simple to obtain. Denoting G^{-1} as H, and $SG^{-1} = SH$ as T, the elements of T are

$$T_{I,J} = \sum_{K=1}^{N} S_{I,K} H_{K,J}$$

Since G^{-1} or H is a diagonal matrix and hence

$$H_{K,J} = 0 \text{ if } K \neq J,$$

400.4 $$T_{I,J} = S_{I,J} * H_{J,J}$$

The columns of matrix T are thus the columns of matrix S, each multiplied by a different scalar quantity. The product SG^{-1} (=T) can be stored in matrix S itself.

The required modifications may be outlined in the following manner, from the programming viewpoint.

Step 1. Obtain the eigenvalues and eigenvectors of matrix B by using SUBROUTINE EV(B,S,N). Matrix B now stores the eigenvalues of the original matrix B and S is its matrix of eigenvectors. (These correspond to matrices D and S in the derivation.)

Step 2. Obtain the product SG^{-1} and store it in matrix S. Note that each diagonal element of G^{-1} (off-diagonal elements are zero) is merely the reciprocal of the square root of the corresponding diagonal element of matrix B.

Step 3. Obtain the product AS and store it in matrix B using SUBROUTINE MULT(A,S,B,N,N,N).

Step 4. Obtain the product S^TB using the property of symmetry of the resulting matrix and store it in matrix A. The product corresponds to matrix C in the derivation.

Step 5. Obtain the eigenvalues and eigenvectors of matrix A using SUBROUTINE EV(A,B,N). Matrix A now corresponds to matrix L, the matrix of eigenvalues in the derivation (no further modification required for matrix

400.4

L). Matrix B corresponds to matrix Ψ in the derivation.

Step 6. Obtain the product SB by using SUBROUTINE POSTM (S,B,N,N,T) where T is a vector required in that subprogram for computations. Matrix B now stores the required eigenvectors (corresponding to matrix Φ in the derivation).

2.5.3 Features

This subprogram uses other subprograms developed in this package, and coding is mainly a set of CALL statements. Additional storage spaces in the form of an extra matrix and a vector, which are required to carry out the computations, are believed to be the minimum necessary.

2.5.4 Limitations

Original matrices A and B must be symmetric. Matrix B must also be positive definite. The original matrices are destroyed when this subprogram is used. Other subprograms of this package as detailed in the User's Guide (Chapter 4) must be available and loaded when this subprogram is used.

2.6 SUBROUTINE MINV (A,N,DET,NEXCH)

2.6.1 Function

The matrix A of size N rows by N columns is inverted and the inverse returned to the calling program in A itself. Also, the determinant of the original matrix is computed and returned in DET.

2.6.2 Development of the Subprogram

A. The Core of the Subprogram

By a process of elementary transformations (on rows) of A, it is possible to reduce matrix A to the Identity Matrix, referred to as matrix B below.

$$T_M T_{M-1} \cdots T_2 T_1 A = B$$

Hence, $T_M T_{M-1} \cdots T_2 T_1 B = BA^{-1} = A^{-1}$

Thus, the same elementary transformations (on rows) of B will generate A^{-1}.

In the discussion, the elementary transformations will initially be limited to (i) division of a row by a scalar and (ii) addition of a multiple of a row to another row. Later, they will also include row exchanges.

In general, there is no need to follow any definite pattern in the reduction of matrix A to matrix B. The rules below, however, provide a systematic method of achieving the reduction, with a view to easy programming on a computer.

Step 0. Create the Identity Matrix

Step 1. Let K = 1

Step 2. The pivot element is the diagonal element in row K of matrix A. Divide the Kth row of both matrices A and B by the pivot element. The element A(K,K) will then be reduced to 1.0.

Step 3. Add suitable multiples of the (modified) Kth row of matrices A and B to all other rows of the respective matrices to generate 0.0 in the Kth column of matrix A. If the row number is I, -A(I,K) is the suitable multiple.

Step 4. Increment K by 1.

Step 5. If K exceeds N, stop. Otherwise, return to Step 2.

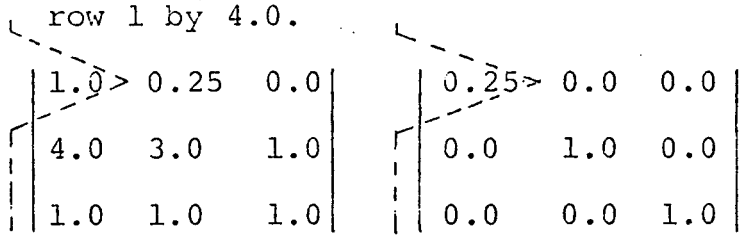
The numerical example below illustrates this procedure and provides as well some useful material for later discussion. The matrix on the left is A and that on the right is B.

Step 1. K = 1

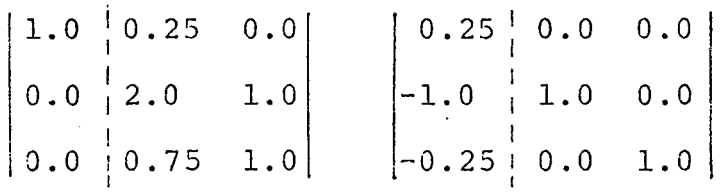
$$\left|\begin{array}{ccc} 4.0 & 1.0 & 0.0 \\ 4.0 & 3.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{array}\right| \quad \left|\begin{array}{ccc} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{array}\right|$$

(The significance of the pair of dashed lines will be explained later.)

Step 2. Pivot element has the value 4.0. Divide row 1 by 4.0.



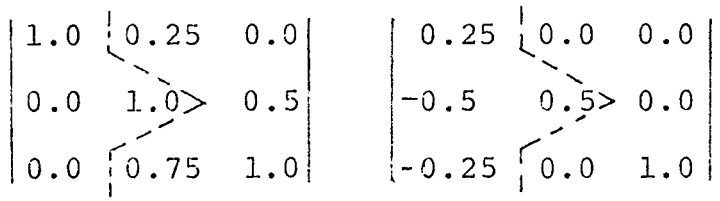
Step 3. Add -4.0 times row 1 to row 2, -1.0 times row 1 to row 3.



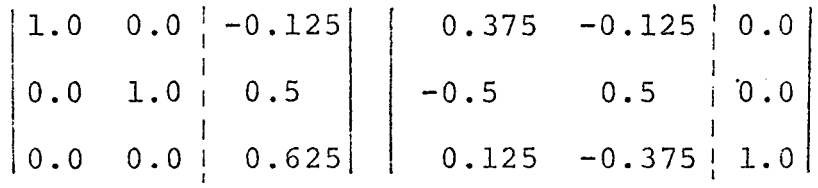
Step 4. K = 1 + 1 = 2

Step 5. 2 $\not>$ 3, Return to Step 2.

Step 2. Pivot element has the value 2.0. Divide row 2 by 2.0.



Step 3. Add -0.25 times row 2 to row 1,-0.75 times row 2 to row 3.



Step 4. K = 2 + 1 = 3.

Step 5. 3 $\not>$ 3. Return to Step 2.

400.4

Step 2. Pivot element has the value 0.625. Divide row 3 by 0.625.

$$\left|\begin{array}{ccc|c} 1.0 & 0.0 & -0.125 & \\ 0.0 & 1.0 & 0.5 & \\ 0.0 & 0.0 & 1.0 > & \end{array}\right| \quad \left|\begin{array}{ccc|c} 0.375 & -0.125 & 0.2 & \\ -0.5 & 0.5 & 0.0 & \\ 0.2 & -0.6 & 1.6 > & \end{array}\right|$$

Step 3. Add -(-0.125) times row 3 to row 1, -0.5 times row 3 to row 2.

$$\left|\begin{array}{ccc|c} 1.0 & 0.0 & 0.0 & \\ 0.0 & 1.0 & 0.0 & \\ 0.0 & 0.0 & 1.0 & \end{array}\right| \quad \left|\begin{array}{ccc|c} 0.4 & -0.2 & 0.2 & \\ -0.6 & 0.8 & -0.8 & \\ 0.2 & -0.6 & 1.6 & \end{array}\right|$$

Step 4. K = 3 + 1 = 4

Step 5. 4 > 3. Stop.

Evidently, the process has to be repeated N times.

The procedure is summarized in the program section below.

```
      CALL DIAG(B,1.0,N)
      DO 1000 K = 1,N
      TEMP = 1.0/A(K,K)
      DO 2000 J = 1,N
      A(K,J) = A(K,J)*TEMP
2000  B(K,J) = B(K,J)*TEMP
      DO 3000 I = 1,N
      IF(I .EQ. K) GO TO 3000
      TEMP = -A(I,K)
      DO 4000 J = 1,N
      A(I,J) = A(I,J) + TEMP*A(K,J)
```

400.4

```
4000  B(I,J) = B(I,J) + TEMP*B(K,J)
3000  CONTINUE
1000  CONTINUE
```

The program requires that i) separate memory space for the matrix B be provided and ii) matrix B be defined initially as the Identity Matrix. The ensuing discussion aims at proving that these requirements need not be met.

Note that, in any problem, the elements that do not lie between the dashed lines will remain the same. These invariant elements need not be stored, if the program can account for them suitably when required. Consequently, only matrix A is assumed to be available. The rules earlier stated need to be modified to suit the new situation.

Step 0. Let K = 1

Step 1. Assume that a column vector corresponding to column K of the Identity matrix is attached to the right of the matrix.

Step 2. The pivot element is the first element in row K. Divide the Kth row by the pivot element and store each element in the column to the left of its original position. This column "shift" has the effect of "pushing" out the element referred to as the pivot element (to the left) and "borrowing" an element from the Identity Matrix

(from the right). It is helpful to visualize that the kinks in the dashed lines have been straightened out by pulling the row K to the left in Step 2 of the numerical example above at each of the N stages.

Step 3. Add -A(I,1) times the Kth row to row I (where $I \neq K$), starting with the element in column 2 of row I. Again, store each element in the column to the left of its original position. Steps 2 and 3 together have the effect of advancing the dashed lines, one column to the right.

Step 4. Increment K by 1.

Step 5. If K exceeds N, stop. Otherwise, return to Step 1.

To illustrate, consider the previous example again.

Step 0. K = 1

Step 1.

$$\left|\begin{array}{ccc|c} 4.0 & 1.0 & 0.0 & 1.0 \\ 4.0 & 3.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 0.0 \end{array}\right|$$

Note that the Kth column of the Identity matrix is available by implication only and is not stored in any vector or matrix.

400.4

Step 2. Pivot element has the value 4.0. Divide row 1 by 4.0.

$$\left|\begin{array}{ccc|c} 0.25 & 0.0 & 0.25 & \\ 4.0 & 3.0 & 1.0 & 0.0 \\ 1.0 & 1.0 & 1.0 & 0.0 \end{array}\right|$$

Step 3. Add -4.0 times row 1 to row 2, -1.0 times row 1 to row 3.

$$\left|\begin{array}{ccc} 0.25 & 0.0 & 0.25 \\ 2.0 & 1.0 & -1.0 \\ 0.75 & 1.0 & -0.25 \end{array}\right|$$

Step 4. K = 1 + 1 = 2

Step 5. $2 \neq 3$; Return to Step 1.

Step 1.

$$\left|\begin{array}{ccc|c} 0.25 & 0.0 & 0.25 & 0.0 \\ 2.0 & 1.0 & -1.0 & 1.0 \\ 0.75 & 1.0 & -0.25 & 0.0 \end{array}\right|$$

Step 2. Pivot element has the value 2.0. Divide row 2 by 2.0.

$$\left|\begin{array}{ccc|c} 0.25 & 0.0 & 0.25 & 0.0 \\ 0.5 & -0.5 & 0.5 & \\ 0.75 & 1.0 & -0.25 & 0.0 \end{array}\right|$$

Step 3. Add -0.25 times row 2 to row 1, -0.75 times row 2 to row 3.

$$\left|\begin{array}{ccc} -0.125 & 0.375 & -0.125 \\ 0.5 & -0.5 & 0.5 \\ 0.625 & 0.125 & -0.375 \end{array}\right|$$

Step 4. K = 2 + 1 = 3

Step 5. 3 ≯ 3. Return to Step 1.

Step 1.

$$\left|\begin{array}{rrr|r} -0.125 & 0.375 & -0.125 & 0.0 \\ 0.5 & -0.5 & 0.5 & 0.0 \\ 0.625 & 0.125 & -0.375 & 1.0 \end{array}\right|$$

Step 2. Pivot element has the value 0.625. Divide row 3 by 0.625.

$$\left|\begin{array}{rrr|r} -0.125 & 0.375 & -0.125 & 0.0 \\ 0.5 & -0.5 & 0.5 & 0.0 \\ 0.2 & -0.6 & 1.6 & \end{array}\right|$$

Step 3. Add -(-0.125) times row 3 to row 1, -0.5 times row 3 to row 2

$$\left|\begin{array}{rrr} 0.4 & -0.2 & 0.2 \\ -0.6 & 0.8 & -0.8 \\ 0.2 & -0.6 & 1.6 \end{array}\right|$$

Step 4. K = 3 + 1 = 4

Step 5. 4 > 3. Stop.

In the program section that follows, note especially the manner in which the elements of the Kth column of the Identity matrix are assumed to be available. Further, note also the necessity of generating the elements of the Nth column outside the DO-loops.

```
      NM1 = N-1
      DO 4000 K = 1,N
      T = 1.0/A(K,1)
      DO 8000 J = 1,NM1
8000  A(K,J) = A(K,J+1)*T
      A(K,N) = T
      DO 9000 I = 1,N
      IF (I.EQ.K) GO TO 9000
      BIG = -A(I,1)
      DO 8888 J = 1,NM1
8888  A(I,J) = A(I,J+1) + BIG*A(K,J)
      A(I,N) = BIG*T
9000  CONTINUE
4000  CONTINUE
```

B. The Problem of Vanishing Pivot Element

An important tacit assumption in the above development is that the pivot element never has the value zero and hence, division by the value of that element is legal. This is equivalent to the assumption that no submatrix of A in the upper left is singular. (If matrix A is itself singular, there is obviously no solution to the problem.)

If the assumption does not hold, as in SUBROUTINE DETMT (A,DA,N), here also row interchanges need to be performed to overcome the difficulty. If the Identity

matrix is assumed to be available, the following is a simple example for illustration of the procedure.

$$\begin{vmatrix} 0.0 & 2.0 \\ 3.0 & 4.0 \end{vmatrix} \quad \begin{vmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{vmatrix}$$

Pivot element has the value of 0.0.

Interchange rows as division by zero is not permitted. Note that the rows of the Identity matrix are also to be interchanged.

$$\begin{vmatrix} 3.0 & 4.0 \\ 0.0 & 2.0 \end{vmatrix} \quad \begin{vmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \end{vmatrix}$$

Step 1. K = 1

Step 2. Pivot element has the value 3.0. Divide row 1 by 3.0

$$\begin{vmatrix} 1.0 & 1.33 \\ 0.0 & 2.0 \end{vmatrix} \quad \begin{vmatrix} 0.0 & 0.33 \\ 1.0 & 0.0 \end{vmatrix}$$

Step 3. Add -(0.0) times row 1 to row 2. This leaves the matrices unaffected.

Step 4. K = 1 + 1 = 2

Step 5. $2 \neq 2$. Return to Step 2.

Step 2. Pivot element has the value 2.0. Divide row 2 by 2.0.

$$\begin{vmatrix} 1.0 & 1.33 \\ 0.0 & 1.0 \end{vmatrix} \quad \begin{vmatrix} 0.0 & 0.33 \\ 0.5 & 0.0 \end{vmatrix}$$

Step 3. Add -1.33 times row 2 to row 1.

$$\begin{vmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{vmatrix} \quad \begin{vmatrix} -0.67 & 0.33 \\ 0.5 & 0.0 \end{vmatrix}$$

Step 4. K = 2 + 1 = 3

Step 5. 3 > 2. Stop.

Next, attempt the solution without the availability of the Identity matrix.

Step 0. K = 1

Step 1.
$$\left|\begin{array}{cc|c} 0.0 & 2.0 & 1.0 \\ 3.0 & 4.0 & 0.0 \end{array}\right|$$

Rows of A will be interchanged, but since the Kth column of the Identity matrix is available only by implication, the elements here cannot be interchanged. Hence

$$\left|\begin{array}{cc|c} 3.0 & 4.0 & 1.0 \\ 0.0 & 2.0 & 0.0 \end{array}\right|$$

Step 2. Pivot element has the value 3.0. Divide row 1 by 3.0.

$$\left|\begin{array}{cc|c} 1.33 & 0.33 & \\ 0.0 & 2.0 & 0.0 \end{array}\right|$$

Step 3. Add -(0.0) times row 1 to row 2.

$$\left|\begin{array}{cc|c} 1.33 & 0.33 & \\ 2.0 & 0.0 & \end{array}\right|$$

Step 4. K = 1 + 1 = 2

Step 5. 2 ≯ 2. Return to Step 1

Step 1.
$$\left|\begin{array}{cc|c} 1.33 & 0.33 & 0.0 \\ 2.0 & 0.0 & 1.0 \end{array}\right|$$

Step 2. Pivot element has the value 2.0. Divide row 2 by 2.0.

$$\left|\begin{matrix} 1.33 & 0.33 \\ 0.0 & 0.5 \end{matrix}\right| \quad 0.0$$

Step 3. Add -1.33 times row 2 to row 1.

$$\left|\begin{matrix} 0.33 & -0.67 \\ 0.0 & 0.5 \end{matrix}\right|$$

Step 4. $K = 2 + 1 = 3$

Step 5. $3 > 2$. Stop.

The answer here differs from the previous (correct) answer in that the columns have been interchanged. This is a direct consequence of our failure to interchange rows of the Identity matrix in the modified method. The remedy is simple. Merely interchange the columns.

The numerical example above provides the necessary background for the development of the generalized approach which follows.

C. Row Interchanges and Corrective Modifications

Assume that the rows of matrix A are interchanged. Row interchange is one of the elementary transformations and is equivalent to premultiplying a matrix by the corresponding elementary transformation matrix. Briefly, $TA = C$ where C is the resulting matrix after the interchange. Hence $C^{-1} = A^{-1}T^{-1}$ and $C^{-1}T = A^{-1}$.

The product $C^{-1}T$ implies that the same elementary transformation T operates on C^{-1} but this time post-multiplication by T is involved. The conclusion is that, to get A^{-1} from C^{-1}, the columns of C^{-1} have to be interchanged to match the interchange of rows of A.

Applying the principles to the 2 x 2 example, we see that, in effect, given A, a matrix C was generated by interchange of rows of A, and C^{-1} was found. The answer thus obtained needs, therefore, further modification (interchange of columns of C^{-1}).

With a 2 x 2 matrix, the number of interchanges is limited to 1. Also, if the interchange is necessary, it will be at stage K = 1. In the case of larger matrices, the number may be as high as N-1 and the necessity may arise at some or all stages from K = 1 to K = N-1.

In general, we may assume that the original matrix after the required number of row interchanges has been modified during the process itself to a new matrix C given by $C = T_M T_{M-1} \cdots T_2 T_1 A$

Hence $CA^{-1} = T_M T_{M-1} \cdots T_2 T_1$

and $A^{-1} = C^{-1} T_M T_{M-1} \cdots T_2 T_1$

The last and first row interchanges are represented by T_M and T_1 respectively. In the modification, however, the first and last column interchanges are represented

by T_M and T_1 respectively.

So, if a number of row interchanges was involved, the corresponding column interchanges must be performed in reverse order. Appendix 1 contains a more formal proof.

D. The Best Pivotal Element

In the discussion so far, row interchange was suggested only if the pivotal element was identically zero. The pivot element could be small in absolute value and hence division by that element and later modifications of other elements may lead to inaccuracies.

On the one hand, there is the need to check whether the pivot element is zero. If so, on the other hand, there is the need to search other rows (limited to rows K through N of column K so as to preserve the action of the previous modifications of the reduction process) for a non-zero element.

Combining all of these, as in SUBROUTINE DETMT(A,DA, N), assume that a search will be made for the largest absolute valued element in the column containing the pivot element. Three possibilities arise as a result of the search.

1. Every element in the column is zero. The matrix is then singular and A^{-1} does not exist. (Note that the subprogram does not produce an error message in this instance. A message will result from the system when the

machine attempts later to handle infinity and the program will be aborted.)

2. The pivot element itself is the largest in absolute value and hence, no exchange is necessary.

3. An exchange is necessary.

Vector NEXCH is a bookkeeping vector which takes into account items 2 and 3 above. The elements of this vector get defined during the reduction process as C^{-1} is generated and are used later to perform the corresponding column exchanges (in reverse order).

E. The Determinant of the Original Matrix A as a By-Product

A comparison of the procedures involved in this subprogram and SUBROUTINE DETMT (A,DA,N) shows that the reduction processes in the two cases have much in common. In the latter case, the process was limited to reducing the given matrix to an upper triangular matrix. In this subprogram (see the reduction process assuming the existence of the Identity matrix), this process has been extended so that the given matrix is diagonalized. This extension which merely includes more elementary transformations does not alter the value of the determinant. The determinant is hence the product of the pivotal elements (before their reduction to 1.0).

F. The Special Case of a 1 x 1 Matrix

An inspection of the DO-loop parameters in the final version of the subprogram indicates that, if N = 1, the ranges of some DO-loops would be from 1 to zero. This difficulty is avoided by branching out control early in the subprogram.

