COMPUTER PROGRAMS FOR THE SOLUTION OF
SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

by William T. Sequi

Prepared by
MEMPHIS STATE UNIVERSITY
Memphis, Tenn. 38152
for George C. Marshall Space Flight Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • JANUARY 1973
ERRATA

NASA Contractor Report CR-2173

COMPUTER PROGRAMS FOR THE SOLUTION OF SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

By

William T. Segui
January 1973

Cover and Title Page: The author's name is incorrectly spelled as Sequi. It is correctly spelled as Segui.

Issued July 1973
FORTRAN subprograms for the solution of systems of linear algebraic equations are described, listed, and evaluated in this report. Procedures considered are direct solution, iteration, and matrix inversion. Both incore methods and those which utilize auxiliary data storage devices are considered. Some of the subroutines evaluated require the entire coefficient matrix to be in core, whereas others account for banding or sparceness of the system.

General recommendations relative to equation solving are made, and on the basis of tests, specific subprograms are recommended.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>3</td>
</tr>
<tr>
<td>1.1 General</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Test Matrix and Procedures</td>
<td>3</td>
</tr>
<tr>
<td>II. SUBROUTINE DESCRIPTIONS AND LISTINGS</td>
<td>7</td>
</tr>
<tr>
<td>2.1 In-core, Non-banded Equation Solvers</td>
<td>9</td>
</tr>
<tr>
<td>SIMEQ</td>
<td>10</td>
</tr>
<tr>
<td>SIMEQ3</td>
<td>13</td>
</tr>
<tr>
<td>WUGEL</td>
<td>16</td>
</tr>
<tr>
<td>GAUSEL</td>
<td>19</td>
</tr>
<tr>
<td>SIMEQ2</td>
<td>22</td>
</tr>
<tr>
<td>FACTOR and LU</td>
<td>26</td>
</tr>
<tr>
<td>GAUSS</td>
<td>30</td>
</tr>
<tr>
<td>SOLVIT</td>
<td>34</td>
</tr>
<tr>
<td>SIMEQ1</td>
<td>41</td>
</tr>
<tr>
<td>DPSIM</td>
<td>45</td>
</tr>
<tr>
<td>SHORT</td>
<td>48</td>
</tr>
<tr>
<td>SOLVE1</td>
<td>50</td>
</tr>
<tr>
<td>WUIGJ</td>
<td>53</td>
</tr>
<tr>
<td>WUGJN</td>
<td>56</td>
</tr>
<tr>
<td>SLVE</td>
<td>59</td>
</tr>
<tr>
<td>GASSIM</td>
<td>62</td>
</tr>
<tr>
<td>GJR</td>
<td>65</td>
</tr>
<tr>
<td>SIMULT</td>
<td>69</td>
</tr>
<tr>
<td>SOLVE3</td>
<td>77</td>
</tr>
<tr>
<td>SOLVE2</td>
<td>82</td>
</tr>
<tr>
<td>GSEID</td>
<td>88</td>
</tr>
<tr>
<td>2.2 In-core, Non-banded Inversion Subroutines</td>
<td>91</td>
</tr>
<tr>
<td>AINVER</td>
<td>92</td>
</tr>
<tr>
<td>BINVER</td>
<td>94</td>
</tr>
<tr>
<td>MATIN1</td>
<td>96</td>
</tr>
<tr>
<td>MATIN4</td>
<td>98</td>
</tr>
<tr>
<td>MIV</td>
<td>100</td>
</tr>
<tr>
<td>INVT</td>
<td>103</td>
</tr>
<tr>
<td>INVERT</td>
<td>107</td>
</tr>
</tbody>
</table>
This report was prepared by Memphis State University under a cooperative agreement with Marshall Space Flight Center, Marshall Space Flight Center, Alabama. The purpose of the agreement was to collect and evaluate as many matrix solution and inversion subroutines as could be reasonably obtained. The technical monitor for the agreement was John E. Key of the Analytical Mechanics Division, Astronautics Laboratory.
COMPUTER PROGRAMS FOR THE SOLUTION OF SYSTEMS
OF LINEAR ALGEBRAIC EQUATIONS

SUMMARY

Forty-seven FORTRAN subprograms for the solution of systems of linear algebraic equations are catalogued. Complete descriptions of their use are provided as is a listing of each. Both in-core schemes and those requiring auxiliary data storage devices are considered. The in-core methods include those which require the full coefficient matrix in core and those which take advantage of banding or sparceness.

All subprograms are tested and evaluated experimentally on the basis of execution time and accuracy. A symmetric, positive definite, banded test matrix is used in each case.

Matrix inversion is found to be an inefficient technique, even if multiple right-hand side constant vectors are to be solved. In such cases, either Gauss elimination with multiple constant vectors treated simultaneously, or decomposition with the retention of the upper and lower triangular matrices, is recommended. Double precision arithmetic is suggested as a means of reducing roundoff errors, and should always be employed when permitted by the computer's core capacity.
On the basis of results obtained with the single test system of equations, specific subprograms are recommended for each category of problem. These are: SOLVIT, the combination of FACTOR and LU, and WUGEL for in-core, non-banded systems; SIMULT, for in-core, non-banded, sparse systems; DPINV and MIV if an inverse is required; CHOLES, the combination of DECOM and SOLV, and BANSLV for in-core, banded systems; and either DISPL or BANSOL for large systems requiring out-of-core storage.
I. INTRODUCTION

1.1 General

Many methods of analysis and design in various disciplines of engineering require the solution of systems of linear algebraic equations. In particular, the finite element method can require the solution of extremely large numbers of equations. Until relatively recently, however, the scope of such methods was limited by the excessive time required to solve the equations manually. With the advent of the digital computer, such problems can now be handled routinely in most cases.

In this report, a collection of FORTRAN subroutines for the solution of simultaneous equations has been assembled, tested, and evaluated. Most of these subroutines can be easily used for small as well as large sets of equations, but the emphasis has been on their performance with large systems. In Section 2., instructions for the use of these subprograms are given, and listings are provided.

A presentation of some of the basic concepts and common solution methods can be found in Appendix A.

1.2 Test Matrix and Procedures

In order for the accuracy of a solution technique to be determined, it is necessary to solve a set of equations whose
exact solution is known. Many such systems exist, and their properties are well documented \((1),(2),(3)\). The one selected for use in the present study is given in matrix form as follows:

\[
\begin{bmatrix}
2 & -1 & & & \\
-1 & 2 & -1 & & \\
& -1 & 2 & -1 & \\
& & & \ddots & \\
& & & & -1 & 2 & -1 \\
& & & & & -1 & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
\vdots \\
x_n
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
\vdots \\
\vdots \\
0
\end{bmatrix} \quad (1.1)
\]

The solutions are \(x_i = i\) for \(i = 1, \ldots, n\) where \(n\) is the number of unknowns. The coefficient matrix is tri-diagonal, all elements being zero with the exception of the diagonal elements and those adjacent to the main diagonal. All diagonal elements have a value of 2 with the exception of the last, which has a value of 1. The off-diagonal elements are all \(-1\). The constant vector contains all zeros except for the \(n\)th element which is \(1\).

Since this study is oriented toward systems of equations arising from physical situations, the system represented by Eq. (1.1) is desirable for several reasons. The coefficient matrix is symmetric, banded, and positive definite, properties
frequently encountered in physical problems. In fact, this system is the set of equilibrium equations obtained from the stiffness analysis of a series of \( n - 1 \) springs of unit stiffness, fixed at one end with a unit load applied at the other. Other, more restrictive matrices could also be used in order to fully test the capabilities of the subroutines included herein, but such an evaluation is beyond the scope of this report.

No attempt has been made to compare methods on any but an experimental basis, even though it is known that some techniques are less efficient than others (i.e., Gauss–Jordan elimination involves more operations than Gauss elimination). The execution time and accuracy can also be influenced by the way an algorithm is programmed.

To compare the accuracy obtained with different sub-programs when solving the same system of equations, two error measures were used. These are given by

\[
f = \frac{1}{n} \left( \sum e_i^2 \right)^{1/2}
\]

and

\[
m = \max |e_i|
\]

where \( e_i \) is the difference between the exact and computed values of \( x_i \). The error measure \( f \) estimates the overall inaccuracy of the solution, and \( m \) is simply the largest error in any of the unknowns.
For a comparison of execution times, machine language subroutines which initialize and read the computer's internal clock were furnished by the Computing Center at Memphis State University. Although the operating system is such that execution times vary slightly with the status of the system at the time that a program is run, a relative measure, adequate for comparison purposes, can be obtained.

The basic test procedure, insofar as the programming is concerned, was to (1) generate the test matrix and constant vector, (2) initialize the internal clock, (3) execute the solution subroutine, (4) read the internal clock, (5) compute the error measures, and (6) print the results. In the case of inversion subroutines, the execution time was the total time required to invert the coefficient matrix and obtain a solution vector. The error measures were computed with respect to the solution vector and not the inverse matrix. The procedure was necessarily modified somewhat for those methods requiring auxiliary data storage devices. A typical test program with its output is given in Appendix B.