2.6.3 Features

It has been shown that (even) in the case of a general square matrix (unsymmetric), the inversion process can be carried out within the space of the original matrix itself, provided no submatrix in the upper left of the original matrix is singular. To handle matrices of a general nature, only an additional vector of N elements is required. Speed is considerably enhanced as constants are not generated during execution.

2.6.4 Limitations

Original matrix A is destroyed.

2.6.5 Additional Remarks

It is not essential that NEXCH be a vector of integer elements in the calling program. The use of that vector is local to the subprogram and the user needs only to be aware that any such vector, if defined prior to the CALL statement in the calling program, may be destroyed when this subprogram is used.

2.7 SUBROUTINE MOVE (A,B,M,N)

2.7.1 Function

The matrix B is defined to be the same as the given matrix A. Both the matrices are of the same size M rows by N columns.

2.7.2 Development of the Subprogram

The subprogram listed in Appendix 2 follows directly from the relationship $B_{I,J} = A_{I,J}$. The matrices are treated as single-subscripted arrays in the subprogram.

2.7.3 Features

In some cases, a matrix referenced in the CALL statement of a calling program gets destroyed in the subprogram when using the routines of this package.

Assume that the original matrix is needed later in the calling program. In such situations, the use of this subprogram, prior to a CALL to the routine where the matrix gets destroyed, assures the availability of the original matrix by a different name. The example below illustrates this procedure.

```
CALL MOVE(A,B,N,N)
CALL DETMT(A,DA,N)
```

The original matrix A is destroyed by the second CALL statement. However, by virtue of the first CALL statement, matrix B is the same as the original matrix A and can be operated on as if it were matrix A itself.

In the example, if the second CALL statement is CALL DETMT(B,DA,N), matrix B gets destroyed and the original matrix A is available for further operations by its own name.

2.8 SUBROUTINE MULT (A,B,C,L,M,N)

2.8.1 Function

The product of matrix A (size L rows by M columns) and matrix B (size M rows by N columns) is made available to the calling program as matrix C (size L rows by N columns). C = AB

2.8.2 Development of the Subprogram

In algebraic terms, the elements of matrix C are given by

$$C_{I,J} = \sum_{K=1}^{M} A_{I,K} \cdot B_{K,J}$$

and the following subprogram may, therefore, be written.

```
      SUBROUTINE MULT (A,B,C,L,M,N)
      COMMON/IYENGAR/ I,J,K,Y (11)
      REAL A(L,M),B(M,N),C(L,N)
      DO 1000 I = 1,L
      DO 1000 J = 1,N
      C(I,J) = 0.0
      DO 1000 K = 1,M
1000  C(I,J) = C(I,J) + A(I,K)*B(K,J)
      RETURN
      END
```

It is worth noting here that for a specific set of values of I and J, C(I,J) is completely defined only after the innermost loop is satisfied. Hence, instead of referring to C(I,J) which is a double-subscripted variable, it is advantageous to refer to a scalar quantity, say, SUM when this loop is being executed. This modification saves time of execution.

To illustrate, assume two matrices each of size 100 x 100 are multiplied. The modified version listed in Appendix 2 saves two million address computations. The saving in time in a trial run was approximately 6 seconds (23.262 seconds vs. 17.148 seconds).

2.8.3 Limitations

In general, each of the matrices A, B and C will differ in size. Hence, no attempt is made in this subprogram to store the product matrix C in either matrix A or matrix B.

The size of matrix C will match the size of either matrix A or matrix B or both, if matrix B or matrix A or both are square. The use of SUBROUTINE PMULT(A,B,K,L,X) or SUBROUTINE POSTM(A,B,K,L,X) should be considered under these conditions.

In any case, when SUBROUTINE MULT(A,B,C,L,M,N) is used, the product matrix C should be distinct from both the matrices A and B, although matrices A and B may

themselves be identical.

2.9 SUBROUTINE OUTE (A,I,J,TITLE,TITEL) and ENTRY OUTF and ENTRY OUTG

2.9.1 Function

Matrix A of size I rows by J columns is printed. Printing begins on a new page. Rows and columns are numbered. Labels (up to 20 characters) provided by the user appear at the top of each page.

If $J \leq 10$, and at the same time, $I \leq 25$, the matrix is printed on one page. If $J > 10$, the first 10 columns of matrix A are printed until all the rows (25 or less to a page) are exhausted. Then the second 10 columns (or less) of matrix A are printed until all the rows (25 or less to a page) are exhausted. And so on.

The user has the choice of printing the elements in E-, F- or G-FORMAT.

2.9.2 Development of the Subprogram

A. Matrix Partitioning

The limits of 10 columns to a page and 25 rows (double-spaced) to a page have been chosen taking into consideration the number of characters that can be printed per line (135 excluding the carriage control character in column 1), the number of lines (about 63)

per page, and ease of readability.

Since J columns are to be printed, the number of vertical partitions of the matrix is given by JMAX below. The use of integer arithmetic (FORTRAN) should be noted here.

$$JMAX = (J + 9)/10$$

If $1 \leq J \leq 10$, JMAX = 1

If $11 \leq J \leq 20$, JMAX = 2, and so on.

In a similar way, the number of horizontal partitions of the matrix is given by

$$IMAX = (I + 24)/25$$

Schematically, the partitions are as in Figure 2. (I and J are assumed to be 58 and 45 respectively. Hence, IMAX and JMAX have the values 3 and 5 respectively.)

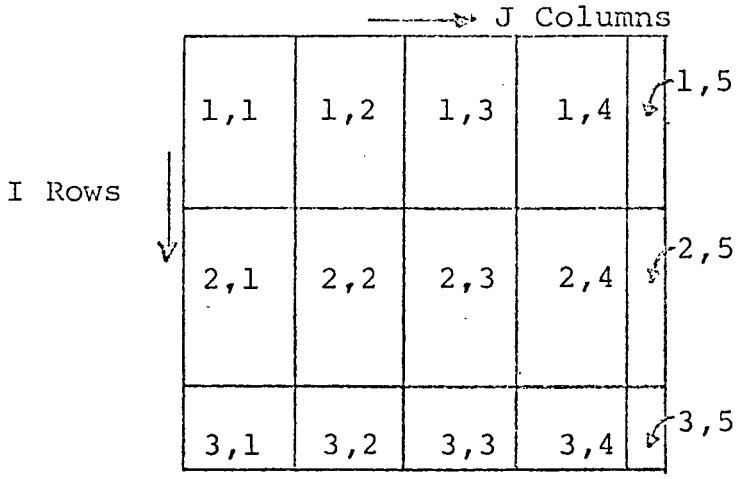


Figure 2

The required number of pages for the printout is given by the product of JMAX and IMAX.

The choice is made to print the matrix in vertical partitions. Hence, for the example in Figure 2, the submatrix marked 1,1 will be printed on the first page. This is followed by printing of submatrix marked 2,1 and then by the one marked 3,1.

The process is next repeated for submatrices marked 1,2 and 2,2 and 3,2 in that order. And so on.

The outermost DO loop hence ranges from 1 to JMAX. This is followed by the next DO loop whose range is 1 to IMAX.

Consider next the items on a specific page. Let the submatrix being printed on the page have indices IROW and JCOL.

The table below gives the first and the last row and column numbers (of matrix A) printed on the page.

First row	25 (IROW) - 24
Last row	25 (IROW) or I, whichever is less
First column	10 (JCOL) - 9
Last column	10 (JCOL) or J, whichever is less

B. FORMAT Control

The form of output (E-,F- or G-FORMAT) is controlled as follows. The FORMAT for printing elements of the matrix is generated in an integer array IV. If the CALL statement is CALL OUTE (A,I,J,TITLE,TITEL), the character

stored in IV (2) is E in a suitable part of that (computer) word, and hence the output is in E-FORMAT. Similarly, CALL OUTF(...) and CALL OUTG(...) produce output in F- and G-FORMAT respectively.

The X preceding F12.5 in IV (2) is deliberately built in to ensure at least one blank space between two numbers in a row, when printing in F-FORMAT. This limits the number of characters (digits plus the negative sign, if any) to the left of the decimal point to 6. If an element has 7 or more characters to the left of the decimal point, an asterisk will be printed by the machine (CDC 6400) at the beginning of the field, warning the user that the most significant digit/s and/or the negative sign, if any, are not printed. Briefly, the number is too "large" for a suitable printout under this FORMAT control. On the other hand, if the number is too small, only zeroes may appear on the printout. Under these conditions, as well as under conditions where the magnitudes of the elements being output are unknown or unpredictable, the E-FORMAT should be preferred.

The E-FORMAT output is not as easily readable as the F-FORMAT output but this is a minor inconvenience, particularly after experience is gained in reading the numbers.

When G-FORMAT is specified, five significant digits appear on output in F-FORMAT if the number lies in the

range 100000.0 > absolute value ≥ 0.1. Otherwise, the number is output in E-FORMAT. That is, the machine makes the choice between F-FORMAT and E-FORMAT, depending on the magnitude of the number to be output. The decimal points in G-FORMAT output may not line up vertically and, in general, the output will be a mixture of E- and F-FORMAT outputs. Some elegance is thereby lost. On the other hand, once G-FORMAT output is specified, concern need not be wasted on losing the most significant digits (5 digits in this subprogram).

C. Labels

The user may label the matrix being output by using appropriate "values" for the arguments TITLE and TITEL. These "values" are Hollerith character strings up to a maximum of 10 each.

The labels will appear at the top of each page. Further, these labels are followed by the word CONTINUED in parentheses on the second and succeeding pages of output to indicate that parts of the matrix have already been output in previous pages.

The size of the matrix also is indicated by a printout directly after labelling. If no labels are to be used, TITLE and TITEL must be matched with 1HØ and 1HØ respectively (Ø denotes blank). Examples on the use of labels

are given in the User's Guide (Chapter 4).

2.9.3. Limitations

A matrix with a large number of columns and only a few rows (say less than 10), when output using this subprogram, will use several sheets of paper. Consider, for example, a matrix 8 rows by 25 columns. The output, in this case, is on 3 sheets. If, however, the transpose of the matrix is printed, the output is on one sheet. Apart from economy of paper, a further advantage of the latter procedure is that an overall view of the matrix is available without having to turn pages.

The user should weigh these advantages against the inconvenience of having to transpose the matrix and reading the transposed matrix.

2.10 SUBROUTINE PMULT(A,B,K,L,X)

2.10.1 Function

The square matrix B of size L rows by L columns is premultiplied by a rectangular matrix A of size K rows by L columns and the product is returned to the calling program in the rectangular matrix A. A = AB

2.10.2. Development of the Subprogram

In SUBROUTINE MULT(A,B,C,L,M,N), it was remarked that the three matrices A, B, and C are of different sizes when dealing with general rectangular matrices and hence, matrix C could not be stored in either matrix A or matrix B. If, however, matrix B is square, it follows from

$$[A]_{K \times L} \cdot [B]_{L \times L} = [C]_{K \times L}$$

that the product matrix C is of the same size as the rectangular matrix A, which premultiplies the square matrix B. Advantage is taken of this feature to save memory space by storing matrix C in matrix A.

Schematically, the order in which the elements of the product matrix are generated (row by row) is shown in Figure 3.

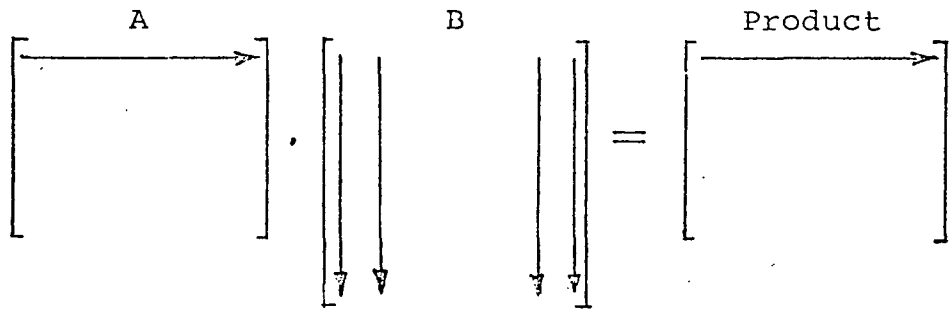


Figure 3

400.4

Consider row 1 of matrix A. This row is used in conjunction with each column of matrix B to generate elements in row 1 of the product matrix. After this is done, row 1 of matrix A is never referenced again. It is for this reason that row 1 of the product matrix may be stored in row 1 of matrix A. (The argument is valid for all the rows in turn.)

However, throughout the process of generating the elements in row 1 of the product matrix, the elements in row 1 of the original matrix A must be available. Hence the necessity of a vector X of size L elements for intermediate storage of the generated product elements.

The procedure involves, therefore, the following steps for all I from 1 to K.

1. Generate elements in row I of the product matrix and store these in vector X.
2. Transfer the contents of vector X to row I of matrix A.

2.10.3 Features

Memory requirements are less when this subprogram is used. A vector X of L elements will suffice to perform the operations, whereas, previously a matrix of size K rows by L columns was required (SUBROUTINE MULT (A,B,C,K,L,L)).

See also Section 2.11.5.

400.4

2.10.4 Limitations

Matrix B must be square.

The advantage of saving of memory space is partly offset by a loss in speed of execution, as additional operations (transfer of vector X to a row of matrix A for each I, I = 1 to K) are necessary.

Original matrix A is destroyed.

2.10.5 Additional Remarks

Matrix A may be rectangular or square.

If matrix A is square, it should differ from matrix B at least by name. That is, the product of a square matrix and itself (say, matrix A times matrix A) cannot be obtained with the use of this subprogram. SUBROUTINE MULT (A,A,PROD,K,K,K) of this package is to be used to obtain such a product in matrix PROD which is distinct from matrix A.

2.11 SUBROUTINE POSTM(A,B,K,L,X)

2.11.1 Function

The square matrix A of size K rows by K columns is postmultiplied by the rectangular matrix B of size K rows by L columns and the product is returned to the calling program in the rectangular matrix B. B = AB

2.11.2 Development of the Subprogram

The logic here is essentially the same as in SUBROUTINE PMULT(A,B,K,L,X). The elements of the product matrix are generated column by column. Schematically, this process is represented by Figure 4.

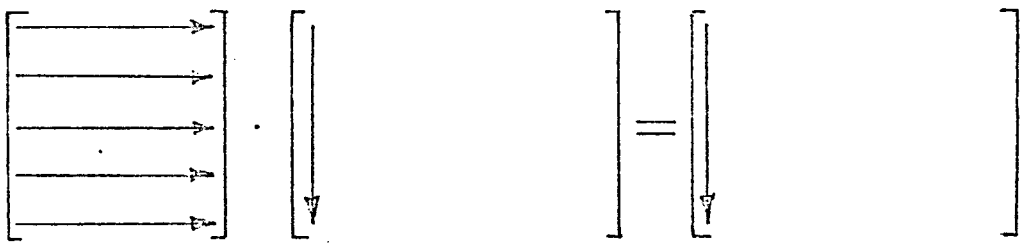


Figure 4

The process involves the following steps for all J from 1 to L.

1. Generate elements in column J of the product matrix and store these in vector X which is of size K elements.
2. Transfer the contents of vector X to column J of matrix B.

2.11.3 Features

Memory requirements are less when this subprogram is used. A vector X of K elements will suffice to perform the operations, whereas, previously a matrix of size K rows by L columns was required. (SUBROUTINE MULT (A,B,C,K,K,L).)

2.11.4 Limitations

Matrix A must be square.

Original matrix B is destroyed.

The advantage of saving of memory space is partly offset by a loss in speed of execution, as additional operations (transfer of vector X to a column of matrix B for each J, J = 1 to L) are necessary.

2.11.5 Additional Remarks

Matrix B may be rectangular or square.

If matrix B is square, it should differ from matrix A at least by name. That is, the product of a square matrix and itself (say, matrix B times matrix B) cannot be obtained with the use of this subprogram. SUBROUTINE MULT (B,B,PROD,K,K,K) of this package is to be used to obtain such a product in matrix PROD which is distinct from matrix B.

In the following discussion concerning the three multiplication routines described so far, assume that matrices

A and B are square and differ at least by name. For compatibility in matrix multiplication, they are necessarily of the same size, say K rows by K columns.

The user, in this case, has a choice of subprograms as indicated in the table below, the choice being governed by the criterion of saving one, none, or both the matrices.

To Save	Use CALL Statement
Matrix A	CALL POSTM(A,B,K,K,X)
Matrix B	CALL PMULT(A,B,K,K,X)
Neither matrix	Either of the above two
Both matrices	CALL MULT(A,B,C,K,K,K)

2.12 SUBROUTINE RDCBC(A,M,N)
SUBROUTINE RDCOLG(A,M,N)
SUBROUTINE RDRBR(A,M,N)
SUBROUTINE RDROWG(A,M,N)

2.12.1 Function

Values are read in from data cards for the elements of matrix A which is of size M rows by N columns.

2.12.2 Development of the Subprogram

A. The FORMAT Declaration

For convenience in punching values, a field width of 10 columns for each piece of data is prescribed. Thus, 8 values can be punched per card. In brief, the FORMAT used is (8F10.0).

This choice of FORMAT requires that the decimal point be punched in the field unless it is to be assumed (by the machine) to be at the end of the field.

On the CDC6400 it is permissible to punch data in E-FORMAT also under the same FORMAT control. Care must, however, be exercised to see that the exponent is placed at the end of a field.

B. Order in Assigning Values

The order in which the values are assigned to the elements depends on the user's choice of one of the four routines. The details are given in the User's Guide (Chapter 4).

2.12.3 Limitations

The decimal point must be punched in the field, unless it is to be assumed to be at the end of the field. The maximum number of characters (digits plus the negative sign, if any) in any data piece is limited to 10 if the decimal point is not punched and to 9, otherwise.

2.12.4 Additional Remarks

If an element is to be assigned the value zero, the corresponding data field may, at user's option, be left blank. In this case, the machine actually assigns the value -0.0 to the element.

SUBROUTINE RDCOLG(A,M,N) is the fastest of the four routines since the transfer of data occurs, with its use, in the natural order of storage of array elements (in the machine) without any interruptions. It is also the most economical one in terms of field length requirements (see Section 3.4).

2.13 SUBROUTINE SCMUL(A,M,N,X)

2.13.1 Function

Elements of the given matrix A of size M rows by N columns are modified so that the fresh elements are X times the original ones.

2.13.2 Development of the Subprogram

Each element of the original matrix is multiplied by the scalar X and the result stored back in the element itself. The matrix is treated as a single-subscripted array in the subprogram.

2.13.3 Limitations

The original matrix A is destroyed.

2.14 SUBROUTINE SINV(A,DA,N)

2.14.1 Function

The matrix A, which is symmetric (size N rows by N columns) and positive-definite, is inverted and the inverse returned to the calling program in matrix A itself. The determinant of the original matrix A is computed and returned in DA.

2.14.2 Development of the Subprogram

A. Introduction

In civil engineering applications, the matrices that need to be inverted are, in general, symmetric or can be rendered symmetric; e.g., stiffness matrix, flexibility matrix. These matrices are also positive-definite.

If the features of symmetry and positive-definite nature are utilized in a program, the inversion process can be speeded up, the comparison being with SUBROUTINE MINV(A,N,DET,NEXCH) which can invert matrices that may be symmetric or unsymmetric.

B. Procedure

The symmetric matrix A can be expressed in the form $A = \lambda\lambda^T$ where λ is a lower triangular matrix.

The elements of λ are given by (Appendix 1)

$$\lambda_{J,J} = (A_{J,J} - \sum_{K=1}^{J-1} \lambda_{J,K}^2)^{\frac{1}{2}}$$

$$\lambda_{I,J} = \frac{1}{\lambda_{J,J}} (A_{I,J} - \sum_{K=1}^{J-1} \lambda_{I,K} \lambda_{J,K}) \qquad \text{for } I > J$$

$$\lambda_{I,J} = 0 \qquad \text{for } I < J$$

From $A = \lambda\lambda^T$, $\quad A^{-1} = (\lambda^T)^{-1}(\lambda^{-1}) = (\lambda^{-1})^T(\lambda^{-1})$

The inverse of a lower-triangular matrix is also lower-triangular. If $\mu = \lambda^{-1}$, the elements of μ are (Appendix 1)

$$\mu_{I,J} = 0 \qquad \text{for } I < J$$

$$\mu_{I,I} = \frac{1}{\lambda_{I,I}}$$

$$\mu_{I,J} = -\frac{1}{\lambda_{I,I}} \sum_{K=J}^{I-1} \lambda_{I,K} \mu_{K,J} \qquad \text{for } I > J$$

Briefly, then, the procedure involves 1) factorization of A into λ and λ^T, 2) obtaining λ^{-1} and 3) forming the product $(\lambda^{-1})^T(\lambda^{-1})$ to give A^{-1}.

The algorithm below shows that these operations can be accomplished within the space of the original matrix itself.

By symmetry, matrix A can be considered fully defined if all the elements on and above the diagonal are known. The space below the diagonal is hence available to store

elements of matrix λ. Since λ is a lower triangular matrix, all its elements above the diagonal have the value 0.0 and these elements will therefore neither be generated or stored.

Regarding the diagonal elements of matrix λ, it is fortunate that, once $\lambda_{I,I}$ is generated, $A_{I,I}$ is no longer required and hence, it is possible to store $\lambda_{I,I}$ in place of $A_{I,I}$ for all I.