The computing facilities used in the evaluation of these subroutines consist of an XDS Sigma 9 with 128K words of core storage, one word consisting of 32 bits.
II. SUBROUTINE DESCRIPTIONS AND LISTINGS

In the descriptions that follow, an attempt is made to give a potential user of these subroutines complete information on their implementation. No evaluation or comparison is made in this section; this part of the report functions solely as a user's manual.

In most cases, the subroutines are in the same form as the author found them, and no attempt was made to improve them. Where changes were made to a previously published version, these changes are documented in the description.

All subroutines described are accompanied by a listing with the exception of BANSLV. At the time of this writing, permission to print the listing had not been received from the publisher of its source. A listing can be found in Reference 4, however.

With three exceptions, those subroutines designated as in-core and non-banded are those which require the full coefficient matrix in core. Subroutines SIMULT, SOLV 3, and SOLVE 2 are designed for sparse matrices and require only the nonzero coefficients in core. Subroutine SOLVE 3 accounts for symmetry and requires only the upper triangular portion of the coefficient matrix. Those termed in-core and banded account for the banded nature of the coefficient matrix and require only the coefficients within the band to be stored in core. In most cases only the upper or lower half-band of symmetric matrices
is required. The term half-band as used here is actually the semi-band width which includes the diagonal coefficient. The two out-of-core subroutines given use auxiliary data storage files, and require only a portion of the coefficient matrix to be in core at any given time.
2.1 **In-core, Non-banded Equation Solvers**
SIMEQ


Method: Gauss elimination (See appendix A).

Usage:

\begin{verbatim}
DIMENSION A(n,n),Y(n),X(n)
.
.
.
CALL SIMEQ (A,Y,X,N)
\end{verbatim}

where

A = coefficient matrix

Y = constant vector

X = solution vector

n = maximum permissible number of equations

N = number of equations to be solved

The original value of A is destroyed.
The array sizes in the DIMENSION statement in SIMEQ must conform to the following:

\[ \text{DIMENSION } A(n,n), Y(n), X(n) \]

**Comment**: The original version is a program. Only those changes necessary to convert it to a subroutine were made.
SUBROUTINE SIMCQ (A, Y, X, N)
DIMENSION A(100,100), Y(100), X(100)
DOBLE PRECISION A, X, Y
M = N = 1
DO 10 I = 1, M
L = I + 1
DO 10 J = L, N
IF (A(J,J)) 6, 10, 6
6 DO K = L, N
8 A(J,K) = A(J,K) - A(I,K)*A(J,I)/A(I,I)
   Y(J) = Y(J) - Y(I)*A(J,I)/A(I,I)
10 CONTINUE
X(N) = Y(N)/A(N,N)
DO 30 I = 1, M
K = N - 1
L = K + 1
DO 20 J = L, N
20 Y(K) = Y(K) - X(J)*A(K,J)
30 X(K) = Y(K)/A(K,K)
RETURN
END
SIMEQ3

Method:  Gauss elimination (see Appendix A).

Usage:

DIMENSION A(n,n+1),C(n)

.  

.  

.  

CALL SIMEQ3 (A,N,C)

where

A = coefficient matrix augmented by the constant vector attached as an additional column

C = solution vector

n = maximum permissible number of equations

N = number of equations to be solved

The original value of A is retained.

The array sizes in the DIMENSION statement in SIMEQ3 must conform to the following:

DIMENSION A(n,n+1),B(n,n+1),C(n)

In-the-event A is singular, an error message is printed.

The output device designation is specified in the statement

INTEGER OUT/108/
SUBROUTINE SIMEQ3(A,N,C)  
DIMENSION A(100*101),B(100,101),C(100)  
INTEGER OUT/108/  
TOL=5.0E-08  
KA=N+1  
DO 5 I=1,KA  
B(I,J)=0.0  
K2=2  
K3=1  
DO 10 I=1,N  
DO 10 J=1,KA  
MOVE MATRIX TO WORKING AREA  
10 B(I,J)=A(I,J)  
20 IF(ABS(B(K3,K3))=TOL)30,100,100  
30 IF(K2=N)40,40,30C  
40 DO 50 I=1,KA  
D=B(K3,I)  
B(K3,I)=B(K2,I)  
B(K2,I)=D  
50 CONTINUE  
K2=K2+1  
GO TO 20  
100 K1=K3  
K3=K3+1  
K2=K3+1  
IF(K3=N)120,120,200  
120 DO 130 I=K3,KA  
B(K1,I)=B(K1,I)/B(K1,K1)  
130 CONTINUE  
DO 140 I=K3,N  
DO 140 J=K3,KA  
B(I,J)=B(I,J)/B(I,K1)*B(K1,J)  
140 CONTINUE  
GO TO 20  
200 C(N)=B(N,N+1)/B(N,N)  
K3=N+1  
210 K1=K3+1  
D=0.0  
DO 220 I=K1,N  
220 D=D+B(K3,I)*C(I)  
C(K3)=B(K3,KA)-C  
K3=K3+1  
IF(K3)350,350,210  
350 TOL=TOL/10.0  
IF(TOL=1.0E-30)320,310,310  
310 K2=K3+1  
GO TO 20  
320 WRITE(OUT,321)  
321 FORMAT(/'23H NO SOLUTION FROM SIMEQ//')  
14
350 RETURN
END
WUGEL

Source: Written by T. S. Wu, Department of Civil Engineering, Memphis State University

Method: Gauss elimination (see Appendix A)

Usage:

\[
\text{DIMENSION } A(n,n + r)
\]

\[
\cdot
\]

\[
\cdot
\]

\[
\cdot
\]

\[
\text{CALL WUGEL (A,N,M)}
\]

where

\[A = \text{coefficient matrix augmented by the constant vectors attached as additional columns}\]

\[n = \text{maximum permissible number of equations}\]

\[r = \text{maximum permissible number of constant vectors}\]

\[N = \text{number of equations to be solved}\]

\[M = N + \text{the number of constant vectors to be solved}\]

The original value of A is destroyed, and the solution vectors replace the corresponding constant vectors.
The DIMENSION statement in WUGEL must conform to the following:

\[
\text{DIMENSION } A(n, n + r)
\]
SUBROUTINE WUGEU(A,N,M)
DIMENSION A(100,100)

DO 100 K=1,N
K1=K+1
100  J=K1*M
A(J)=A(J)/A(K)
IF (K1-N) GO TO 100
DO IOC I=K1,N
DO 100 J=K1*M
A(I,J)=A(I,J)-A(I,K)
CONTINUE
DO 115 L=N,N-1
K=K-L
DO 115 I=1,K2
A(I,M)=A(I,K)+A(I,K)
115 CONTINUE
RETURN
END
GAUSEL

Method: Gauss elimination (see Appendix A).

Usage:

DIMENSION A(n,n + 1)

CALL GAUSEL (A,ND)

where

A = the augmented coefficient matrix. The constant vector is attached as an additional column with its algebraic sign reversed. That is, for the matrix equation

\[ AX = B \]

the augmented matrix takes the form

\[
\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1N} & -b_1 \\
  a_{21} & a_{22} & \cdots & a_{2N} & -b_2 \\
  \vdots & \vdots & \ddots & \vdots & \vdots \\
  a_{N1} & a_{N2} & \cdots & a_{NN} & -b_N
\end{bmatrix}
\]
n = maximum permissible number of equations.

ND = number of equations to be solved

The original value of A is destroyed, and the solution vector is returned in the last column of A.

The array size in the DIMENSION statement in GAUSEL must conform to the following:

DIMENSION A(n,n + 1)
SUBROUTINE GAUSEL( A, ND )
DIMENSION A(100,101)
N1=ND+1
DO 1 I=1,ND
   IF( I=1 ) 2,2,3
3 M=I=1
   DO 4 K=1,M
      R=A(I,K)
   DO 4 J=K,N1
4 A(I,J)=R*A(K,J) + A(I,J)
   D=A(I,I)
   DO 1 J=I,N1
1 A(I,J)=A(I,J)/(-D)
   N2=ND=1
   DO 5 I=1,N2
      L=ND=1
   M=L+1
   DO 5 J=M,N1
5 A(L,N1)= A(L,N1) + A(L,J)*A(J,N1)
RETURN
END
SIMEQ2

Source: George C. Marshall Space Flight Center
Computation Laboratory Programmer Procedures Manual

Method: Method of rotations (see Reference 5).

Usage:

```plaintext
DIMENSION A(m),X(n)

CALL SIMEQ2 (A,X,N,IE)
```

where

A = coefficient matrix augmented by the constant vector attached as an additional column. This augmented matrix must be stored in one-dimensional array, row by row.