In fact, it is advantageous to store the reciprocal of $\lambda_{I,I}$ in place of $A_{I,I}$, since the diagonal elements will then be elements of matrix (λ^{-1}) or of matrix $(\lambda^{-1})^T$. Refer to the schematic representation of this process in Figure 5.

The order of generating elements of matrix λ is as follows. First, $\lambda_{1,1}$ is computed as $A_{1,1}^{\frac{1}{2}}$. Next, the other elements of column 1 are computed from $\lambda_{I,1} = \frac{1}{\lambda_{1,1}} (A_{1,I})$ using the symmetry of matrix A $(A_{1,I} = A_{I,1})$. For all J in the range $(N-1) \geq J \geq 2$, the diagonal element is computed first, and the other elements of column J next. When J = N, only the diagonal element needs to be computed.

The next step is to generate matrix λ^{-1}. The diagonal elements of matrix $(\lambda^{-1})^T$, which are the same as those of matrix λ^{-1}, are already available. λ^{-1} is a lower triangular matrix. Since matrix λ occupies the lower half of the matrix space at this stage and matrix A is no longer

required, matrix $(\lambda^{-1})^T$ instead of matrix λ^{-1} is stored in the upper half of the matrix space. Refer again to the schematic representation. If now $\mu = (\lambda^{-1})^T$, the elements of matrix μ, which is upper-triangular, are

$$\mu_{J,I} = 0 \qquad \text{for } J > I$$

$$\mu_{I,I} = \frac{1}{\lambda_{I,I}}$$

$$\mu_{J,I} = -\frac{1}{\lambda_{I,I}} \left(\sum_{K=J}^{I-1} \lambda_{I,K} \mu_{J,K} \right) \qquad \text{for } I > J$$

The off-diagonal elements of matrix $(\lambda^{-1})^T$ are generated in the subprogram, column by column.

The product $(\lambda^{-1})^T(\lambda^{-1})$ is next required. The resulting matrix is A^{-1}, and the matrix λ is no longer required. Matrix A^{-1} can hence be stored in the lower half of the matrix space.

For clarity, assume $X = (\lambda^{-1})^T$, $Y = (\lambda^{-1}) = X^T$ and $Z = A^{-1} = (\lambda^{-1})^T(\lambda^{-1}) = XY$.

Then

$$Z_{I,J} = \sum_{K=1}^{N} X_{I,K}Y_{K,J} = \sum_{K=1}^{N} X_{I,K}X_{J,K} \text{ since } Y = X^T$$

Expansion of $Z_{I,J}$ yields (for $I \geq J$)

$$Z_{I,J} = X_{I,1}X_{J,1} + X_{I,2}X_{J,2} + \ldots + X_{I,J}X_{J,J} +$$

$$X_{I,J+1}X_{J,J+1} + \ldots + X_{I,I}X_{J,I} +$$

$$X_{I,I+1}X_{J,I+1} + \ldots + X_{I,N}X_{J,N}$$

For $L > M$, $X_{L,M}$ is zero, since X is an upper-triangular matrix. Hence, for $I \geq J$,

$$Z_{I,J} = \sum_{K=I}^{N} X_{I,K} X_{J,K} .$$

Assume that the elements $Z_{I,J}$ (elements of matrix A^{-1}) are generated row by row. The index I therefore varies from 1 to N, while index J varies from 1 to I for each I. A study of the following three expressions shows that, in this scheme, once $Z_{I,J}$ is generated, $X_{J,I}$ is no longer required.

$$Z_{I,J} = X_{I,I} X_{J,I} + X_{I,I+1} X_{J,I+1} + \ldots + X_{I,N} X_{J,N}$$

Hence, $Z_{I,J+1} = X_{I,I} X_{J+1,I} + X_{I,I+1} X_{J+1,I+1} + \ldots + X_{I,N} X_{J+1,N}$

and $Z_{I+1,J} = X_{I+1,I+1} X_{J,I+1} + X_{I+1,I+2} X_{J,I+2} + \ldots + X_{I+1,N} X_{J,N}$

Hence, $X_{J,I}$ may be overwritten by the generated value of $Z_{I,J}$ each time. The result is matrix A^{-1}.

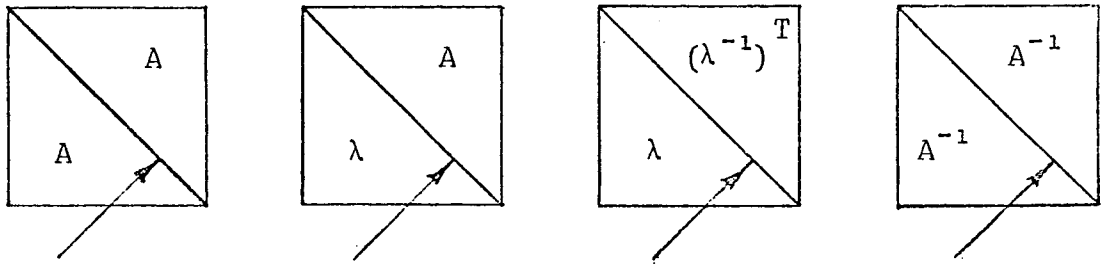


Figure 5

C. Built-in Subprogram

The computations require summations of the form $\sum_{K=INIT}^{IFIN} A_{I,K}A_{J,K}$ on four different occasions. It is well-known that subprograms are used in such situations to save coding and reduce the number of machine instructions. Since, however, such a subprogram may not have the same degree of usefulness in other programs or subprograms, a compromise is struck by incorporating this feature in the subprogram itself in an indirect manner.

This specific area of calculations in the subprogram is referenced whenever the required summations are to be computed, through the use of unconditional GO TO statements (analagous to CALL statements). The RETURN statement of a regular subprogram is simulated by an assigned GO TO statement, of which the control variable is INDEX. This variable is assigned the suitable statement label prior to each occasion when the area of the "built-in" subprogram is referenced. The four variables I, J, INIT, IFIN also need definition. The DO-loop indices themselves are generally used to account for this feature.

D. The Determinant of the Original Matrix as a By-Product

Since $A = \lambda\lambda^T$, $\det(A) = \det(\lambda).\det(\lambda^T)$. Noting that a matrix and its transpose have the same determinant, $\det(A) = \det(\lambda).\det(\lambda)$. Since λ is a lower-triangular

matrix, its determinant is merely the product of its diagonal elements. The computation of det(A) is thus incidental to the inversion process.

E. The Case of a 1 x 1 Matrix

If N = 1, many DO-loops in the routine would have their ranges such as 2 to 1. The difficulty is avoided by testing whether N equals 1 early in the routine and if so, calculating det(A) and A^{-1} directly.

2.14.3 Features

Although the inverse of a symmetric matrix is symmetric, it is conceivable that a symmetric matrix inverted by the use of SUBROUTINE MINV(A,N,DET,NEXCH) does not result in a perfectly symmetric matrix due to machine round-off errors.

This subprogram, however, utilizes the property of symmetry of matrix A and hence, guarantees the symmetric nature of its inverse.

The algorithm of this subprogram utilizes only the elements on and above the diagonal of the original matrix A. Hence , only these need be defined when using this subprogram. The inverse will be fully defined in any case.

2.14.4 Limitations

Matrix A must be symmetric and also positive-definite. It is destroyed in the subprogram. The necessity of matrix A being positive-definite requires some elaboration. If det(λ) = x , det(A) = x^2. If det(A) and hence x^2 are negative, x is imaginary. Since det(λ) = x is the product of the diagonal elements of matrix λ, this implies that the number of imaginary diagonal elements of matrix λ is odd.

If det(A) is positive, but a submatrix of A has a negative determinant, there will be an even number of imaginary diagonal elements in matrix λ.

If det(A) is zero, matrix A is singular and its inverse has no definition.

All of which leads to the conclusion that, if matrix A is positive-definite, there will be no need to deal with imaginary numbers. This restriction has, therefore, been imposed in developing this subprogram and, in view of the introductory remarks, is a justifiable one.

However, if matrix A is not positive-definite (and non-singular), its inverse does exist and the additional problem is that of handling imaginary numbers. It is possible to modify this subprogram without actually using complex number routines. This has been done in another version of this subprogram. An even more general subprogram

employing pivoting during factorization of matrix A has also been attempted. However, these are obviously longer and slower in execution. They are not included in this package.

2.14.5 Additional Remarks

In solving civil engineering problems, where the inversion of either the flexibility matrix or the stiffness matrix is involved, it is recommended that this subprogram be used in preference to SUBROUTINE MINV(A,N,DET,NEXCH), at least during the development stage of a program.

For, the use of this subprogram will prevent full execution if the matrix is not positive-definite. Logical errors (or punching errors) in the generation of the matrix are thereby indicated.

It is possible to rewrite this subprogram using only about half the matrix space (upper or lower triangle plus the diagonal). In such a case, it is suspected that the maximum use of the built-in subprogram may not be feasible. Further, a partial definition of the inverse is of questionable benefit.

2.15 SUBROUTINE SOLVE(A,B,N,L,DET)

2.15.1 Function

A system of linear simultaneous equations is solved.

2.15.2 Development of the Subprogram

A. Procedure

The system solved is AX = B. The dimensions of the matrices are as under.

Matrix	Size
Coefficient Matrix A	N rows by N columns
Right-Hand Side B	N rows by L columns

Only the spaces of matrices A and B are used and the solution matrix X is returned in matrix B.

Matrix A is reduced to an upper-triangular matrix in essentially the same manner as in SUBROUTINE DETMT(A,DA,N). The only major difference is that, instead of column exchanges, row exchanges are made, if necessary, in this subprogram. This variation is required in order to preserve the correct order of the elements in the columns of the solution matrix X.

The reduction process is equivalent to forming suitable linear combinations of the original set of equations. Representing these operations as a set of elementary row transformations given by T, we have, TAX = TB. Hence, matrix B also needs modification in the same manner.

At the conclusion of the reduction process, the simultaneous equations have the form

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & & A_{1,N} \\ & A_{2,2} & \cdots & & A_{2,N} \\ & & A_{I,I} & \cdots & A_{I,N} \\ & \text{(Zeroes)} & & A_{N-1,N-1} & A_{N-1,N} \\ & & & & A_{N,N} \end{bmatrix} \cdot$$

$$\begin{bmatrix} \cdots & X_{1,K} & \cdots \\ \cdots & X_{2,K} & \cdots \\ \cdots & X_{I,K} & \cdots \\ \cdots & X_{N-1,K} & \cdots \\ \cdots & X_{N,K} & \cdots \end{bmatrix} = \begin{bmatrix} \cdots & B_{1,K} & \cdots \\ \cdots & B_{2,K} & \cdots \\ \cdots & B_{I,K} & \cdots \\ \cdots & B_{N-1,K} & \cdots \\ \cdots & B_{N,K} & \cdots \end{bmatrix}$$

Starting with the last row and assuming that a typical column K of matrix X is being generated,

$A_{N,N}X_{N,K} = B_{N,K}$. Hence $X_{N,K} = B_{N,K}/A_{N,N}$

$A_{N-1,N-1}X_{N-1,K} + A_{N-1,N}X_{N,K} = B_{N-1,K}$

Hence, $X_{N-1,K} = (B_{N-1,K} - A_{N-1,N}X_{N,K})/A_{N-1,N-1}$

For $1 \leq I \leq N-1$,

$$A_{I,I}X_{I,K} + A_{I,I+1}X_{I+1,K} + \ldots + A_{I,N}X_{N,K} = B_{I,K}$$

Hence,

$$X_{I,K} = (B_{I,K} - \sum_{J=I+1}^{N} A_{I,J}X_{J,K})/A_{I,I}$$

It is interesting to note the order in which the elements of a column of the solution matrix X are generated. The last element is generated first and the first generated last. The process is hence generally referred to as back-substitution.

Since $B_{I,K}$ is not required after $X_{I,K}$ is generated, it is possible to store matrix X in matrix B.

B. The Determinant of the Coefficient Matrix as a By-Product

The generation of this quantity is incidental to the reduction process. Since the original matrix A has been (upper) triangulated, the determinant is given by the product of the diagonal elements of the reduced matrix A.

C. The Special Case of N = 1

In this case, some DO-loops would have the range 1 to 0. The difficulty is avoided by branching out early in the subprogram to make the evident direct calculations to get final results.

2.15.3 Limitations

The original matrices A and B are destroyed.

2.16 SUBROUTINE SQTR(A,N)

2.16.1 Function

The transpose of the square matrix A of size N rows by N columns is returned to the calling program in matrix A itself.

2.16.2 Development of the Subprogram

A general rectangular matrix A of size M rows by N columns and its transpose, matrix B of size N rows by M columns differ in dimensions. If, however, matrix A is square, its transpose also has the same dimensions. It is, therefore, possible to transpose a square matrix in its own space.

If N = 1, the matrix and its transpose are identical and control is merely returned to the calling program in this case.

The transpose operation does not affect the values of the diagonal elements. These are, therefore, never referenced in the subprogram.

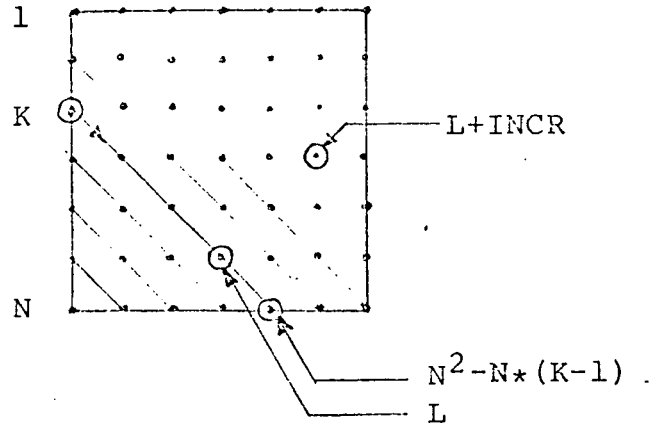


Figure 6.

The matrix is treated as a single-subscripted array. In Figure 6, consider the element in row K of column 1. Proceeding parallel to the main diagonal from this element, the subsequent elements (of the matrix when treated as a single-subscripted array) have subscripts in increments of N+1 and the last element has the subscript $N^2-N*(K-1)$. The elements of the lower triangle can thus be referenced using two DO-loops, one with index K for elements in column 1, and the other with index L (for a specific K) for elements parallel to the main diagonal.

For a typical element with subscript L in the lower triangle, the matching element in the upper triangle has the subscript L+INCR, where INCR = (K-1)*(N-1). These elements are exchanged.

2.16.3 Features

The matrix is transposed in its own space.

2.16.4 Limitations

The matrix must be square. The original matrix is destroyed.

2.17 SUBROUTINE SUB(A,B,C,M,N)

2.17.1 Function

Matrix B is subtracted from matrix A and the result returned to the calling program in matrix C. C = A-B.

Each matrix is of size M rows by N columns.

2.17.2 Development of the Subprogram

In algebraic terms, $C_{I,J} = A_{I,J} - B_{I,J}$. This subprogram is hence the same as SUBROUTINE ADD(A,B,C,M,N), except for the sign. The matrices are treated as single-subscripted arrays in the subprogram.

2.17.3 Additional Remarks

The resultant matrix C can be stored in either of the original matrices (say B). In this case, the original matrix (matrix B) will be destroyed.

2.18 SUBROUTINE TMULT(A,B,C,L,M,N)

2.18.1 Function

The transpose of matrix A (A is of size L rows by M columns) is post-multiplied by matrix B (size L rows by N columns) to give matrix C (size M rows by N columns). $C = A^T B$.

2.18.2 Development of the Subprogram

In civil engineering applications, products of the type $A^T B$ are required in computations quite often. Using two of the subprograms in this package, the product may be obtained as follows.

```
CALL TRANS(A,TEMP,L,M)
CALL MULT(TEMP,B,C,M,L,N)
```

Such an approach requires a temporary matrix TEMP of size M rows by L columns. The transpose operation, as well as the storage required for TEMP, may be saved by a slight modification of the multiplication routine.

Assume that A^T = TEMP and [TEMP]·[B] = C. An element of C is then defined by

$$C_{I,J} = \sum_{K=1}^{L} TEMP_{I,K} \cdot B_{K,J}$$

$$= \sum_{K=1}^{L} A_{K,I} \cdot B_{K,J} \quad \text{by virtue of } A^T = TEMP.$$

This formulation shows that matrix TEMP does not need either generation or storage.

2.18.3 Limitations

The product matrix C should be distinct from matrices A and B. However, matrices A and B may be identical.

2.19 SUBROUTINE TRANS(A,B,M,N)

2.19.1 Function

The transpose of matrix A (size M rows by N columns) is made available to the calling program as matrix B (size N rows by M columns).

2.19.2 Development of the Subprogram

The subprogram utilizes the property $B_{J,I} = A_{I,J}$.

2.19.3 Limitations

If matrix A is square, its transpose (matrix B) is also square. However, if they do not differ by name, the resulting matrix will always be symmetric. Hence, the matrices should be distinct from each other in every case.

2.19.4 Additional Remarks

SUBROUTINE SQTR(A,N) is more efficient than this subprogram in terms of memory space if a square matrix is to be transposed and the original matrix may be destroyed.

2.20 SUBROUTINE XABATC(A,B,C,L,M,X)

SUBROUTINE XABTA (A,B,L,N,X)

SUBROUTINE XABTC (A,B,C,L,M,N)

SUBROUTINE XATBAC(A,B,C,L,M,X)

SUBROUTINE XATBB (A,B,L,N,X)

2.20.1 Function

Products of matrices i) $C = ABA^T$, ii) $A = AB^T$, iii) $C = AB^T$, iv) $C = A^TBA$ and v) $B = A^TB$ are obtained.

2.20.2 Development of the Subprograms

The concepts used in the earlier multiplication routines from the bases for the development of this set also. The dimensions of the matrices are as noted in the User's Guide (Chapter 4) for the several routines.

2.20.3 Additional Remarks

i) The multiplication routines of this package [barring SUBROUTINE SCMUL(A,M,N,X)] may be classified into three groups as under:

Group 1	Group 2
MULT ($C = AB$)	PMULT ($A = AB$)
	POSTM ($B = AB$)
TMULT ($C = A^TB$)	XATBB ($B = A^TB$)
XABTC ($C = AB^T$)	XABTA ($A = AB^T$)

Group 3

XABATC ($C = ABA^T$)
XATBAC ($C = A^TBA$)

a. Groups 1 and 2 have the following features:

	Group 1	Group 2
Product Matrix	Storage is in a matrix C which is distinct from both the original matrices A and B in every case (that is, even if one or both the original matrices match the product matrix in size).	Storage is in one of the matrices A or B, as indicated in the above notations.
Dimensions	All the matrices are assumed to be rectangular (some or all of them may, however, be square).	Matrix A must be square, if the product is stored in matrix B, and vice versa (both the matrices may, however, be square).
Matrix Destroyed	None	The original matrix that bears the name of the product matrix finally (or, in other words, the matrix that is not assumed to be square) is destroyed.
Vector X	Vector X is not required.	Vector X is of size NS elements if the square matrix is of size NS rows by NS columns.
Speed of Execution	Faster	Slower, because of the additional operation of replacement of a column or a row by vector X.
Memory Requirement (Arrays)	More	Less, since vector X will suffice instead of a matrix.

b. Group 3 has the following features.

Matrix B must be square as well as symmetric. A vector X of size NS elements, if matrix B is of size NS rows by NS columns, is required. Matrix A may be rectangular or square. The product matrix C is distinct from both the matrices A

and B (even if matrix A is square). Matrix C is square as well as symmetric. The original matrices A and B are not destroyed in the subprograms.

ii) Other Products. Products as indicated on the left of the following table may be obtained by the set of CALL statements on the right, provided both the matrices A and B are square (the original matrix B need not be symmetric in any of these cases). All the matrices are assumed to be of size N rows by N columns. Vector X is of size N elements.

	Product	CALL Statements
a.	$A = A^TB$	CALL SQTR(A,N) CALL PMULT(A,B,N,N,X)
b.	$B = AB^T$	CALL SQTR(B,N) CALL POSTM(A,B,N,N,X)
c.	$A = A^TBA$	CALL XATBB(A,B,N,N,X) CALL POSTM(B,A,N,N,X)
d.	$B = A^TBA$	CALL XATBB(A,B,N,N,X) CALL PMULT(B,A,N,N,X)
e.	$A = ABA^T$	CALL XABTA(B,A,N,N,X) CALL PMULT(A,B,N,N,X)
f.	$B = ABA^T$	CALL POSTM(A,B,N,N,X) CALL XABTA(B,A,N,N,X)

In items c and e above, the original matrix B is destroyed, even though the final product is formed in matrix A. The original matrix A is not destroyed when the final product is in matrix B.

3. GENERAL NOTES

3.1 COMMON Block Labelled IYENGAR

A. Purpose

In normal usage, a COMMON block establishes the required correspondence amongst the variables in the routines sharing the block. Values of the variables are thus transmitted from one routine to another.

The purpose of the block in this package is totally different. It merely serves as a means to economize on the memory space requirements for the temporary variables used in the several subprograms.

SUBROUTINE OUTE(A,I,J,TITLE,TITEL) with ENTRY OUTF and ENTRY OUTG uses 14 temporary variables, and this is the maximum number required in any routine [with the exception of SUBROUTINE EV(A,S,N) which will be treated as a special case later in this discussion]. Consider, next, SUBROUTINE SINV (A,DA,N) which requires 9 temporary variables. Together, these two routines would need 23 locations in memory for these variables.