X = solution vector

n = maximum permissible number of equations

m = maximum permissible number of elements in A

m = n(n + 1)
$N =$ number of equations to be solved

$IE =$ a singularity indicator

$IE = 0$ if $A$ is non-singular

$IE \neq 0$ if $A$ is singular

The original value of $A$ is destroyed.

**Note:** This subroutine calls subroutine SIM.
SUBROUTINE SIMEQ2(A, X, N, IE)
DIMENSION A(1), X(1)
NN = N*N
MM = N+1
N11 = N+1
DO 1 K = 1, N11
DO 1 J = K, N11
I = K + MM - J
L = J + MM*(K+1)
TEMP = A(L)
A(L) = A(I)
A(I) = TEMP
NM1 = N11*MM
L = 0
DO 3 K = MM, NM1, MM
L = L + 1
3 X(L) = A(K)
I = N
DO 4 K = 2, N
DO 4 J = 1, N
L = J + MM*(K+1)
I = I + 1
4 A(I) = A(L)
MMM = MM + N - 1
NNM = NN + 1
I = 0
DO 5 K = NNM, MMM
I = I + 1
5 A(K) = X(I)
NNM = N*MM
CALL SIM(A, N, IE, MMM)
L = 0
DO 2 I = N, N, -N
L = L + 1
2 X(L) = A(I)
RETURN
END
SUBROUTINE SIM(A,N,IE,MM)
DIMENSION A(N,MM)
IE=0
NN=N+1
DO 8 K=N,NN
AMAX=0.0
DO 2 I=K,N
SIGN=A(I,K)
ABPIV=ABS(SIGN)
IF (ABPIV=AMAX)2,2,1
1 AMAX=ABPIV
DIV=SIGN
IMAX=I
2 CONTINUE
IF (AMAX)3,3,4
IE=1
RETURN
4 IF (IMAX=K)5,7,5
5 DO 6 J=K,MM
AMAX=A(IMAX,J)
A(IMAX,J)=A(K,J)
A(K,J)=AMAX
6 CONTINUE
7 KK=K+1
DO 8 I=KK,N
AMAX=A(I,K)/DIV
DO 8 J=KK,MM
A(I,J)=A(I,J)*AMAX/A(K,J)
8 CONTINUE
JJ=N
DO 12 J=1,N
A(N,J)=A(JJ,MM)/A(JJ,JJ)
KK=N
IF (KK)13,13,10
10 DO 11 I=1,NN
A(KK,MM)=A(KK,MM)+A(N,J)*A(KK,JJ)
11 KK=KK+1
12 JJ=JJ+1
13 RETURN
END
FACTOR and LU


Method: Decomposition (see Appendix A).

Usage:

\[
\text{DIMENSION A(n,n),B(n,r),L(n)}
\]

\[
\text{.}
\]

\[
\text{.}
\]

\[
\text{.}
\]

\[
\text{DETER = FACTOR(N,A,L)}
\]

\[
\text{CALL LU(A,L,N,B,M)}
\]

where

\[A = \text{coefficient matrix}\]

\[B = \text{matrix of right-hand side constants (consisting of one or more constant vectors)}\]

\[L = \text{working storage}\]

\[n = \text{maximum permissible number of equations}\]
r = maximum permissible number of constant vectors

DETER = determinant of A

N = number of equations to be solved

M = number of constant vectors to be solved

FACTOR is a function subprogram which decomposes A into upper and lower triangular matrices. The value of the function is the determinant of A. Subroutine LU solves for a given set of constant vectors and may be called repeatedly for different sets of constant vectors.

The constant vectors are replaced by the corresponding solution vectors.

The array sizes in the DIMENSION statements in FACTOR and LU must conform to the following:

In FACTOR: \[
\text{DIMENSION } A(n,n), P(n)
\]