Assume, now, the existence of a COMMON block 14 words long, the length corresponding to the maximum number of temporary variables in any routine. Let each set of temporary variables in the two routines belong to this block. Then, 14 locations will suffice for the variables, whereas previously 23 were required. The advantage is even more pronounced

if the principle is extended by specifying that the temporary variables of each routine in the entire package belong to the same COMMON block. The same 14 locations still suffice for all the temporary variables, as against 131 which would have been required without the use of the COMMON block. (To maintain the length of the block at 14 in each subprogram, a filler array Y of variable size is used wherever necessary.)

Regarding the values associated with the variables, it is immaterial whether these variables belong to the COMMON block or not. This is because of the nature of the variables themselves. They get defined in a subprogram before actual use in the subprogram. The values they finally attain, at the time of exit from the subprogram, are inconsequential to any other subprogram since they get redefined before use in that subprogram.

SUBROUTINE EV(A,S,N) actually uses more than 18 temporary variables by name. The number is reduced to 18 through the use of an EQUIVALENCE declaration. Of these, 14 are included in the COMMON block. The other four variables FN, IND, THRI and THRF are not. Recall that this subprogram has an ENTRY IEV statement for improvement of eigenvalues and eigenvectors. Assume that SUBROUTINE EV(A,S,N) has been CALL-ed in the calling program. Then, the four variables listed have some values in the subprogram at the time of exit from the subprogram. These values need to be preserved for a sub-

sequent improvement procedure, if any. The calling program may use the improvement procedure immediately (that is, before CALL-ing other subprograms). There is no special problem if it does. If not, assume that other subprograms in this package will be used by the calling program before the improvement procedure. In such a situation, these variables (whose values are not regenerated during the improvement phase) will be affected in values if they are included in the COMMON block. Their values are preserved by excluding them from the block.

Finally, therefore, 18 memory locations are required for the temporary variables in the entire package.

B. COMMON Block Label

COMMON blocks may or may not have labels (or numbers). The name blank COMMON applies if the block has no label (or number). Else, the block is a labelled (or numbered) COMMON block.

If blank COMMON were used in this package, the user would be unable to specify another independent blank COMMON block of variables of his own choice in the main program and other (user's) subprograms, since there can exist only one blank COMMON block in a program.

Labelled (or numbered) COMMON blocks can be numerous, the maximum number depending on the machine used. (In CDC 6400 at

Lehigh, the number of labelled and numbered COMMON blocks is limited to 61). One requirement of these blocks is that each label (or number) used in a program must be unique.

The originator of this package has chosen his last name for the label in the justifiable hope that a general user will not dream up this identical label for any of his labelled COMMON blocks. All the same, it is necessary to emphasize that the label IYENGAR should be considered taboo by the user for any of his labelled COMMON blocks when using this package. If the label is used, the variables listed by the user in the block may get redefined with each CALL to a subprogram of this package.

400.4

3.2 USE OF VARIABLE (ADJUSTABLE) DIMENSIONS

3.2.1 Adopted Procedure

In this package, the dimensions of all arrays used in the subprograms are adjustable dimensions. The motivation for this approach is explained in what follows.

A primary rule in communication between the calling program and the called subprogram is that the row dimensions of all double-subscripted arrays which are arguments must match exactly in both routines. This follows from the fact that double-subscripted arrays are stored column by column in the machine.

Consider SUBROUTINE MULT(A,B,C,L,M,N) as an example for discussion in the rest of this section, Sec. 3.2.

The dimensions of arrays A, B and C are specified in the subprogram by the declaration

```
REAL A(L,M),B(M,N),C(L,N)
```

If, in the calling program, a matrix A (of size 10 rows by 15 columns) is to be postmultiplied by a matrix B (of size 15 rows by 20 columns) to give the product matrix C (of size 10 rows by 20 columns), the proper declaration in the calling program is

```
REAL A(10,15),B(15,20),C(10,20)
```

and the corresponding CALL statement is

```
CALL MULT(A,B,C,10,15,20)
```

In this case, the dimensions match automatically. Note

that the array sizes chosen are exact. Under-dimensioning will not work in any scheme, and over-dimensioning does not permit suitable matching since double-subscripted arrays are involved.

3.2.2 A Limitation of the Adopted Procedure

Occasionally, a matrix with the same name but of different sizes needs to be used in a calling program. The dimension declaration in such a case must obviously account for at least the maximum size of the matrix. (Note that a subscripted variable can be dimensioned only once in a program.)

As an example, matrix A may vary in size, say, from 2 rows by 3 columns to 10 rows by 15 columns. Then the dimension declaration is

```
REAL A(10,15)
```

Assuming that there are other similar matrices, the declaration may be

```
REAL A(10,15),B(15,20),C(10,20)
```

No difficulty is experienced if the problem to be solved involves the maximum dimensions; e.g., CALL MULT(A,B,C,10,15,20). However, when solving problems using dimensions less than the prescribed maximum dimensions, the sizes will not match. For example,

```
CALL MULT(A,B,C,6,9,12)
```

implies that, in the subprogram, matrix A has dimensions

6 rows by 9 columns, whereas in the calling program it has been dimensioned for size 10 rows by 15 columns.

To rectify the situation, the subprograms themselves need to be modified slightly as suggested in the following subsection.

3.2.3 Suggestion for Modification by the User

If array sizes are variable in the calling program, the required subprograms may be modified on the following lines.

The dimension declaration for the arrays in the calling program should correspond to (at least) the maximum size of each array in the program.

As an example,

```
REAL A(10,15), B(15,20),C(10,20)
```

For matching the arrays in the subprograms and specifying the particular sizes to be operated on, the following is a suitable set of statements.

In the calling program,

```
CALL MULT (A,10,B,15,C,10,6,9,12)
```

In the subprogram,

```
SUBROUTINE MULT (A,NRA,B,NRB,C,NRC,L,M,N)
REAL  A(NRA,1),B(NRB,1),C(NRC,1)
```

Recall that whenever variable dimensioning is used in subprograms, the array names as well as the variables which represent the dimensions must be formal parameters. The last three parameters in the argument lists represent

the sizes of the matrices to be operated on. The rest prescribe the array names and maximum row dimensions.

The modified form is obviously suitable even if the sizes are not variable in the calling program. Hence, an additional advantage of this procedure is that the user may choose suitable maximum dimensions for matrices in a program for solving problems of a particular type. The dimension declaration, in such a case, needs no further attention from problem to problem. However, it is worthwhile to remember that memory space is wasted when problems that do not involve maximum dimensions are solved.

The disadvantages of this procedure, attributable to the longer argument list, are these:

1. Mistakes in matching arguments in CALL statements are more likely.
2. There is a loss of efficiency since addresses of a larger number of arguments need to be passed back and forth by the system.

In the authors' opinion, the disadvantages outweigh the advantages. Also, the additional type of problem covered by the modified procedure is of infrequent occurrence.

3.2.4 A Special Procedure

A few users have violated the rule of matching dimensions and used the earlier version (FCMXPK) of this package successfully without any modifications.

The procedure is stated here only for the sake of completeness of this discussion on variable dimensions. It is not recommended for general use as special care needs to be exercised in programming.

Assuming that the required matrices are overdimensioned, the matrices in this procedure are defined through the use of relevant subprograms only. As an example,

```
REAL A(10,15),B(15,20),C(10,20)
CALL RDROWG (A,6,9)
CALL RDROWG (B,9,12)
```

Here, 54 elements of matrix A and 108 elements of matrix B get defined. The sequence of storage of the elements in this case, although not unpredictable, is dependent on the extent of over-dimensioning and will hence change with each problem. For the sake of brevity, the elements may be said to be in "wrong" locations. If, now, CALL MULT (A,B,C,6, 9,12) is used, the machine again refers to the same wrong locations of elements in matrices A and B. Since this is, however, a consistent process in the machine, matrix C will have proper values but its elements are again stored in some other wrong locations.

The importance of printing each array through the use of other subprograms is, by now, evident. In these subprograms, the machine again refers to the same wrong locations and the printed values will therefore be right.

In short, this procedure will be successful if the main program always CALL-s the relevant subprograms whenever arrays are to be defined through reading in data from cards, operated on, or printed.

Wastage of memory space can be considerable if the special procedure is adopted in solving problems where the sizes of the matrices in the calling program remain stationary.

3.3 RESERVED NAMES

When executing a FORTRAN main program with several subprograms, it is essential that each of these bear a unique name so that the machine does not get "confused". The rules of operation vary with each (machine) system when this requirement is violated. In the following, the discussion applies to the CDC 6400 installation at the Lehigh University Computing Center.

If more than one routine by the same name is loaded in the machine, the one that was loaded first is recognized by the machine as the one meant for use* Assume that this package is loaded first in the machine and the user's subprogram SUBROUTINE SOLVE (X,Y,Z) is loaded subsequently. Since the matrix package was loaded first and contains a subprogram SUBROUTINE SOLVE (A,B,N,L,DET), the user's subprogram SUBROUTINE SOLVE (X,Y,Z) will be ignored during execution. If this is the intention, there is no problem.

The converse is equally true. If the user's subprogram is loaded first, the subprogram SUBROUTINE SOLVE (A,B,N,L,DET) of the matrix package will be ignored.

Hence, it is best to treat all the 30 names used for the routines in this package as reserved names, and choose other names for the user's program and subprograms.

*For loading sequence, see User's Guide (Section 4.4).

400.4

Further, it is obligatory that the name of every variable used in a routine differ from the name of any subprogram CALL-ed by the routine.

(A reminder -- the name IYENGAR is a reserved name when labelling COMMON blocks.)

3.4 SUBPROGRAM LENGTHS

The length of each subprogram in octal numbers, using the RUN and FTN compilers of the CDC 6400 installation at Lehigh, is indicated below. Small variations in these numbers are likely with periodic revisions to the compilers. The length of the COMMON block is 16_8.

Sl.No.	Subprogram	Length (octal) RUN	FTN(OPT=2)
1	ADD	31	42
2	DETMT	141	221
3	DIAG	33	43
4	EV (IEV)	405	364
5	GEVP	146	215
6	MINV	230	254
7	MOVE	26	37
8	MULT	53	77
9	OUTE (OUTF,OUTG)	233	224
10	PMULT	63	105
11	POSTM	62	101
12	RDCBC	46	52
13	RDCOLG	27	34
14	RDRBR	46	52
15	RDROWG	50	52
16	SCMUL	24	40
17	SINV	217	255
18	SOLVE	265	362
19	SQTR	50	65
20	SUB	31	42
21	TMULT	53	103
22	TRANS	36	57
23	XABATC	105	130
24	XABTA	64	102
25	XABTC	54	74
26	XATBAC	102	131
27	XATBB	62	104
	COMMON block	16	16
	TOTAL	3555	4436

[The earlier version (FCMXPK) required 4265_8 words (RUN) and had fewer routines.]

The package uses two routines, SQRT and ABS, available in the machine.

The following notes are especially for users of the CDC 6400 at Lehigh.

Although this package may be catalogued as one unit on a permanent file, it is not necessary to load the entire package each time the file is attached. Selection of routines is possible through the use of control cards (COPY routines) such as COPYN, to minimize field length requirements for the job.(2) Note, however, that if SUBROUTINE GEVP (A,B,S,T,N) is selected, other routines as mentioned in the User's Guide (Chapter 4) must also be selected.

SUBROUTINE OUTE (A,I,J,TITLE, TITEL) must be selected if any of the three print routines (OUTE,OUTF,OUTG) is required by a calling program.

Selection of SUBROUTINE EV (A,S,N) assures the availability of the associated ENTRY IEV.

Finally, such selection by the user may become unnecessary in due course, because it is anticipated that the loader itself will be modified to select routines, as required, for each job.(3)

3.5 TEMPORARY VECTORS

A few subprograms of this package require the use of temporary vectors to store intermediate values in computations. The matching vectors in the calling program must obviously be dimensioned. Such vectors, if they have been defined in the calling program, will be destroyed in the subprograms.

It is sufficient to provide for only one temporary vector in the calling program, even if it uses more than one subprogram of the above type. The vector should be dimensioned for the maximum size required in the use of all such subprograms. It may then be used repeatedly in all the corresponding CALL statements. (Alternately, the several vectors that result may be EQUIVALENCE-d at their starting addresses to save memory space.)

SUBROUTINE GEVP(A,B,S,T,N), in this regard, is a special case. It requires the use of a temporary matrix S in addition to that of a temporary vector T. In the calling program, matrix S may be matched either by a matrix or by a vector of N*N elements. Assuming matrix S is matched by a vector, the two vectors corresponding to S and T must be independent of each other in the calling program. In other words, they do not share the same memory locations.

3.6 Features of the CDC 6400

The routines in this package were developed using the CDC 6400 computer at Lehigh. Some of the features of this machine are noted below, since minor modifications may be necessary when this package is used in other machines.

1. The name of a variable, routine, label etc., is limited to 7 characters.

In this package, only the COMMON block label (IYENGAR) is 7 characters long. All other names used are shorter.

2. On the line printer, a maximum of 135 characters (not counting the carriage control character in column 1) can be printed on a line.

Practically full advantage of this feature is taken in the output routines of this package.

3. An ENTRY statement has no associated argument list. However, in the CALL statement corresponding to the ENTRY statement, the argument list (if any) of the routine in which the ENTRY statement appears must be matched.

Example: CALL IEV(A,S,N)

4. Conversion of an integer to its real form may be done by an assignment statement across the equality sign.

See line 34 of the coding in SUBROUTINE EV(A,S,N)

5. The length of each computer word is 60 bits. Hence, it is possible to store 10 alphanumeric characters in each word.

A maximum of 20 characters may thus be used to match the two variables TITLE and TITEL in the output routines of this package. The other entities in these routines using the same feature are arrays Y and IV, and the A-field in the FORMAT statement.

3.7 Use of Single Subscripts in Selected Routines

In only a few routines, the (double-subscripted) matrices are treated as single-subscripted arrays. These are some of the simpler routines which could be so written with the advantages of saving memory space and execution time. Trials with a few other routines showed that such modifications lead to one of the advantages at the expense of the other, the execution time being more generally the item hurt. Further probing in this area seems warranted.

4. USER'S GUIDE

4.1 INTRODUCTION

The subprograms in this package were developed by Mr. Sampath Iyengar of the Computer Systems Group in Fritz Engineering Laboratory, for use by members of the Laboratory. Dr. C. N. Kostem is the Chairman of the Computer Systems Group.

This version supersedes the earlier one (FCMXPK - Fortran Callable Matrix Package) which will be withdrawn by a date to be specified. Mr. Edward T. Manning, Jr. was associated with Mr. Iyengar in the development of FCMXPK.

In this Guide, only a brief description of the function, limitations and requirements of each subprogram is included. More detailed information on how the subprograms were developed are available in Chapters 2 and 3.

The package includes routines for matrix manipulation as under:

OPERATION	SUBPROGRAM
Add matrices	ADD
Determinant of matrix	DETMT, MINV, SINV, SOLVE
Create Diagonal matrix	DIAG
Eigenvalues of symmetric matrices	EV, IEV, GEVP
Invert matrix	MINV, SINV
Copy matrix	MOVE
Multiply matrices	MULT, PMULT, POSTM, SCMUL, TMULT, XABATC, XABTA, XABTC, XATBAC, XATBB
Print matrix	OUTE, OUTF, OUTG
Read matrix	RDCBC, RDCOLG, RDRBR, RDROWG
Simultaneous equations	SOLVE
Transpose matrix	SQTR, TRANS
Subtract matrices	SUB

4.2 GENERAL LIMITATIONS

The subprograms do not provide any diagnostics if operations which are not possible mathematically, such as inversion of a singular matrix, are attempted. The requirements of the subprograms are not tested prior to or during execution and hence, when the requirements are violated, the answers obtained are, in general, unpredictable and wrong.

Only the following subprograms provide printouts:

i) SUBROUTINE OUTE (A,I,J,TITLE,TITEL)

ii) SUBROUTINE OUTF (A,I,J,TITLE,TITEL)

iii) SUBROUTINE OUTG (A,I,J,TITLE,TITEL)

SUBROUTINE RDCBC (A,M,N), SUBROUTINE RDCOLG (A,M,N), SUBROUTINE RDRBR (A,M,N) and SUBROUTINE RDROWG (A,M,N) enable reading in values from data cards for the matrix A. All others are "calculation" subprograms.

In all the subprograms, "variable" or "adjustable" dimensions are used for the several arrays. The user must, therefore, prescribe exact dimensions for all his arrays to be handled by this package. Overdimensioning, except under special circumstances of usage (Chapter 3), may lead to wrong results. Underdimensioning invariably produces wrong results or aborts the execution of the program.

For Input-Output operations, card input and printer output are assumed.

The limitations on the sizes of the matrices that can be

handled are not due to any feature in programming involved in this package but due to the capacity of the machine used.

The label IYENGAR may not be used in the user's program for any of his labelled COMMON blocks. Similarly, the names of routines in this package may not be used in the user's program for his program or any of the subprograms.

4.3 DISCLAIMER

The burden of proof on the validity and applicability of this package to a particular problem rests with the user, and not the authors. No guarantee is stated or implied that the package will give correct results, or that the mathematical relations and assumptions used are proper and applicable to the problem under consideration by the user. The authors cannot be held responsible for incorrect results or damages resulting from the use of the package, although it is believed that the package is correctly formulated.

The authors welcome suggestions for improvements and notice of any errors. If a correction is possible and implemented, proper publicity will be given to the revised status of the package. Else, the concerned subprogram will be withdrawn.

4.4 DECK SETUP

This package is expected to be available as a permanent file at Lehigh University Computing Center, in due course.

At the present time, a prospective user who belongs to Fritz Laboratory may borrow a binary deck and make his own copy. Interested users will please contact the authors.

A sample deck setup, using the binary deck, is as follows:

```
Job Card ...
RUN(S)
LOAD(INPUT)
LGO.
7/8/9
User's FORTRAN Program with Subprograms, if any
7/8/9
Binary Deck (FLMXPK)
7/8/9
Data, if any
6/7/8/9
```

The matrix package will be loaded first and the user's program next. To reverse the order, replace the control cards LOAD(INPUT) and LGO. by LOAD(LGO) and INPUT., respectively.

4.5 DESCRIPTION OF THE SUBPROGRAMS

From the user's viewpoint, there are, in all, 30 routines in this package. These will be described in alphabetical order under the following headings:

a. Function:

b. Calling Program:

i) Dimensions: Those that are required in the calling program.

If the calling program is the main program, the dimensions must be stated in terms of absolute numbers, such as

REAL A (10, 15). If it is a subprogram, the dimension statement is either of the same form or of the form REAL A (M,N) where M and N have been defined through operations prior to the use of the subprogram. (The user is obliged, in the latter instance, to include A, M and N in the argument list of his subprogram.)

All the subscripted variables handled by this package are "real" variables. [The solitary exception, vector NEXCH in SUBROUTINE MINV (A,N,DET,NEXCH), needs no special consideration by the user.] Mistakes often occur when this fact is overlooked and the user prescribes an "integer" name for what is clearly an array of "real" variables. An example from civil engineering is to refer to the stiffness matrix as K without a corresponding TYPE statement. An easy solution is to dimension the arrays in a TYPE statement such as REAL K (20,20), TEMP (20). An advantage here is that the user has a free choice of names. Further, a separate DIMENSION statement need not be (in fact, should not be) provided. Inclusion in the TYPE statement of names of arrays which are "real" even without a TYPE statement, like TEMP in the example, does not hurt in any way.

ii) Definitions: Arrays and variables that must be defined prior to or in the CALL statement.

iii) Values Returned to the Calling Program:

c. Limitations (if any):

The general limitations mentioned in Section 4.2 will not be repeated.

c. or d. Additional Notes (if any):

c. or d. or e. Examples of CALL Statement:

4.5.1 SUBROUTINE ADD (A,B,C,M,N)

a. Function:

Add matrices A and B and store the sum in matrix C. (C = A + B).

b. Calling Program:

i) Each matrix is of size M rows by N columns.

ii) Matrices A and B, as well as integers M and N, must be defined.

iii) Matrix C is defined in the subprogram.

c. Additional Notes:

The resultant matrix may be stored in one of the original matrices. Only in such a case, the specific original matrix will be destroyed.

d. Examples:

i) CALL ADD (A,B,C,M,N)

ii) CALL ADD (A,B,C,15,20)

iii) CALL ADD (A,B,A,10,15)

4.5.2 SUBROUTINE DETMT (A,DA,N)

a. Function:

The determinant of the given (square) matrix A is made available to the calling program as DA. DA = det(A).

b. Calling Program:

i) Matrix A is of size N rows by N columns.

ii) Matrix A and integer N should be defined.

iii) DA is defined in the subprogram.

c. Limitations:

The original matrix A is destroyed.

d. Examples:

i) CALL DETMT (A,DA,N)

ii) CALL DETMT (ARRAY,DET,20)

4.5.3 SUBROUTINE DIAG (A,DA,N)

a. Function:

A diagonal matrix A is generated as follows: Each diagonal element has the value DA, and each off-diagonal element has the value zero.

b. Calling Program:

i) Matrix A is of size N rows by N columns.

ii) The value of the diagonal element DA and integer N must be defined.

iii) Matrix A is defined in the subprogram.

c. Examples:

i) CALL DIAG (A,DA,N)

ii) CALL DIAG (A,1.0,15)

iii) CALL DIAG (ARRAY,DE,NSIZE)

4.5.4 SUBROUTINE EV (A,S,N)

a. Function:

Eigenvalues and eigenvectors of the symmetric matrix A are computed.

b. Calling Program:

i) Matrices A and S are of size N rows by N columns.

ii) Matrix A and integer N must be defined.

iii) On return to the calling program, matrix A has, for its diagonal elements, the eigenvalues of the original matrix A and matrix S has, for its columns, the corresponding eigenvectors.

c. Limitations:

The original matrix A must be symmetric. It will be destroyed in the subprogram, as the eigenvalues are returned in the same matrix.

d. Additional Notes:

The eigenvalues and eigenvectors may be improved further, if so desired, by using SUBROUTINE IEV (A,S,N).

e. Examples:

i) CALL EV (A,S,N)

ii) CALL EV (ARRAY,EVEC,10)

4.5.5 SUBROUTINE GEVP (A,B,S,T,N)

a. Function:

Eigenvalues and eigenvectors of the given matrix A where [A] {X} $=\lambda$[B] {X} are computed. Both the matrices A and B are symmetric, and further matrix B is also positive-definite.

b. Calling Program:

i) Matrices A, B and S are of size N rows by N columns. T is a vector of size N elements.

ii) Matrices A and B, as well as integer N, should be defined.

iii) The eigenvalues are returned as the diagonal elements of matrix A and the corresponding eigenvectors as the columns of matrix B. Matrix S and vector T are used for storing some intermediate values in computations.

c. Limitations:

Matrices A and B must be symmetric. Matrix B must also be positive-definite. Both the original matrices A and B are destroyed in the subprogram.

The following subprograms of this package must be available and loaded when this subprogram is used:

i) SUBROUTINE EV (A,S,N)

ii) SUBROUTINE MULT (A,B,C,L,M,N)

iii) SUBROUTINE POSTM (A,B,K,L,X)

d. Examples:

i) CALL GEVP (A,B,S,T,N)

ii) CALL GEVP (EVAL,EVEC,S,TEMP,10)

4.5.6 SUBROUTINE IEV (A,S,N)

a. Function:

Eigenvalues and eigenvectors of the symmetric matrix A computed by the use of SUBROUTINE EV (A,S,N) are improved.

b. Calling Program:

Same as in SUBROUTINE EV (A,S,N).

c. Limitations:

Same as in SUBROUTINE EV (A,S,N).

d. Additional Notes:

The accuracy of calculations in SUBROUTINE EV (A,S,N) is prescribed according to the following scheme. The square root of the sum of the squares of the elements above the major diagonal (of the original matrix A) is computed first. This is called the initial threshold. A final threshold value of one-millionth of such sum is then established. The diagonalization, which is an iterative process, proceeds up to the stage when the absolute value of every off-diagonal element is less than or equal to the final threshold value.

Since the process is iterative, the user has the option to improve the accuracy of the results by successive CALL-s to the subprogram. For reasons explained in the

documentation (Chapter 2), these successive CALL-s must be to SUBROUTINE IEV (A,S,N). The following rules apply.

i) SUBROUTINE EV (A,S,N) must be CALL-ed once only and before the SUBROUTINE IEV (A,S,N) is CALL-ed.

ii) SUBROUTINE IEV (A,S,N) may be CALL-ed subsequently the required number of times to achieve the desired accuracy. If a total number of n CALL-s are made to (both) the subprograms, each off-diagonal element will be reduced in absolute value to (at least) 10^{-6n} times the initial threshold.

iii) Neither matrix A nor matrix S may be altered in the calling program between any two of the above CALL-s to the subprogram.

Trial runs have indicated that the improvement procedure causes small but significant changes in the eigenvectors, and practically no changes in the eigenvalues (apparently because these are already very close to the exact values). An excessive number of improvement cycles may result in an underflow in the machine.

e. Example:

```
CALL EV (A,S,N)
:
:
CALL IEV (A,S,N)
:
:
CALL IEV (A,S,N)
```

4.5.7 SUBROUTINE MINV (A,N,DET,NEXCH)

a. Function:

The matrix A is inverted in its own space and its determinant is computed.

b. Calling Program:

i) Matrix A is of size N rows by N columns. Vector NEXCH is of size N elements (see item d below).

ii) Matrix A and integer N should be defined.

iii) The inverse of the original matrix A is returned in A itself. The value of the determinant of the original matrix A is returned in DET.

c. Limitations:

The original matrix A is destroyed in the subprogram. See also "Additional Notes" under SUBROUTINE SINV (A,DA,N).

d. Additional Notes:

The vector NEXCH is used for computations only in the subprogram and the values of its elements are of no consequence to the calling program. Hence , the matching vector in the calling program need not necessarily be a vector of integer elements.

3. Examples:

i) CALL MINV (A,N,DET,NEXCH)

ii) CALL MINV (ARRAY,10,DET,TEMP)

4.5.8 SUBROUTINE MOVE (A,B,M,N)

a. Function:

Matrix A is copied as matrix B. (B = A)

b. Calling Program:

i) Matrices A and B are of size M rows by N columns.

ii) Matrix A and integers M and N should be defined.

iii) Matrix B is defined in the subprogram.

c. Additional Notes:

When certain subprograms such as SUBROUTINE DETMT (A,DA,N) of this package are CALL-ed, the original matrices get destroyed in the subprograms. The user may have a need to store the original matrices for further use at a later time. This subprogram meets such a need.

d. Examples:

i) CALL MOVE (A,B,M,N)

ii) CALL MOVE (ARRAY,SAME,10,15)

4.5.9 SUBROUTINE MULT (A,B,C,L,M,N)

a. Function:

Matrix A is post-multiplied by matrix B to yield matrix C. (C = AB)

b. Calling Program:

i) Matrices A, B and C have the following dimensions:

Matrix	Size
A	L rows by M columns
B	M rows by N columns
C	L rows by N columns

ii) Matrices A, B and integers L, M and N should be defined.

iii) The product matrix C is defined in the subprogram.

c. Limitations:

Matrix C should be distinct from matrices A and B. However, matrices A and B may be identical. See also SUBROUTINE PMULT (A,B,K,L,X) and SUBROUTINE POSTM (A,B,K,L,X).

d. Examples:

i) CALL MULT (A,B,C,L,M,N)

ii) CALL MULT (A,A,C,N,N,N)

The examples below yield wrong results:

iii) CALL MULT (A,B,A,L,M,M)

iv) CALL MULT (A,B,B,L,L,N)

v) CALL MULT (A,A,A,N,N,N)

4.5.10 SUBROUTINE OUTE (A,I,J,TITLE,TITEL)

a. Function:

Matrix A of size I rows by J columns is printed. Printing begins on a new page. The matrix is labelled at the top of each page with the labels provided by the user. The word CONTINUED in parantheses appears against the label if printing is on more than one page. The size of the matrix is indicated below the label.

Rows and columns are numbered. On any one page, the

maximum number of rows printed is 25, and the maximum number of columns is 10. Hence, if $J \leq 10$ and $I \leq 25$, printing is completed on one page. If $J > 10$, the first 10 columns are printed, until all the rows (25 or less to a page) are exhuasted. Then the second 10 columns (or less) are printed, until all rows are exhuasted, and so on. The elements of matrix A are output in E-FORMAT. Five digits appear to the right of the decimal point (E12.5).

b. Calling Program:

i) Matrix A is of size I rows by J columns.

ii) Matrix A and integers I and J should be defined. Also, the user's label must be provided as a Hollerith string of characters (maximum 20) through alphanumeric variables or "values" corresponding to the arguments TITLE and TITEL. See examples of CALL statement.

iii) No formal "values" are returned by this subprogram.

c. Additional Notes:

This subprogram is recommended for use in preference to SUBROUTINE OUTF (A,I,J,TITLE,TITEL) whenever the magnitudes of the elements of the matrix to be printed are unknown, unpredictable, or exceed the field F12.5. A slight sacrifice of easy readability is implicit.

d. Examples:

i) CALL OUTE(A,I,J,TITLE,TITEL)

ii) VAR = 10HMATRIX OF
TITLE = 10HREDUNDANTS
CALL OUTE(A,I,J,VAR,TITLE)

iii) CALL OUTE(A,I,J,10HORIGINAL M,5HATRIX)

4.5.11 SUBROUTINE OUTF(A,I,J,TITLE,TITEL)

a. Function:

All the details are the same as in SUBROUTINE OUTE(A,I, J,TITLE,TITEL) except that the elements are output in F-FORMAT. Five digits appear to the right of the decimal point, and a maximum of six digits (five, if the value is negative) appear to its left. (F12.5)

b. Calling Program:

Same as in SUBROUTINE OUTE(A,I,J,TITLE,TITEL)

c. Limitations:

The "largest" numbers that can be printed are of the form abcdef.ghijk or -bcdef.ghijk. If a number "larger" than these is attempted to be printed, an asterisk (*) will appear at the beginning of the corresponding field.

c. Examples:

i) CALL OUTF(A,I,J,TITLE,TITEL)

ii) CALL OUTF(A,I,J,8HMATRIX A,1H)

4.5.12 SUBROUTINE OUTG(A,I,J,TITLE,TITEL)

a. Function:

All the details are the same as in SUBROUTINE OUTE(A,I, J,TITLE,TITEL), except that the elements are output in G-FORMAT.

400.4

b. Calling Program:

Same as in SUBROUTINE OUTE(A,I,J,TITLE,TITEL)

c. Additional Notes:

Five significant digits appear on output, if the absolute value x of the element being printed is in the range

$$0.1 \leq x < 10^5 \quad \text{(G 12.5)}$$

Otherwise, the output is in E-FORMAT for the element.

d. Examples:

i) CALL OUTG(A,I,J,TITLE,TITEL)

ii) CALL OUTG(A,I,J,1H ,1H)

4.5.13 SUBROUTINE PMULT(A,B,K,L,X)

a. Function:

The square matrix B is premultiplied by a rectangular (or square) matrix A and the product matrix is stored in A. A = AB.

b. Calling Program:

i) Matrices A and B, and vector X, have the following dimensions:

Matrix	Size
A	K rows by L columns (may be square, K = L)
B	L rows by L columns
X (Vector)	L elements

ii) Matrices A, B and integers K, L should be defined.

iii) The product matrix is returned in matrix A. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix B must be square. The original matrix A is destroyed. If matrix A is square, it should differ from matrix B at least by name.

d. Examples:

i) CALL PMULT(A,B,K,L,X)

ii) CALL PMULT(A,B,K,K,X)

The example below yields wrong results:

iii) CALL PMULT(A,A,L,L,X)

4.5.14 SUBROUTINE POSTM(A,B,K,L,X)

a. Function:

The square matrix A is postmultiplied by a rectangular (or square) matrix B and the product matrix is stored in B.

B = AB.

b. Calling Program:

i) Matrices A and B, and vector X, have the following dimensions:

Matrix	Size
A	K rows by K columns
B	K rows by L columns (may be square, K = L)
X (Vector)	K elements

ii) Matrices A, B and integers K, L should be defined.

iii) The product matrix is returned in matrix B. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix A must be square. The original matrix B is destroyed. If matrix B is square, it should differ from matrix A at least by name.

d. Examples:

i) CALL POSTM(A,B,K,L,X)

ii) CALL POSTM(A,B,L,L,X)

The example below yields wrong results:

iii) CALL POSTM(A,A,K,K,X)

4.5.15 SUBROUTINE RDCBC(A,M,N)

a. Function:

Elements of matrix A are defined (column by column) by reading in values from data cards.

b. Calling Program:

i) Matrix A is of size M rows by N columns.

ii) Integers M and N should be defined.

The number of data cards required per column of matrix A is (M+7)/8. (Integer division)

iii) Matrix A is defined in the subprogram.

c. Additional Notes:

i) FORMAT Control:

The FORMAT is (8F10.0) and the decimal point should

preferably be punched in each data field. If it is not punched, it will be assumed to be at the end of the field. The non-punch positions in the field are assumed to be filled with zeroes.

For example, if 23.5 is punched beginning in column 21, the value assigned to the corresponding element is the same. If the decimal point is not punched, the value assigned is 2305000000.0.

ii) Order in Assigning Values:

Assume matrix A is of size 14 rows by 6 columns. Two data cards are required per column. Hence, the total number of data cards required is 12. The eight values on the first data card will be assigned in order to $A_{1,1}$, $A_{2,1}$,...,$A_{8,1}$. The six values on the second data card to $A_{9,1}$, $A_{10,1}$,..., $A_{14,1}$. And so on.

d. Examples:

i) CALL RDCBC(A,M,N)

ii) CALL RDCBC(A,14,6)

4.5.16 SUBROUTINE RDCOLG(A,M,N)

a. Function:

Elements of matrix A are defined by reading in values from data cards. The elements are assumed to be in a continuous string of columns of matrix A.

b. Calling Program:

i) Matrix A is of size M rows by N columns.

ii) Integers M and N should be defined.

The number of data cards required is (M*N+7)/8.

(Integer division)

iii) Matrix A is defined in the subprogram.

c. Additional Notes:

i) Same as in SUBROUTINE RDCBC(A,M,N).

ii) Order in Assigning Values:

Assume matrix A is of size 14 rows by 6 columns. Eleven data cards are required. The eight values on the first data card will be assigned in order to $A_{1,1}$, $A_{2,1}$,...,$A_{8,1}$. The first six values on the second card to $A_{9,1}$,$A_{10,1}$,..., $A_{14,1}$. The last two values on the second card to $A_{1,2}$, $A_{2,2}$. The eight values on the third card to $A_{3,2}$, $A_{4,2}$,..., $A_{10,2}$. And so on.

iii) Of the four READ subprograms in this package, this one requires minimum execution time.

d. Examples:

i) CALL RDCOLG(A,M,N)

ii) CALL RDCOLG(A,14,6)

4.5.17 SUBROUTINE RDRBR(A,M,N)

a. Function:

Elements of matrix A are defined (row by row) by

reading in values from data cards.

b. Calling Program:

i) Matrix A is of size M rows by N columns.

ii) Integers M and N should be defined.

The number of data cards required per row of matrix A is (N+7)/8. (Integer division)

iii) Matrix A is defined in the subprogram.

c. Additional Notes:

i) Same as in SUBROUTINE RDCBC(A,M,N)

ii) Order in Assigning Values:

Assume matrix A is of size 14 rows by 6 columns. One data card is required per row. Hence, the total number of data cards required is 14. The six values on card number I (I ranges in value from 1 to 14) are assigned in order to $A_{I,1}$, $A_{I,2}$,..., $A_{I,6}$.

d. Examples:

i) CALL RDRBR(A,M,N)

ii) CALL RDRBR(A,14,6)

4.5.18 SUBROUTINE RDROWG(A,M,N)

a. Function:

Elements of matrix A are defined by reading in values from data cards. The elements are assumed to be in a continuous string of rows of matrix A.

b. Calling Program:

Same as in SUBROUTINE RDCOLG(A,M,N)

c. Additional Notes:

i) Same as in SUBROUTINE RDCBC(A,M,N)

ii) Order in Assigning Values:

Assume matrix A is of size 14 rows by 6 columns. Eleven data cards are required. The first six values on the first data card will be assigned in order to $A_{1,1}$, $A_{1,2}$,..., $A_{1,6}$. The last two values on the first card to $A_{2,1}$, $A_{2,2}$. The first four values on the second card to $A_{2,3}$, $A_{2,4}$, $A_{2,5}$, $A_{2,6}$. The last four values on the second card to $A_{3,1}$, $A_{3,2}$, $A_{3,3}$, $A_{3,4}$. And so on.

d. Examples:

i) CALL RDROWG(A,M,N)

ii) CALL RDROWG(A,14,6)

4.5.19 SUBROUTINE SCMUL(A,M,N,X)

a. Function:

Elements of matrix A are multiplied by the scalar quantity X.

b. Calling Program:

i) Matrix A is of size M rows by N columns.

ii) Matrix A and the scalar multiplier X as well as integers M and N should be defined.

iii) The modified matrix is returned in A itself.

c. Limitations:

The original matrix is destroyed.

d. Examples:

i) CALL SCMUL(A,M,N,X)

ii) REI = 1.0/EI

CALL SCMUL(A,M,N,REI)

iii) CALL SCMUL(A,M,N,1.0/30000.0)

4.5.20 SUBROUTINE SINV(A,DA,N)

a. Function:

The symmetric matrix A (also positive-definite) is inverted in its own space and its determinant is computed.

b. Calling Program:

i) Matrix A is of size N rows by N columns.

ii) Matrix A and integer N should be defined. The subprogram utilizes only the elements on and above the diagonal of the original matrix A. Hence, if so desired, only these elements of matrix A may be defined.

iii) The inverse is returned in matrix A itself. DA stores the value of the determinant of the original matrix A.

c. Limitations:

The original matrix A must be symmetric as well as positive-definite. The original matrix is destroyed.

d. Additional Notes:

The inverse of a symmetric matrix is also symmetric. This property has been utilized in this subprogram, and

hence the resulting inverse of a symmetric matrix will be symmetric when this subprogram is used.

It is possible that the inverse of a symmetric matrix obtained by the use of SUBROUTINE MINV(A,N,DET,NEXCH) is not ideally symmetric because of round-off errors in the machine.

e. Examples:

i) CALL SINV(A,DA,N)

ii) CALL SINV(K,DETK,20)

4.5.21 SUBROUTINE SOLVE(A,B,N,L,DET)

a. Function:

A system of linear simultaneous equations AX = B is solved.

b. Calling Program:

i) Matrices A and B have the following dimensions:

Matrix	Size
Coefficient Matrix A	N rows by N columns
Right-Hand Side Matrix B	N rows by L columns

ii) Matrices A and B as well as integers N and L should be defined.

iii) The solution matrix is returned in B. The value of the determinant of the coefficient matrix A is returned in DET.

c. Limitations:

The original matrices A and B are destroyed.

d. Examples:

i) CALL SOLVE(A,B,N,L,DET)

ii) CALL SOLVE(COEFF,RHS,N,1,DA)

4.5.22 SUBROUTINE SQTR(A,N)

a. Function:

The transpose of the square matrix A is returned to the calling program in A itself.

b. Calling Program:

i) Matrix A is of size N rows by N columns.

ii) Matrix A and integer N should be defined.

iii) The transposed matrix is returned in A itself.

c. Limitations:

Matrix A must be square. The original matrix is destroyed.

d. Examples:

i) CALL SQTR(A,N)

ii) CALL SQTR(ASQ,10)

4.5.23 SUBROUTINE SUB(A,B,C,M,N)

a. Function:

Subtract matrix B from matrix A and store the result in matrix C. (C = A -B)

b. Calling Program:

i) Matrices A, B and C are each of size M rows by N columns.

ii) Matrices A and B, as well as integers M and N should be defined.

iii) Matrix C is defined in the subprogram.

c. Additional Notes:

The resultant matrix may be stored in one of the original matrices. Only in such a case, the specific original matrix will be destroyed.

d. Examples:

i) CALL SUB(A,B,C,M,N)

ii) CALL SUB(A,B,C,10,15)

iii) CALL SUB(A,B,B,25,30)

4.5.24 SUBROUTINE TMULT(A,B,C,L,M,N)

a. Function:

The transpose of matrix A is postmultiplied by matrix B to give matrix C. ($C = A^TB$).

b. Calling Program:

i) Matrices A, B and C have the following dimensions:

Matrix	Size
A	L rows by M columns
B	L rows by N columns
C	M rows by N columns

ii) Matrices A and B, as well as integers L, M and N, should be defined.

iii) The product matrix is returned in matrix C.

c. Limitations:

Matrix C should be distinct from matrices A and B.

However, matrices A and B may be identical.

d. Examples:

i) CALL TMULT(A,B,C,L,M,N)

ii) CALL TMULT(A,A,B,L,M,M)

iii) CALL TMULT(A,A,B,N,N,N)

The examples below yield wrong results:

iv) CALL TMULT(A,B,A,L,L,L)

v) CALL TMULT(A,B,B,M,M,M)

vi) CALL TMULT(A,A,A,N,N,N)

4.5.25 SUBROUTINE TRANS(A,B,M,N)

a. Function:

Matrix A is transposed to give matrix B. ($B = A^T$)

b. Calling Program:

i) Matrix A is of size M rows by N columns.
Matrix B is of size N rows by M columns.

ii) Matrix A and integers M and N should be defined.

iii) Matrix B is defined in the subprogram.

c. Limitations:

Even if matrix A is square, it should be distinct from the (square) matrix B.

d. Additional Notes:

If matrix A is square and the original matrix may be destroyed, SUBROUTINE SQTR(A,N) should preferably be used.

e. Examples:

i) CALL TRANS(A,B,M,N)

ii) CALL TRANS(A,B,10,15)

iii) CALL TRANS(A,B,N,N)

The example below yields wrong results:

iv) CALL TRANS(A,A,N,N)

4.5.26 SUBROUTINE XABATC(A,B,C,L,M,X)

a. Function:

The matrix product ABA^T is formed in matrix C. Matrix B is symmetric. $C = ABA^T$

b. Calling Program:

i) Matrices A,B,C and vector X have the following dimensions:

Matrix	Size
A	L rows by M columns
B	M rows by M columns
C	L rows by L columns
X(vector)	M elements

ii) Matrices A, B and integers L, M should be defined.

iii) The product matrix (symmetric) is returned in matrix C. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix B must be symmetric.

Matrix C should be distinct from matrices A and B.

d. Examples:

i) CALL XABATC(A,B,C,L,M,X)

ii) CALL XABATC(A,B,C,L,L,X)

The examples below yield wrong results:

iii) CALL XABATC(A,B,B,L,L,X)

iv) CALL XABATC(A,B,A,L,L,X)

v) CALL XABATC(A,A,A,L,L,X)

4.5.27 SUBROUTINE XABTA(A,B,L,N,X)

a. Function:

The matrix product AB^T is formed in matrix A. Matrix B is square. $A = AB^T$.

b. Calling Program:

i) Matrices A, B and vector X have the following dimensions:

Matrix:	Size
A	L rows by N columns
B	N rows by N columns
X(Vector)	N elements

ii) Matrices A, B and integers L, N should be defined.

iii) The product matrix is returned in matrix A. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix B must be square. The original matrices A and B should be distinct from each other even if matrix A is square. The original matrix A is destroyed.

d. Examples:

i) CALL XABTA(A,B,L,N,X)

ii) CALL XABTA(A,B,N,N,X)

The example below yields wrong results:

iii) CALL XABTA(A,A,N,N,X)

4.5.28 SUBROUTINE XABTC(A,B,C,L,M,N)

a. Function:

The matrix product AB^T is formed in matrix C. $C = AB^T$

b. Calling Program:

i) Matrices A, B and C have the following dimensions:

Matrix	Size
A	L rows by M columns
B	N rows by M columns
C	L rows by N columns

ii) Matrices A, B and integers L, M and N should be defined.

iii) The product matrix C is defined in the subprogram.

c. Limitations:

Matrix C should be distinct from matrices A and B.

d. Examples:

i) CALL XABTC(A,B,C,L,M,N)

ii) CALL XABTC(A,A,C,L,M,L)

iii) CALL XABTC(A,A,C,L,L,L)

The examples below yield wrong results:

iv) CALL XABTC(A,B,A,L,M,M)

v) CALL XABTC(A,B,B,L,L,L)

vi) CALL XABTC(A,A,A,L,L,L)

4.5.29 SUBROUTINE XATBAC(A,B,C,L,M,X)

a. Function:

The matrix product A^TBA is formed in matrix C. Matrix B is symmetric. $C = A^TBA$

b. Calling Program:

i) Matrices A, B and C and vector X have the following dimensions:

Matrix	Size
A	L rows by M columns
B	L rows by L columns
C	M rows by M columns
X(Vector)	L elements

ii) Matrices A, B and integers L, M should be defined.

iii) The product matrix C (symmetric) is defined in the subprogram. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix B must be symmetric. Matrix C should be distinct from matrices A and B.

d. Examples:

i) CALL XATBAC(A,B,C,L,M,X)

ii) CALL XATBAC(A,B,C,L,L,X)

The examples below yield wrong results:

iii) CALL XATBAC(A,B,B,L,L,X)

iv) CALL XATBAC(A,B,A,L,L,X)

v) CALL XATBAC(A,A,A,L,L,X)

4.5.30 SUBROUTINE XATBB(A,B,L,N,X)

a. Function:

The matrix product A^TB is formed in matrix B. Matrix A is square. $B = A^TB$.

b. Calling Program:

i) Matrices A, B and vector X have the following dimensions:

Matrix	Size
A	L rows by L columns
B	L rows by N columns
X(Vector)	L elements

ii) Matrices A, B and integers L, N should be defined.

iii) The product matrix is returned in matrix B. Vector X is required in the subprogram for computations only.

c. Limitations:

Matrix A must be square. The original matrix B is destroyed. The original matrices A and B should be distinct from each other even if matrix B is square.

d. Examples:

i) CALL XATBB(A,B,L,N,X)

ii) CALL XATBB(A,B,N,N,X)

The example below yields wrong results:

iii) CALL XATBB(A,A,N,N,X)

5. READY REFERENCE SHEET

After a few problems have been solved with the use of this package, the user will have gained enough experience to operate within its requirements and limitations, and frequent references to the text become unnecessary in day-to-day usage. All the same, matching the argument list of each subprogram and dimensioning arrays suitably are problems which may require some memory aid. A ready reference sheet has, therefore, been provided. The subprograms are listed alphabetically.

READY REFERENCE SHEET (FLMXPK)

NO.	SUBPROGRAM	BRIEF DESCRIPTION	DIMENSIONS	NOTES
1.	ADD(A,B,C,M,N)	C = A + B	A(M,N),B(M,N),C(M,N)	
2.	DETMT(A,DA,N)	Determinant	A(N,N)	
3.	DIAG(A,DA,N)	Diagonal matrix	A(N,N)	
4.	EV(A,S,N)	E-values,E-vectors	A(N,N),S(N,N)	
5.	GEVP(A,B,S,T,N)	Solve AX = λBX	A(N,N),B(N,N),S(N,N),T(N)	
6.	IEV(A,S,N)	Improve on EV	A(N,N),S(N,N)	
7.	MINV(A,N,DET,NEXCH)	Invert A	A(N,N),NEXCH(N)	
8.	MOVE(A,B,M,N)	B = A	A(M,N),B(M,N)	
9.	MULT(A,B,C,L,M,N)	C = AB	A(L,M),B(M,N),C(L,N)	
10.	OUTE(A,I,J,TITLE,TITEL)	Print A	A(I,J)	
11.	OUTF(A,I,J,TITLE,TITEL)	Print A	A(I,J)	
12.	OUTG(A,I,J,TITLE,TITEL)	Print A	A(I,J)	
13.	PMULT(A,B,K,L,X)	A = AB	A(K,L),B(L,L),X(L)	
14.	POSTM(A,B,K,L,X)	B = AB	A(K,K),B(K,L),X(K)	
15.	RDCBC(A,M,N)	Read col. by col.	A(M,N)	
16.	RDCOLG(A,M,N)	Read columns (grouped)	A(M,N)	
17.	RDRBR(A,M,N)	Read row by row	A(M,N)	
18.	RDROWG(A,M,N)	Read rows (grouped)	A(M,N)	
19.	SCMUL(A,M,N,X)	A = xA	A(M,N)	
20.	SINV(A,DA,N)	Sym. inversion	A(N,N)	
21.	SOLVE(A,B,N,L,DET)	Sim. Eqs. (AX = B)	A(N,N),B(N,L)	
22.	SQTR(A,N)	Transpose in place	A(N,N)	
23.	SUB(A,B,C,M,N)	C = A - B	A(M,N),B(M,N),C(M,N)	
24.	TMULT(A,B,C,L,M,N)	C = A^TB	A(L,M),B(L,N),C(M,N)	
25.	TRANS(A,B,M,N)	B = A^T	A(M,N),B(N,M)	
26.	XABATC(A,B,C,L,M,X)	C = ABA^T	A(L,M),B(M,M),C(L,L),X(M)	
27.	XABTA(A,B,L,N,X)	A = AB^T	A(L,N),B(N,N),X(N)	
28.	XABTC(A,B,C,L,M,N)	C = AB^T	A(L,M),B(N,M),C(L,N)	
29.	XATBAC(A,B,C,L,M,X)	C = A^TBA	A(L,M),B(L,L),C(M,M),X(L)	
30.	XATBB(A,B,L,N,X)	B = A^TB	A(L,L),B(L,N),X(L)	

APPENDIX 1

Proofs of Some Basic Theorems

APPENDIX 1

In support of the relations which were used earlier in the mathematical derivations, proof of some basic theorems in matrix algebra have been included in this Appendix for ready reference. For the sake of brevity, the proofs are given in only a few lines. A standard text, such as Reference 4, may be consulted for more detailed information. The algebraic derivation of sin θ and cos θ from tan 2θ is also included.

1. The transpose of a product is the product of the transposes in reverse order.

Let $C = AB$, $X = A^T$, $Y = B^T$ and $Z = YX = B^TA^T$.

$$C_{I,J} = \sum_{K=1}^{N} A_{I,K}B_{K,J} = \sum_{K=1}^{N} X_{K,I}Y_{J,K} = \sum_{K=1}^{N} Y_{J,K}X_{K,I} = Z_{J,I}$$

Hence, $C^T = Z$ or $(AB)^T = B^TA^T$

Extending the proof,

$$(ABC)^T = [A(BC)]^T = (BC)^TA^T = C^TB^TA^T$$

2. The inverse of A exists iff (if and only if) the determinant of A is non-zero.

Some authors define and others show that $A^{-1} = \frac{1}{\det(A)}$ [co-factor], where det(A) = determinant of A. If $\det(A) \neq 0$, the Right-Hand Side of this equation exists. Hence, A^{-1} exists.

If A^{-1} exists, the Left-Hand Side exists and, hence, $\det(A) \neq 0$.

3. A^{-1} is unique.

If B and C are both inverses of A,

$AB(=BA)=I$ and $AC(=CA)=I$,

and hence $A(B-C) = 0$, the null matrix.

Premultiply by B or C.

$B - C = 0$

Hence, $B = C$.

4. The determinant of the product is the product of the determinants.

Consider 2 x 2 matrices.

$$|A| \cdot |B| = \begin{vmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{vmatrix} \cdot \begin{vmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{vmatrix}$$

This is the same as $\begin{vmatrix} A_{1,1} & A_{1,2} & 0 & 0 \\ A_{2,1} & A_{2,2} & 0 & 0 \\ -1 & 0 & B_{1,1} & B_{1,2} \\ 0 & -1 & B_{2,1} & B_{2,2} \end{vmatrix}$,

since, by Laplace expansion, using the first and second rows, the result is

$$(-1)^{1+2+1+2}|A| \cdot |B| = |A| \cdot |B|$$

Add $B_{1,1}$ times first column to the third, $B_{2,2}$ times second column to the fourth, and then add $B_{1,2}$ times

first column to the fourth, $B_{2,1}$ times second column to the third. The result is

$$\begin{vmatrix} A_{1,1} & A_{1,2} & A_{1,1}B_{1,1}+A_{1,2}B_{2,1} & A_{1,1}B_{1,2}+A_{1,2}B_{2,2} \\ A_{2,1} & A_{2,2} & A_{2,1}B_{1,1}+A_{2,2}B_{2,1} & A_{2,1}B_{1,2}+A_{2,2}B_{2,2} \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{vmatrix}$$

Again, by Laplace expansion, using the third and the fourth rows, this is equal to $|AB|$.

The proof may intuitively be extended to matrices larger than 2 x 2.

5. <u>The determinant of the inverse is the reciprocal of the determinant of the original matrix</u>.

$AA^{-1} = I$, hence $\det(A)\cdot\det(A^{-1}) = \det(I) = 1$. This also shows that, if $\det(A) = 0$, $\det(A^{-1}) = \infty$, or the elements of A^{-1} are undefined.

6. <u>The inverse of a product is the product of the inverses in reverse order</u>.

By definition, $(AB)(AB)^{-1} = I$

Premultiply by A^{-1}, $B(AB)^{-1} = A^{-1}$

Premultiply by B^{-1}, $(AB)^{-1} = B^{-1}A^{-1}$

Extending the proof,

$$(ABC)^{-1} = [A(BC)]^{-1} = (BC)^{-1}A^{-1} = C^{-1}B^{-1}A^{-1}$$

7. The inverse of the transpose is the transpose of the inverse.

By definition, $AA^{-1} = I$

By transpose, $(A^{-1})^T A^T = I^T = I$

Hence, $(A^{-1})^T = (A^T)^{-1}$

8. The inverse of a symmetric matrix is symmetric

$(A^{-1})^T = (A^T)^{-1}$

Since $A^T = A$, $(A^{-1})^T = A^{-1}$

9. The product of symmetric matrices is symmetric iff the matrices commute with each other.

By definition, A and B commute if $AB = BA$.

$$(AB)^T = B^T A^T$$

By symmetry of A and B, $(AB)^T = BA$

If A and B commute, $(AB)^T = AB$

Also (only if part),

if $AB = (AB)^T$, $AB = B^T A^T = BA$ or A and B commute

10. The product of a matrix and its transpose is symmetric.

$$(AA^T)^T = (A^T)^T A^T = AA^T$$

Note that A need not be square, and in general,

$AA^T \neq A^T A$

11. The product RT is orthonormal, if R and T are othonormal.

$$RT(RT)^T = RTT^T R^T$$

Since T is orthonormal, $RT(RT)^T = RR^T$

Since R is orthonormal, $RT(RT)^T = I$

Hence, $(RT)^T = (RT)^{-1}$

12. A symmetric matrix A may be expressed in the form $A = \lambda\lambda^T$ where λ is lower triangular. Find elements of λ.

For $I < J$, $\lambda_{I,J} = 0$

For $I \geq J$,

$$A_{I,J} = \sum_{K=1}^{N} \lambda_{I,K}\lambda_{J,K} = \lambda_{I,1}\lambda_{J,1} + \lambda_{1,2}\lambda_{J,2} + \ldots + \lambda_{I,J-1}\lambda_{J,J-1}$$

$$+ \lambda_{I,J}\lambda_{J,J} + \lambda_{I,J+1}\lambda_{J,J+1} + \ldots + \lambda_{I,N}\lambda_{J,N}$$

Since $\lambda_{L,M} = 0$ if $L < M$,

$$A_{I,J} = \sum_{K=1}^{J-1} \lambda_{I,K}\lambda_{J,K} + \lambda_{I,J}\lambda_{J,J}$$

If $I = J$, $\lambda_{I,I} = \left(A_{I,I} - \sum_{K=1}^{I-1} \lambda_{I,K}^2\right)^{\frac{1}{2}}$

If $I > J$, $\lambda_{I,J} = \frac{1}{\lambda_{J,J}} \left(A_{I,J} - \sum_{K=1}^{J-1} \lambda_{I,K}\lambda_{J,K}\right)$

13. A^{-1} is lower-triangular if A is lower-triangular. Find elements of A^{-1}.

By definition, $AB = I$ where $B = A^{-1}$.

$$\sum_{K=1}^{N} A_{I,K}B_{K,J} = \delta_{I,J} \quad \text{(Kronecker Delta)}$$

$$\delta_{I,J} = A_{I,1}B_{1,J} + A_{I,2}B_{2,J} + \ldots + A_{I,I}B_{I,J} + A_{I,I+1}B_{I+1,J} +$$

$$\ldots + A_{I,N}B_{N,J}$$

Since $A_{L,M} = 0$ for $L < M$,

$$\delta_{I,J} = \sum_{K=1}^{I} A_{I,K} B_{K,J}$$

If $I = 1$, and $J > I$, $\delta_{I,J} = 0$

$$0 = A_{1,1} B_{1,J} \text{ , hence, } B_{1,J} = 0$$

If $I = 2$, and $J > I$, $\delta_{I,J} = 0$

$$0 = A_{2,1} B_{1,J} + A_{2,2} B_{2,J} = 0 + A_{2,2} B_{2,J}, \text{ hence, } B_{2,J} = 0$$

Similarly, for all $J > I$, $B_{I,J} = 0$

Hence, $B = A^{-1}$ is lower-triangular.

Non-zero elements of B.

If $I = J$, $\delta_{I,J} = 1$

$1 = \sum_{K=1}^{I} A_{I,K} B_{K,I} = A_{I,I} B_{I,I}$, since both A and B are lower-triangular. Hence, $B_{I,I} = \frac{1}{A_{I,I}}$

If $I > J$, $\delta_{I,J} = 0$

$$0 = A_{I,1} B_{1,J} + A_{I,2} B_{2,J} + \ldots + A_{I,J} B_{J,J} + \ldots + A_{I,I} B_{I,J}$$

$B_{L,M} = 0$ if $L < M$. Hence,

$$0 = A_{I,J} B_{J,J} + \ldots + A_{I,I-1} B_{I-1,J} + A_{I,I} B_{I,J}$$

Hence, $B_{I,J} = -\frac{1}{A_{I,I}} \left(\sum_{K=J}^{I-1} A_{I,K} B_{K,J} \right)$ for $I > J$.

14. Given tan 2θ, find sin θ and cos θ.

The value of tan 2θ is given by $\tan 2\theta = \dfrac{-A_{P,Q}}{0.5(A_{P,P} - A_{Q,Q})}$

This problem arises in SUBROUTINE EV(A,S,N).

Let λ = Numerator and μ = Denominator.

Since $\tan\alpha = \tan(\alpha - \pi)$, there are two solutions for 2θ in the range $-\pi$ to π. Since the solutions are supplements of each other, a unique solution is available if the range is chosen as $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. The range of θ is then $-\frac{\pi}{4}$ to $\frac{\pi}{4}$.

$$\tan 2\theta = \frac{\sin 2\theta}{\cos 2\theta} = \frac{2 \sin\theta \sqrt{1 - \sin^2\theta}}{1 - 2\sin^2\theta} = \frac{\lambda}{\mu}$$

By squaring, rearrangement and the substitution $x = \sin^2\theta$,

$$4x^2\left(1 + \frac{\lambda^2}{\mu^2}\right) - 4x\left(1 + \frac{\lambda^2}{\mu^2}\right) + \frac{\lambda^2}{\mu^2} = 0$$

$$\therefore\ x = 0.5\left[1 \pm \sqrt{1 - \frac{\lambda^2}{\lambda^2+\mu^2}}\right]$$

Since $-\frac{\pi}{4} \le \theta \le \frac{\pi}{4}$, $\sin^2\theta$ or x has a maximum value of 1/2.

$$\text{Hence, } x = 0.5\left(1 - \sqrt{1 - \frac{\lambda^2}{\lambda^2+\mu^2}}\right) = 0.5\left[\frac{\lambda^2/(\lambda^2+\mu^2)}{1 + \sqrt{1 - \lambda^2/(\lambda^2+\mu^2)}}\right]$$

If $\omega^2 = \dfrac{\lambda^2}{\lambda^2+\mu^2}$, $\sin\theta = \dfrac{\pm\,\omega}{\sqrt{2(1+\sqrt{1-\omega^2})}}$

The signs of sinθ and tan 2θ in the range chosen for θ are the same. The sign of tan 2θ is governed by the signs of λ and μ. The proper value of sinθ will therefore be available with the definitions,

$$\omega = (\text{sign of } \mu)\ \frac{\lambda}{\sqrt{\lambda^2 + \mu^2}},\quad \sin\theta = \frac{\omega}{\sqrt{2(1 + \sqrt{1 - \omega^2})}}$$

The value of $\cos\theta$ in the range is positive.

$$\cos\theta = \sqrt{1 - \sin^2\theta}$$

The advantages of this formulation are these:

1. $\sin\theta$ and $\cos\theta$ may be computed algebraically.
2. Even if $\mu = 0$ (and hence $\tan 2\theta$ is infinity), the computations for $\sin\theta$ and $\cos\theta$ can be carried out without difficulty in the computer.

15. When the Identity Matrix is available by implication only, the column exchanges, to match previous row exchanges during reduction, must be performed in reverse order. [See SUBROUTINE MINV(A,N,DET,NEXCH)].

With the availability of the I-Matrix, this problem does not arise as rows of I can also be interchanged during the process itself.

When the I-matrix is not available, the Kth column of I is available by implication only. When rows of A are interchanged, the net effect on I is that the corresponding rows and columns of I are interchanged. Thus, if X_1 is the transformation matrix corresponding to the row interchange, we have

$$T_1 X_1 T \left| A \;\vdots\; IX_1 \right|$$

at an intermediate stage of the reduction process and

$$T_M T_{M-1} \cdots T_P X_P \cdots T_2 X_2 T_1 X_1 T \left| A \,\vdots\, I X_1 X_2 \cdots X_P \right|$$

at the final stage.

In short, $S\left|A \,\vdots\, IX_1X_2 \cdots X_P\right| = \left|I \,\vdots\, C^{-1}\right|$

Thus, $SA = I$ or $S = A^{-1}$

and $A^{-1}X_1X_2 \cdots X_P = C^{-1}$

or $A^{-1} = C^{-1}X_P^{-1} \cdots X_2^{-1}X_1^{-1}$

$= C^{-1}X_P \cdots X_2X_1$

APPENDIX 2

Listing of Subprograms

```
      SUBROUTINE ADD (A,B,C,M,N)                                        1
C                                                                       2
C     AUTHOR - SAMPATH IYENGAR, FRITZ ENGINEERING LABORATORY,           3
C     LEHIGH UNIVERSITY, BETHLEHEM, PA. 18015.                          4
C                                                                       5
C     THIS GROUP OF SUBPROGRAMS (FLMXPK - FRITZ LABORATORY MATRIX       6
C     PACKAGE) CONTAINS 30 ROUTINES TO PERFORM MATRIX OPERATIONS.       7
C     FOR DETAILED DOCUMENTATION, SEE FLMXPK - A MATRIX PACKAGE,        8
C     FRITZ LABORATORY REPORT NO. 400.4.                                9
C                                                                      10
      COMMON /IYENGAR/ I,MN,Y(12)                                      11
      REAL A(1),B(1),C(1)                                              12
C                                                                      13
C     ROUTINE TO ADD MATRICES A AND B. C=A+B                           14
C                                                                      15
      MN=M*N                                                           16
      DO 1000 I=1,MN                                                   17
 1000 C(I)=A(I)+B(I)                                                   18
      RETURN                                                           19
      END                                                              20
-----------------------------------------------------------------------
      SUBROUTINE DETMT (A,DA,N)                                         1
C                                                                       2
C     ROUTINE TO FIND THE DETERMINANT DA OF MATRIX A.                   3
C                                                                       4
      COMMON /IYENGAR/ I,IP1,J,M,NM1,T,TEMP,XX,Y(6)                     5
      REAL A(N,N)                                                       6
C                                                                       7
      DA=1.0                                                            8
      IF (N.EQ.1) GO TO 1050                                            9
      NM1=N-1                                                          10
      DO 1040 I=1,NM1                                                  11
      IP1=I+1                                                          12
C                                                                      13
C     SEARCH ROW I FOR THE LARGEST ABSOLUTE-VALUED                     14
C     ELEMENT IN COLUMNS I THROUGH N.                                  15
C                                                                      16
      TEMP=0.0                                                         17
      DO 1000 J=I,N                                                    18
      T=ABS(A(I,J))                                                    19
      IF (TEMP.GE.T) GO TO 1000                                        20
      TEMP=T                                                           21
C                                                                      22
C     THE LARGEST ABSOLUTE-VALUED ELEMENT IS IN COLUMN M OF ROW I.     23
C                                                                      24
      M=J                                                              25
 1000 CONTINUE                                                         26
      IF (TEMP.NE.0.0) GO TO 1010                                      27
C                                                                      28
C     ALL THE ELEMENTS OF ROW I ARE IDENTICALLY ZERO.                  29
C     DETERMINANT VANISHES.                                            30
C                                                                      31
      DA=0.0                                                           32
      RETURN                                                           33
C                                                                      34
C     EXCHANGE COLUMNS M AND I, ONLY IF M IS DIFFERENT FROM I.         35
C                                                                      36
 1010 IF (M.EQ.I) GO TO 1030                                           37
```

```
      DO 1020 J=I,N                                                      36
      T=A(J,M)                                                           39
      A(J,M)=A(J,I)                                                      40
 1020 A(J,I)=T                                                           41
C                                                                        42
C     CHANGE THE SIGN OF THE DETERMINANT,                                43
C     SINCE COLUMNS HAVE BEEN EXCHANGED.                                 44
C                                                                        45
      DA=-DA                                                             46
 1030 TEMP=A(I,I)                                                        47
C                                                                        48
C     CONTINUOUS PART-PRODUCT FOR THE DETERMINANT.                       49
C                                                                        50
      DA=DA*TEMP                                                         51
C                                                                        52
C     THE PROCESS OF REDUCTION TO AN UPPER-TRIANGULAR MATRIX.            53
C                                                                        54
      TEMP=1.0/TEMP                                                      55
      DO 1040 M=IP1,N                                                    56
      XX=-A(M,I)*TEMP                                                    57
      DO 1040 J=IP1,N                                                    58
 1040 A(M,J)=A(M,J)+XX*A(I,J)                                            59
C                                                                        60
C     FINAL VALUE OF THE DETERMINANT.                                    61
C                                                                        62
 1050 DA=DA*A(N,N)                                                       63
      RETURN                                                             64
      END                                                                65
```

```
      SUBROUTINE DIAG (A,DA,N)                                            1
C                                                                         2
C     ROUTINE TO CREATE A DIAGONAL MATRIX A.                              3
C                                                                         4
      COMMON /IYENGAR/ I,NP1,NSQ,Y(11)                                    5
      REAL A(1)                                                           6
C                                                                         7
C     ZERO THE ARRAY.                                                     8
C                                                                         9
      NSQ=N*N                                                            10
      DO 1000 I=1,NSQ                                                    11
 1000 A(I)=0.0                                                           12
C                                                                        13
C     ASSIGN DA TO EACH DIAGONAL ELEMENT.                                14
C                                                                        15
      NP1=N+1                                                            16
      DO 1010 I=1,NSQ,NP1                                                17
 1010 A(I)=DA                                                            18
      RETURN                                                             19
      END                                                                20
```

```
      SUBROUTINE EV (A,S,N)
C
C     ROUTINE TO FIND EIGENVALUES AND EIGENVECTORS
C     OF THE SYMMETRIC MATRIX A.
C
      COMMON /IYENGAR/ AIP,AIQ,APP,AQQ,COSSQ,COST,I,P,Q,QM1,SINSQ,SINT,T
     1EMP,TSCT
      EQUIVALENCE (AIP,SIP,Y), (AIQ,SIQ), (P,IP1), (Q,J), (QM1,NM1), (TE
     1MP,SIGN), (THRI,SUM), (TSCT,X)
      LOGICAL IND
      REAL A(N,N),S(N,N)
      INTEGER P,Q,QM1
C
      IF (N.EQ.1) GO TO 1010
C
C     CALCULATE THRI, THE INITIAL THRESHOLD VALUE.
C     INITIALIZE MATRIX S AS THE IDENTITY MATRIX.
C
      SUM=0.0
      NM1=N-1
      DO 1000 I=1,NM1
      S(I,I)=1.0
      IP1=I+1
      DO 1000 J=IP1,N
      TEMP=A(I,J)
      S(I,J)=0.0
      S(J,I)=0.0
 1000 SUM=SUM+TEMP*TEMP
      THRI=SQRT(SUM)
C
C     SET THE DESIRED ACCURACY (THRF).
C
      THRF=THRI*1.0E-6
      FN=N
C
C     INITIALIZE INDICATOR IND.
C
      IND=.TRUE.
 1010 S(N,N)=1.0
C
      ENTRY IEV
C
C     SUBROUTINE IEV (A,S,N)
C     ROUTINE TO IMPROVE EIGENVALUES AND EIGENVECTORS.
C
      IF (N.EQ.1) RETURN
      IF (THRI.EQ.0.0) RETURN
C
C     LOWER THE INITIAL THRESHOLD THRI,
C     SUCCESSIVELY BY DIVISION BY N.
C
 1020 THRI=THRI/FN
C
C     ITERATION BEGINS HERE.
C
 1030 DO 1050 Q=2,N
      QM1=Q-1
      DO 1050 P=1,QM1
```

```
C                                                                       59
C     TEST EACH OFF-DIAGONAL ELEMENT IN TURN.                           60
C                                                                       61
      X=-A(P,Q)                                                         62
      IF (ABS(X).LT.THRI) GO TO 1050                                    63
C                                                                       64
C     THE OFF-DIAGONAL ELEMENT IS NOT LESS THAN THRI IN ABSOLUTE VALUE. 65
C     ANNIHILATE IT. RESET INDICATOR, SINCE ANNIHILATION IS NECESSARY.  66
C                                                                       67
      IND=.FALSE.                                                       68
C                                                                       69
C     CALCULATE TRIGONOMETRIC FUNCTIONS.                                70
C                                                                       71
      APP=A(P,P)                                                        72
      AQQ=A(Q,Q)                                                        73
      Y=0.5*(APP-AQQ)                                                   74
      SIGN=1.0                                                          75
      IF (Y.LT.0.0) SIGN=-1.0                                           76
      TEMP=SIGN*X/SQRT(X*X+Y*Y)                                         77
      SINT=TEMP/SQRT(2.0*(1.0+SQRT(1.0-TEMP*TEMP)))                     78
      SINSQ=SINT*SINT                                                   79
      COSSQ=1.0-SINSQ                                                   80
      COST=SQRT(COSSQ)                                                  81
      TSCT=-2.0*X*SINT*COST                                             82
C                                                                       83
C     MODIFY ELEMENTS OF MATRICES A AND S ROW BY ROW.                   84
C     ALSO, TRANSPOSE TO MODIFY CORRESPONDING ROWS OF A.                85
C                                                                       86
      DO 1040 I=1,N                                                     87
      AIP=A(I,P)                                                        88
      AIQ=A(I,Q)                                                        89
      TEMP=AIP*COST-AIQ*SINT                                            90
      A(I,P)=TEMP                                                       91
      A(P,I)=TEMP                                                       92
      TEMP=AIP*SINT+AIQ*COST                                            93
      A(I,Q)=TEMP                                                       94
      A(Q,I)=TEMP                                                       95
      SIP=S(I,P)                                                        96
      SIQ=S(I,Q)                                                        97
      S(I,P)=SIP*COST-SIQ*SINT                                          98
 1040 S(I,Q)=SIP*SINT+SIQ*COST                                          99
C                                                                      100
C     EVALUATE FRESH ELEMENTS OF THE PIVOTAL SET.                      101
C                                                                      102
      A(P,P)=APP*COSSQ+AQQ*SINSQ-TSCT                                  103
      A(Q,Q)=AQQ*COSSQ+APP*SINSQ+TSCT                                  104
      A(P,Q)=0.0                                                       105
      A(Q,P)=0.0                                                       106
 1050 CONTINUE                                                         107
C                                                                      108
C     CHECK WHETHER A REPEAT SWEEP IS NECESSARY.                       109
C                                                                      110
      IF (IND) GO TO 1060                                              111
C                                                                      112
C     A REPEAT SWEEP IS NECESSARY.                                     113
C                                                                      114
      IND=.TRUE.                                                       115
      GO TO 1030                                                       116
```

```
C                                                                       117
C     CHECK WHETHER THE DESIRED ACCURACY IS ACHIEVED.                   118
C                                                                       119
 1060 IF (THRI.GE.THRF) GO TO 1020                                      120
C                                                                       121
C     RESET THRF TO BE IN READINESS FOR A                               122
C     SUBSEQUENT IMPROVEMENT PROCEDURE, IF ANY.                         123
C                                                                       124
      THRF=THRF*1.0E-6                                                  125
      RETURN                                                            126
      END                                                               127
```

```
      SUBROUTINE GEVP (A,B,S,T,N)                                         1
C                                                                         2
C     ROUTINE TO SOLVE THE GENERAL EIGENVALUE PROBLEM.                    3
C     A*X=LAMBDA*B*X                                                      4
C                                                                         5
      COMMON /IYENGAR/ I,J,K,SUM,TEMP,Y(9)                                6
      REAL A(N,N),B(N,N),S(N,N),T(N)                                      7
C                                                                         8
C     DIAGONALIZE MATRIX B.                                               9
C                                                                        10
      CALL EV (B,S,N)                                                    11
C                                                                        12
C     IMPROVEMENT PROCEDURE.                                             13
C                                                                        14
      CALL IEV (B,S,N)                                                   15
C                                                                        16
C     FORM THE MATRIX PRODUCT S*INVERSE OF G IN S.                       17
C                                                                        18
      DO 1000 J=1,N                                                      19
      TEMP=1.0/SQRT(B(J,J))                                              20
      DO 1000 I=1,N                                                      21
 1000 S(I,J)=S(I,J)*TEMP                                                 22
C                                                                        23
C     MODIFY MATRIX A.                                                   24
C                                                                        25
      CALL MULT (A,S,B,N,N,N)                                            26
      DO 1020 I=1,N                                                      27
      DO 1020 J=I,N                                                      28
      SUM=0.0                                                            29
      DO 1010 K=1,N                                                      30
 1010 SUM=SUM+S(K,I)*B(K,J)                                              31
      A(I,J)=SUM                                                         32
 1020 A(J,I)=SUM                                                         33
C                                                                        34
C     OBTAIN EIGENVALUES OF A.                                           35
C                                                                        36
      CALL EV (A,B,N)                                                    37
      CALL IEV (A,B,N)                                                   38
C                                                                        39
C     FORM THE MATRIX PRODUCT S*EIGENVECTOR MATRIX                       40
C     TO OBTAIN THE FINAL EIGENVECTORS.                                  41
C                                                                        42
      CALL POSTM (S,B,N,N,T)                                             43
      RETURN                                                             44
      END                                                                45
```

```
      SUBROUTINE MINV (A,N,DET,NEXCH)                                    1
C                                                                        2
C     ROUTINE TO INVERT MATRIX A.                                        3
C                                                                        4
      COMMON /IYENGAR/ BIG,I,J,K,NM1,NP1,NR,T,Y(6)                       5
      REAL A(N,N)                                                        6
      INTEGER NEXCH(N)                                                   7
C                                                                        8
      IF (N.NE.1) GO TO 1000                                             9
      DET=A(1,1)                                                        10
      A(1,1)=1.0/DET                                                    11
      RETURN                                                            12
 1000 DET=1.0                                                           13
      NM1=N-1                                                           14
      DO 1070 K=1,N                                                     15
C                                                                       16
C     SEARCH COLUMN 1 FOR THE LARGEST ABSOLUTE-VALUED                   17
C     ELEMENT IN ROWS K THROUGH N.                                      18
C                                                                       19
      BIG=0.0                                                           20
      DO 1010 I=K,N                                                     21
      T=ABS(A(I,1))                                                     22
      IF (BIG.GE.T) GO TO 1010                                          23
      BIG=T                                                             24
C                                                                       25
C     THE LARGEST ABSOLUTE-VALUED ELEMENT IS IN ROW NR.                 26
C                                                                       27
      NR=I                                                              28
 1010 CONTINUE                                                          29
C                                                                       30
C     EXCHANGE ROWS NR AND K, ONLY IF NR IS DIFFERENT FROM K.           31
C                                                                       32
      IF (NR.EQ.K) GO TO 1030                                           33
      DO 1020 J=1,N                                                     34
      T=A(K,J)                                                          35
      A(K,J)=A(NR,J)                                                    36
 1020 A(NR,J)=T                                                         37
C                                                                       38
C     CHANGE THE SIGN OF THE DETERMINANT,                               39
C     SINCE ROWS HAVE BEEN EXCHANGED.                                   40
C                                                                       41
      DET=-DET                                                          42
C                                                                       43
C     BOOK-KEEPING VECTOR NEXCH STORES THE INFORMATION                  44
C     ABOUT ROW EXCHANGES.                                              45
C                                                                       46
 1030 NEXCH(K)=NR                                                       47
      T=A(K,1)                                                          48
C                                                                       49
C     CONTINUOUS PART-PRODUCT FOR THE DETERMINANT.                      50
C                                                                       51
      DET=DET*T                                                         52
      T=1.0/T                                                           53
C                                                                       54
C     MODIFY ROW K BY DIVISION BY THE PIVOT ELEMENT.                    55
C                                                                       56
      DO 1040 J=1,NM1                                                   57
 1040 A(K,J)=A(K,J+1)*T                                                 58
```

```
      A(K,N)=T                                                          59
C                                                                       60
C     MODIFY ALL ROWS OTHER THAN ROW K BY ADDITION                      61
C     OF SUITABLE MULTIPLES OF ROW K.                                   62
C                                                                       63
      DO 1060 I=1,N                                                     64
      IF (I.EQ.K) GO TO 1060                                            65
      BIG=-A(I,1)                                                       66
      DO 1050 J=1,NM1                                                   67
 1050 A(I,J)=A(I,J+1)+BIG*A(K,J)                                        68
      A(I,N)=BIG*T                                                      69
 1060 CONTINUE                                                          70
 1070 CONTINUE                                                          71
C                                                                       72
C     PERFORM COLUMN EXCHANGES TO MATCH PREVIOUS ROW EXCHANGES          73
C     (IN REVERSE ORDER). NOTE THAT NEXCH(N)=N, ALWAYS. HENCE,          74
C     K IS NOT ALLOWED TO ASSUME THE VALUE N IN THE FOLLOWING DO-LOOP.  75
C                                                                       76
      NP1=N+1                                                           77
      DO 1090 J=2,N                                                     78
      K=NP1-J                                                           79
      NR=NEXCH(K)                                                       80
      IF (NR.EQ.K) GO TO 1090                                           81
      DO 1080 I=1,N                                                     82
      T=A(I,K)                                                          83
      A(I,K)=A(I,NR)                                                    84
 1080 A(I,NR)=T                                                         85
 1090 CONTINUE                                                          86
      RETURN                                                            87
      END                                                               88
```

```
      SUBROUTINE MOVE (A,B,M,N)                                         1
C                                                                       2
C     ROUTINE TO COPY MATRIX A AS MATRIX B. B=A                         3
C                                                                       4
      COMMON /IYENGAR/ I,MN,Y(12)                                       5
      REAL A(1),B(1)                                                    6
C                                                                       7
      MN=M*N                                                            8
      DO 1000 I=1,MN                                                    9
 1000 B(I)=A(I)                                                         10
      RETURN                                                            11
      END                                                               12
```

```
      SUBROUTINE MULT (A,B,C,L,M,N)                                      1
C                                                                        2
C     ROUTINE TO FORM THE PRODUCT OF MATRICES A AND B. C=A*B             3
C                                                                        4
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)                                   5
      REAL A(L,M),B(M,N),C(L,N)                                          6
C                                                                        7
      DO 1010 I=1,L                                                      8
      DO 1010 J=1,N                                                      9
      SUM=0.0                                                           10
      DO 1000 K=1,M                                                     11
 1000 SUM=SUM+A(I,K)*B(K,J)                                             12
 1010 C(I,J)=SUM                                                        13
      RETURN                                                            14
      END                                                               15
```

```
      SUBROUTINE OUTE (A,I,J,TITLE,TITEL)                                1
      COMMON /IYENGAR/ II,IMAX,IROW,IV(2),I1,JCOL,JJ,JMAX,J1,J2,KK,Y(2)  2
      REAL A(I,J)                                                        3
C                                                                        4
C     ROUTINE TO PRINT MATRIX A IN E-FORMAT.                             5
C     STORE E IN IV(2) FOR E-FORMAT OUTPUT.                              6
C                                                                        7
      IV(2)=10H0(XE12.5))                                                8
C                                                                        9
C     JMAX = NUMBER OF VERTICAL PARTITIONS OF THE MATRIX.               10
C                                                                       11
 1000 JMAX=(J+9)/10                                                     12
C                                                                       13
C     IMAX = NUMBER OF HORIZONTAL PARTITIONS OF THE MATRIX.             14
C                                                                       15
      IMAX=(I+24)/25                                                    16
      IV(1)=10H(1H0,I4,X1                                               17
      Y(1)=10H                                                          18
      Y(2)=10H                                                          19
      DO 1020 JCOL=1,JMAX                                               20
C                                                                       21
C     PRESCRIBE THE FIRST (J1) AND LAST (JJ) COLUMN NUMBERS             22
C     TO BE PRINTED ON THE PAGE.                                        23
C                                                                       24
      JJ=10*JCOL                                                        25
      J1=JJ-9                                                           26
      IF (J.LT.JJ) JJ=J                                                 27
      DO 1020 IROW=1,IMAX                                               28
C                                                                       29
C     PRESCRIBE THE FIRST (I1) AND LAST (II) ROW NUMBERS                30
C     TO BE PRINTED ON THE PAGE.                                        31
C                                                                       32
      II=25*IROW                                                        33
      I1=II-24                                                          34
      IF (I.LT.II) II=I                                                 35
C                                                                       36
C     LABEL THE OUTPUT, INDICATE MATRIX SIZE AND NUMBER THE COLUMNS.    37
C                                                                       38
      PRINT 1030, TITLE,TITEL,Y,I,J,(KK,KK=J1,JJ)                       39
      DO 1010 KK=I1,II                                                  40
C                                                                       41
C     NUMBER THE ROWS AND PRINT ELEMENTS OF THE MATRIX.                 42
```

```
C                                                                       43
 1010 PRINT IV, KK,(A(KK,J2),J2=J1,JJ)                                  44
      Y(1)=10H (CONTINUE                                                45
 1020 Y(2)=10HD)                                                        46
      RETURN                                                            47
C                                                                       48
C                                                                       49
      ENTRY OUTF                                                        50
C                                                                       51
C     SUBROUTINE OUTF (A,I,J,TITLE,TITEL)                               52
C     ROUTINE TO PRINT MATRIX A IN F-FORMAT.                            53
C     STORE F IN IV(2) FOR F-FORMAT OUTPUT.                             54
C                                                                       55
      IV(2)=10H0(XF12.5))                                               56
      GO TO 1000                                                        57
C                                                                       58
C                                                                       59
      ENTRY OUTG                                                        60
C                                                                       61
C     SUBROUTINE OUTG (A,I,J,TITLE,TITEL)                               62
C     ROUTINE TO PRINT MATRIX A IN G-FORMAT.                            63
C     STORE G IN IV(2) FOR G-FORMAT OUTPUT.                             64
C                                                                       65
      IV(2)=10H0(XG12.5))                                               66
      GO TO 1000                                                        67
C                                                                       68
C                                                                       69
 1030 FORMAT (1H110X4A10/1H010X4HSIZEI4,8H ROWS BYI4,8H COLUMNS/2H0 10I1 70
     13)                                                                71
      END                                                               72
```

```
      SUBROUTINE PMULT (A,B,K,L,X)                                       1
C                                                                        2
C     ROUTINE TO FORM THE MATRIX PRODUCT A*B IN A WHEN B IS SQUARE.      3
C                                                                        4
      COMMON /IYENGAR/ I,J,M,SUM,Y(10)                                   5
      REAL A(K,L),B(L,L),X(L)                                            6
C                                                                        7
      DO 1020 I=1,K                                                      8
C                                                                        9
C     GENERATE ELEMENTS OF I-TH ROW OF PRODUCT MATRIX                   10
C     AND STORE THESE IN VECTOR X.                                      11
C                                                                       12
      DO 1010 J=1,L                                                     13
      SUM=0.0                                                           14
      DO 1000 M=1,L                                                     15
 1000 SUM=SUM+A(I,M)*B(M,J)                                             16
 1010 X(J)=SUM                                                          17
C                                                                       18
C     REPLACE THE I-TH ROW OF A BY THE VECTOR X.                        19
C                                                                       20
      DO 1020 J=1,L                                                     21
 1020 A(I,J)=X(J)                                                       22
      RETURN                                                            23
      END                                                               24
```

```
      SUBROUTINE POSTM (A,B,K,L,X)                                      1
C                                                                       2
C     ROUTINE TO FORM THE MATRIX PRODUCT A*B IN B WHEN A IS SQUARE.     3
C                                                                       4
      COMMON /IYENGAR/ I,J,M,SUM,Y(10)                                  5
      REAL A(K,K),B(K,L),X(K)                                           6
C                                                                       7
      DO 1020 J=1,L                                                     8
C                                                                       9
C     GENERATE ELEMENTS OF J-TH COLUMN OF PRODUCT MATRIX                10
C     AND STORE THESE IN VECTOR X.                                      11
C                                                                       12
      DO 1010 I=1,K                                                     13
      SUM=0.0                                                           14
      DO 1000 M=1,K                                                     15
 1000 SUM=SUM+A(I,M)*B(M,J)                                             16
 1010 X(I)=SUM                                                          17
C                                                                       18
C     REPLACE THE J-TH COLUMN OF B BY VECTOR X.                         19
C                                                                       20
      DO 1020 I=1,K                                                     21
 1020 B(I,J)=X(I)                                                       22
      RETURN                                                            23
      END                                                               24
```

```
      SUBROUTINE RDCBC (A,M,N)                                          1
C                                                                       2
C     ROUTINE TO READ IN VALUES FOR ELEMENTS OF MATRIX A,               3
C     COLUMN BY COLUMN.                                                 4
C                                                                       5
      COMMON /IYENGAR/ I,J,Y(12)                                        6
      REAL A(M,N)                                                       7
C                                                                       8
      DO 1000 J=1,N                                                     9
 1000 READ 1010, (A(I,J),I=1,M)                                         10
      RETURN                                                            11
C                                                                       12
C                                                                       13
 1010 FORMAT (8F10.0)                                                   14
      END                                                               15
```

```
      SUBROUTINE RDCOLG (A,M,N)                                         1
C                                                                       2
C     ROUTINE TO READ IN VALUES FOR ELEMENTS OF MATRIX A,               3
C     COLUMN BY COLUMN IN A CONTINUOUS STRING OF COLUMNS.               4
C                                                                       5
      COMMON /IYENGAR/ Y(14)                                            6
      REAL A(M,N)                                                       7
C                                                                       8
      READ 1000, A                                                      9
      RETURN                                                            10
C                                                                       11
C                                                                       12
 1000 FORMAT (8F10.0)                                                   13
      END                                                               14
```

```
      SUBROUTINE RORBR (A,M,N)                                          1
C                                                                       2
C     ROUTINE TO READ IN VALUES FOR ELEMENTS OF MATRIX A,               3
C     ROW BY ROW.                                                       4
C                                                                       5
      COMMON /IYENGAR/ I,J,Y(12)                                        6
      REAL A(M,N)                                                       7
C                                                                       8
      DO 1000 I=1,M                                                     9
 1000 READ 1010, (A(I,J),J=1,N)                                        10
      RETURN                                                           11
C                                                                      12
C                                                                      13
 1010 FORMAT (8F10.0)                                                  14
      END                                                              15
```

```
      SUBROUTINE RGROWG (A,M,N)                                         1
C                                                                       2
C     ROUTINE TO READ IN VALUES FOR ELEMENTS OF MATRIX A,               3
C     ROW BY ROW IN A CONTINUOUS STRING OF ROWS.                        4
C                                                                       5
      COMMON /IYENGAR/ I,J,Y(12)                                        6
      REAL A(M,N)                                                       7
C                                                                       8
      READ 1000, ((A(I,J),J=1,N),I=1,M)                                 9
      RETURN                                                           10
C                                                                      11
C                                                                      12
 1000 FORMAT (8F10.0)                                                  13
      END                                                              14
```

```
      SUBROUTINE SCMUL (A,M,N,X)                                        1
C                                                                       2
C     ROUTINE TO MULTIPLY MATRIX A BY THE SCALAR QUANTITY X.            3
C     A = X TIMES A                                                     4
C                                                                       5
      COMMON /IYENGAR/ I,MN,Y(12)                                       6
      REAL A(1)                                                         7
C                                                                       8
      MN=M*N                                                            9
      DO 1000 I=1,MN                                                   10
 1000 A(I)=X*A(I)                                                      11
      RETURN                                                           12
      END                                                              13
```

```
      SUBROUTINE SINV (A,DA,N)                                          1
C                                                                       2
C     ROUTINE TO FIND THE INVERSE OF A SYMMETRIC,                       3
C     POSITIVE-DEFINITE MATRIX A.                                       4
C                                                                       5
      COMMON /IYENGAR/ I,IFIN,INDEX,INIT,J,JP1,K,SUM,TEMP,Y(5)          6
      REAL A(N,N)                                                       7
C                                                                       8
      IF (N.NE.1) GO TO 1020                                            9
      DA=A(1,1)                                                        10
      A(1,1)=1.0/DA                                                    11
      RETURN                                                           12
C                                                                      13
C     BUILT-IN SUBPROGRAM.                                             14
C                                                                      15
 1000 SUM=0.0                                                          16
      DO 1010 K=INIT,IFIN                                              17
 1010 SUM=SUM+A(I,K)*A(J,K)                                            18
      GO TO INDEX, (1040,1050,1070,1080)                               19
C                                                                      20
C     GENERATE MATRIX LAMBDA (A=LAMBDA*LAMBDA TRANSPOSE).              21
C     NOTE - ONLY THE DIAGONAL ELEMENTS ARE THOSE OF                   22
C     LAMBDA-INVERSE-TRANSPOSE (LIT).                                  23
C                                                                      24
C     (1,1) ELEMENT OF LIT.                                            25
C                                                                      26
 1020 DA=1.0/SQRT(A(1,1))                                              27
      A(1,1)=DA                                                        28
C                                                                      29
C     FIRST COLUMN OF LAMBDA.                                          30
C                                                                      31
      DO 1030 I=2,N                                                    32
 1030 A(I,1)=A(1,I)*DA                                                 33
      INIT=1                                                           34
      DO 1050 J=2,N                                                    35
      ASSIGN 1040 TO INDEX                                             36
      IFIN=J-1                                                         37
      I=J                                                              38
      GO TO 1000                                                       39
C                                                                      40
C     J-TH DIAGONAL ELEMENT OF LIT (2 .LE. J .LE. N).                  41
C                                                                      42
 1040 TEMP=1.0/SQRT(A(J,J)-SUM)                                        43
      A(J,J)=TEMP                                                      44
C                                                                      45
C     CONTINUOUS PART-PRODUCT FOR THE RECIPROCAL                       46
C     OF THE SQUARE ROOT OF THE DETERMINANT.                           47
C                                                                      48
      DA=DA*TEMP                                                       49
      IF (J.EQ.N) GO TO 1060                                           50
      ASSIGN 1050 TO INDEX                                             51
      JP1=J+1                                                          52
      DO 1050 I=JP1,N                                                  53
      GO TO 1000                                                       54
C                                                                      55
C     J-TH COLUMN OF LAMBDA (2 .LE. J .LE. N-1).                       56
C                                                                      57
 1050 A(I,J)=(A(J,I)-SUM)*TEMP                                         58
```

```
 1060 ASSIGN 1070 TO INDEX                                              59
C                                                                       60
C     FINAL VALUE OF THE DETERMINANT.                                   61
C                                                                       62
      DA=DA*DA                                                          63
      DA=1.0/DA                                                         64
C                                                                       65
C     GENERATE OFF-DIAGONAL ELEMENTS OF LIT.                            66
C                                                                       67
      DO 1070 I=2,N                                                     68
      IFIN=I-1                                                          69
      TEMP=A(I,I)                                                       70
      DO 1070 J=1,IFIN                                                  71
      INIT=J                                                            72
      GO TO 1000                                                        73
 1070 A(J,I)=-SUM*TEMP                                                  74
      ASSIGN 1080 TO INDEX                                              75
      IFIN=N                                                            76
C                                                                       77
C     FORM THE PRODUCT LIT*INVERSE OF LAMBDA.                           78
C     RESULT IS INVERSE OF THE ORIGINAL MATRIX A.                       79
C                                                                       80
      DO 1090 I=1,N                                                     81
      INIT=I                                                            82
      DO 1090 J=1,I                                                     83
      GO TO 1000                                                        84
 1080 A(I,J)=SUM                                                        85
 1090 A(J,I)=SUM                                                        86
      RETURN                                                            87
      END                                                               88
-------------------------------------------------------------------------
      SUBROUTINE SOLVE (A,B,N,L,DET)                                     1
C                                                                        2
C     ROUTINE TO SOLVE THE SYSTEM OF LINEAR                              3
C     SIMULTANEOUS EQUATIONS A*X=B.                                      4
C                                                                        5
      COMMON /IYENGAR/ ABSA,BIG,D,I,IJK,IP1,J,K,NM1,Y(4),Z               6
      REAL A(N,N),B(N,L)                                                 7
C                                                                        8
      IF (N.NE.1) GO TO 1010                                             9
      DET=A(1,1)                                                        10
      DO 1000 J=1,L                                                     11
 1000 B(1,J)=B(1,J)/DET                                                 12
      RETURN                                                            13
 1010 NM1=N-1                                                           14
      DET=1.0                                                           15
      DO 1070 I=1,NM1                                                   16
      IP1=I+1                                                           17
C                                                                       18
C     SEARCH COLUMN I FOR THE LARGEST ABSOLUTE-VALUED                   19
C     ELEMENT IN ROWS I THROUGH N.                                      20
C                                                                       21
      BIG=0.0                                                           22
      DO 1020 J=I,N                                                     23
      ABSA=ABS(A(J,I))                                                  24
      IF (BIG.GE.ABSA) GO TO 1020                                       25
      BIG=ABSA                                                          26
C                                                                       27
```

```
C     THE LARGEST ABSOLUTE-VALUED ELEMENT IS IN ROW K OF COLUMN I.
C
      K=J
 1020 CONTINUE
C
C     EXCHANGE ROWS K AND I, ONLY IF K IS DIFFERENT FROM I.
C
      IF (K.EQ.I) GO TO 1050
      DO 1030 J=I,N
      Z=A(I,J)
      A(I,J)=A(K,J)
 1030 A(K,J)=Z
      DO 1040 J=1,L
      Z=B(I,J)
      B(I,J)=B(K,J)
 1040 B(K,J)=Z
C
C     CHANGE THE SIGN OF THE DETERMINANT,
C     SINCE ROWS HAVE BEEN EXCHANGED.
C
      DET=-DET
 1050 Z=A(I,I)
C
C     CONTINUOUS PART-PRODUCT FOR THE DETERMINANT.
C
      DET=DET*Z
      Z=1.0/Z
C
C     MODIFY ELEMENTS OF A - THE REDUCTION PROCESS.
C     MODIFY MATRIX B ALSO.
C
      DO 1070 K=IP1,N
      D=-A(K,I)*Z
      DO 1060 J=1,L
 1060 B(K,J)=B(K,J)+D*B(I,J)
      DO 1070 J=IP1,N
 1070 A(K,J)=A(K,J)+D*A(I,J)
      Z=A(N,N)
C
C     FINAL VALUE OF THE DETERMINANT.
C
      DET=DET*Z
      Z=1.0/Z
C
C     PROCESS OF BACK-SUBSTITUTION TO GET THE SOLUTION MATRIX.
C
      DO 1090 K=1,L
      B(N,K)=B(N,K)*Z
      DO 1090 IJK=1,NM1
      I=N-IJK
      IP1=I+1
      D=0.0
      DO 1080 J=IP1,N
 1080 D=D+A(I,J)*B(J,K)
 1090 B(I,K)=(B(I,K)-D)/A(I,I)
      RETURN
      END
```

```
      SUBROUTINE SQTR (A,N)                                              1
C                                                                        2
C     ROUTINE TO TRANSPOSE THE SQUARE MATRIX A IN ITS OWN SPACE.         3
C                                                                        4
      COMMON /IYENGAR/ I,INCR,K,L,LAST,NM1,NP1,T,Y(6)                    5
      REAL A(1)                                                          6
C                                                                        7
      IF (N.EQ.1) RETURN                                                 8
      NM1=N-1                                                            9
      NP1=N+1                                                           10
      LAST=N*N                                                          11
      INCR=0                                                            12
      DO 1000 K=2,N                                                     13
      LAST=LAST-N                                                       14
      INCR=INCR+NM1                                                     15
      DO 1000 L=K,LAST,NP1                                              16
      I=L+INCR                                                          17
C                                                                       18
C     EXCHANGE ELEMENTS WITH SUBSCRIPTS I AND L.                        19
C                                                                       20
      T=A(I)                                                            21
      A(I)=A(L)                                                         22
 1000 A(L)=T                                                            23
      RETURN                                                            24
      END                                                               25
```

```
      SUBROUTINE SUB (A,B,C,M,N)                                         1
C                                                                        2
C     ROUTINE TO SUBTRACT MATRIX B FROM MATRIX A. C=A-B                  3
C                                                                        4
      COMMON /IYENGAR/ I,MN,Y(12)                                        5
      REAL A(1),B(1),C(1)                                                6
C                                                                        7
      MN=M*N                                                             8
      DO 1000 I=1,MN                                                     9
 1000 C(I)=A(I)-B(I)                                                    10
      RETURN                                                            11
      END                                                               12
```

```
      SUBROUTINE TMULT (A,B,C,L,M,N)                                     1
C                                                                        2
C     ROUTINE TO FORM THE MATRIX PRODUCT OF (TRANSPOSE OF MATRIX A)      3
C     AND MATRIX B. C=(TRANSPOSE OF A)*B                                 4
C                                                                        5
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)                                   6
      REAL A(L,M),B(L,N),C(M,N)                                          7
C                                                                        8
      DO 1010 I=1,M                                                      9
      DO 1010 J=1,N                                                     10
      SUM=0.0                                                           11
      DO 1000 K=1,L                                                     12
 1000 SUM=SUM+A(K,I)*B(K,J)                                             13
 1010 C(I,J)=SUM                                                        14
      RETURN                                                            15
      END                                                               16
```

```
      SUBROUTINE TRANS (A,B,M,N)                                        1
C                                                                       2
C     ROUTINE TO TRANSPOSE MATRIX A. B=TRANSPOSE OF A                   3
C                                                                       4
      COMMON /IYENGAR/ I,J,Y(12)                                        5
      REAL A(M,N),B(N,M)                                                6
C                                                                       7
      DO 1000 I=1,M                                                     8
      DO 1000 J=1,N                                                     9
 1000 B(J,I)=A(I,J)                                                     10
      RETURN                                                            11
      END                                                               12
```

```
      SUBROUTINE XABATC (A,B,C,L,M,X)                                   1
C                                                                       2
C     ROUTINE TO FORM THE MATRIX PRODUCT                                3
C     A*B*(TRANSPOSE OF A) WHEN B IS SYMMETRIC.                         4
C                                                                       5
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)                                  6
      REAL A(L,M),B(M,M),C(L,L),X(M)                                    7
C                                                                       8
      DO 1030 I=1,L                                                     9
C                                                                       10
C     GENERATE ROW I OF PART-PRODUCT A*B IN VECTOR X.                   11
C                                                                       12
      DO 1010 J=1,M                                                     13
      SUM=0.0                                                           14
      DO 1000 K=1,M                                                     15
 1000 SUM=SUM+A(I,K)*B(K,J)                                             16
 1010 X(J)=SUM                                                          17
C                                                                       18
C     PRODUCT MATRIX C IS SYMMETRIC, SINCE MATRIX B IS SYMMETRIC.       19
C     GENERATE PRODUCT MATRIX.                                          20
C                                                                       21
      DO 1030 J=I,L                                                     22
      SUM=0.0                                                           23
      DO 1020 K=1,M                                                     24
 1020 SUM=SUM+X(K)*A(J,K)                                               25
      C(J,I)=SUM                                                        26
 1030 C(I,J)=SUM                                                        27
      RETURN                                                            28
      END                                                               29
```

```
      SUBROUTINE XABTA (A,B,L,N,X)
C
C     ROUTINE TO FORM THE MATRIX PRODUCT
C     A*(TRANSPOSE OF B) IN A WHEN B IS SQUARE.
C
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)
      REAL A(L,N),B(N,N),X(N)
C
      DO 1020 I=1,L
C
C     GENERATE ROW I OF PRODUCT MATRIX IN VECTOR X.
C
      DO 1010 J=1,N
      SUM=0.0
      DO 1000 K=1,N
 1000 SUM=SUM+A(I,K)*B(J,K)
 1010 X(J)=SUM
C
C     REPLACE ROW I OF MATRIX A BY VECTOR X.
C
      DO 1020 J=1,N
 1020 A(I,J)=X(J)
      RETURN
      END
```

```
      SUBROUTINE XABTC (A,B,C,L,M,N)
C
C     ROUTINE TO FORM THE MATRIX PRODUCT
C     A*(TRANSPOSE OF B)
C
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)
      REAL A(L,M),B(N,M),C(L,N)
C
      DO 1010 I=1,L
      DO 1010 J=1,N
      SUM=0.0
      DO 1000 K=1,M
 1000 SUM=SUM+A(I,K)*B(J,K)
 1010 C(I,J)=SUM
      RETURN
      END
```

```
      SUBROUTINE XATBAC (A,B,C,L,M,X)                                    1
C                                                                        2
C     ROUTINE TO FORM THE MATRIX PRODUCT                                 3
C     (TRANSPOSE OF A)*B*A WHEN B IS SYMMETRIC.                          4
C                                                                        5
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)                                   6
      REAL A(L,M),B(L,L),C(M,M),X(L)                                     7
C                                                                        8
      DO 1030 J=1,M                                                      9
C                                                                       10
C     GENERATE COLUMN J OF PART-PRODUCT B*A IN VECTOR X.                 11
C                                                                       12
      DO 1010 I=1,L                                                     13
      SUM=0.0                                                           14
      DO 1000 K=1,L                                                     15
 1000 SUM=SUM+B(I,K)*A(K,J)                                             16
 1010 X(I)=SUM                                                          17
C                                                                       18
C     PRODUCT MATRIX C IS SYMMETRIC, SINCE MATRIX B IS SYMMETRIC.       19
C     GENERATE PRODUCT MATRIX.                                          20
C                                                                       21
      DO 1030 I=1,J                                                     22
      SUM=0.0                                                           23
      DO 1020 K=1,L                                                     24
 1020 SUM=SUM+A(K,I)*X(K)                                               25
      C(I,J)=SUM                                                        26
 1030 C(J,I)=SUM                                                        27
      RETURN                                                            28
      END                                                               29
```

```
      SUBROUTINE XATBB (A,B,L,N,X)                                       1
C                                                                        2
C     ROUTINE TO FORM THE MATRIX PRODUCT                                 3
C     (TRANSPOSE OF A)*B IN B WHEN A IS SQUARE.                          4
C                                                                        5
      COMMON /IYENGAR/ I,J,K,SUM,Y(10)                                   6
      REAL A(L,L),B(L,N),X(L)                                            7
C                                                                        8
      DO 1020 J=1,N                                                      9
C                                                                       10
C     GENERATE COLUMN J OF PRODUCT MATRIX IN VECTOR X.                  11
C                                                                       12
      DO 1010 I=1,L                                                     13
      SUM=0.0                                                           14
      DO 1000 K=1,L                                                     15
 1000 SUM=SUM+A(K,I)*B(K,J)                                             16
 1010 X(I)=SUM                                                          17
C                                                                       18
C     REPLACE COLUMN J OF MATRIX B BY VECTOR X.                         19
C                                                                       20
      DO 1020 I=1,L                                                     21
 1020 B(I,J)=X(I)                                                       22
      RETURN                                                            23
      END                                                               24
```

ACKNOWLEDGMENTS

This documentation has been prepared at the Fritz Engineering Laboratory, Department of Civil Engineering, Lehigh University, Bethlehem, Pennsylvania. Dr. David A. VanHorn is the Chairman of the Department, and Dr. Lynn S. Beedle is the Director of the Laboratory.

By his appointment as a member of the Computer Systems Group in Fall 1968, Mr. Iyengar had the opportunity to peruse and debug several programs written by fellow research workers in the Laboratory. Dr. Celal N. Kostem is the Chairman of the Group.

Many colleagues have contributed useful suggestions which now form parts of this work. Mr. Edward T. Manning, Jr. was associated with Mr. Iyengar in the development of the earlier version (FCMXPK) of this package.

A special debt of gratitude is due to Mrs. Michele Kostem who patiently read through the manuscript and typed the draft copy.

Sincere appreciation is extended to Mrs. Mary C. Ambrose who typed the final copy, assisted by Mrs. Ruth A. Grimes.

REFERENCES

1. Ralston, Anthony and Wilf, Herbert S. (Eds)
MATHEMATICAL METHODS FOR DIGITAL COMPUTERS
John Wiley and Sons, Inc., New York, 1960

2. Control Data Corporation
CONTROL DATA 6000 SERIES COMPUTER SYSTEMS
SCOPE 3.3 REFERENCE MANUAL (60305200B)
Sunnyvale, California, 1971

3. Lehigh University Computing Center
COMPUTING NOTICE, APRIL-MAY 1971
Bethlehem, Pennsylvania, May 1971

4. Wylie, C. R., Jr.
ADVANCED ENGINEERING MATHEMATICS
McGraw-Hill Book Company
New York, 1965