In LU: \[
\text{DIMENSION } A(n,n), P(n), B(n,r), T(n)
\]
FUNCTION FACTOR(N,A,P)
DIMENSION A(100),P(100)
REAL MAX
INTEGER P,P1,P2
DO 10 I=1,N
10 P(I)=I
N1=N-1
DELTA=1.0
DO 11 K=1,N1
PK=P(K)
MAX=ABS(A(PK,K))
DO 12 I=K,N
PI=P(I)
IF(ABS(A(PI,K))*LT*MAX) GO TO 12
MAX=ABS(A(PI,K))
PK=I
DELTA=DELTA
12 CONTINUE
IF(MAX*GT*0) GO TO 13
FACTOR = 0
RETURN
13 PI=P(K)
P(K)=P(PK)
P(PK)=PI
K1=K+1
DO 14 I=K1,N
PI=P(I)
A(PI,K)=A(PI,K)*A(PK,K)
DO 14 J=K1,N
A(PI,J)=A(PI,J)*A(PK,K)
14 DELTA = DELTA*A(PK,K)
FACTOR = DELTA*A(P(N),N)
RETURN
END
SUBROUTINE LU(A, P, N, B, M)
DIMENSION A(100,100), P(100), B(100,5), T(100)
INTEGER P, H
DO 11 K=1, M
T(I)=B(P(I),K)
DO 10 I=2, N
H=P(I)
T(I)=B(H,K)
ILIM = I-1
DO 10 J=1, ILIM
10 T(I)=T(I)-A(H, J)*T(J)
T(N)=T(N)/A(P(N), N)
B(N,K)=T(N)
DO 11 I=2, N
IC = N+1-I
H=P(IC)
IP1=IC+1
DO 12 J=1, IP1
12 T(IC)=T(IC)-A(H, J)*T(J)
T(IC)=T(IC)/A(H, IC)
11 B(IC,K)=T(IC)
RETURN
END
GAUSS


Method: Gauss elimination with partial pivoting (see Appendix A).

Usage:

```
DIMENSION A(n,n + 1), X(n)
.
.
.
CALL GAUSS (A,X,N)
```

where

A = coefficient matrix augmented by the constant vector attached as an additional column.

X = solution vector

n = maximum permissible number of equations

N = number of equations to be solved
The original value of A is destroyed.

The array sizes in the DIMENSION statement in GAUSS must conform to the following:

```
DIMENSION A(n,n + 1), X(n)
```

**Comment:** The original version is a program. Only those changes necessary to convert it to a subroutine were made.
SUBROUTINE GAUSS(A,X,N)

REFERENCE - APPLIED NUMERICAL METHODS FOR DIGITAL
COMPUTATION, BY JAMES, SMITH, AND
WOFFORD.

SOLUTION OF SIMULTANEOUS EQUATIONS BY
GAUSSIAN ELIMINATION

DIMENSION A(100,101),X(100)
M=N+1
L=N+1
DO 12 K=1,L
JJ=K
BIG=ABS(A(K,K))
KP1=K+1

SEARCH FOR LARGEST POSSIBLE PIVOT ELEMENT

DO 7 I=KP1,N
AB=ABS(A(I,K))
IF(BIG=AB)6,7,7
6 BIG=AB
JJ=1
7 CONTINUE

DECISION ON NECESSITY OF ROW INTERCHANGE

IF(JJ=K)8,10,8

ROW INTERCHANGE

8 DO 9 J=K,N
TEMP=A(J,J)
A(J,J)=A(K,J)
9 A(K,J)=TEMP

CALCULATION OF ELEMENTS OF NEW MATRIX

10 DO 11 I=KP1,N
GUOT=A(I,K)/A(K,K)
DO 11 J=KP1,M
11 A(I,J)=A(I,J)-GUOT*A(K,J)
DO 12 I=KP1,N
12 A(I,K)=0.

BACK-SUBSTITUTION

X(N)=A(N,M)/A(N,N)
DO 14 NN=1/L
  SUM=0.
  I=N-NN
  IP1=I+1
  DO 13 J=IP1,N
  13 SUM=SUM+A(I,J)*X(J)
  14 X(I)=(A(I,M)=SUM)/A(I,I)
RETURN
END
SOLVIT


Method: Decomposition with partial pivoting, iterative improvement, and initial scaling of each row (see Appendix A). Summations are performed in double precision arithmetic.

Usage:

DIMENSION A(n,n),B(n),X(n),W(n,n + 5)
.
.
.

CALL SOLVIT(A,N,M,B,X,ACC,MAXIT,IT,IN,W)

where

A = coefficient matrix

B = constant vector

X = solution vector

W = working storage

n = maximum permissible number of equations

N = number of equations to be solved
M = value of n in the DIMENSION statement in the calling program

ACC = desired accuracy (see note)

MAXIT = maximum number of iterations desired. If MAXIT is specified as zero, no iterative improvement will be performed.

IT = a singularity and convergence indicator. IT is returned as -1 if A is singular, as 0 if convergence is not achieved within the number of iterations specified (i.e., MAXIT), and as the number of iterations required if convergence is achieved.

IN = 1 for the first entry

IN = 2 for subsequent entries with different constant vectors.

The original value of A is destroyed, but the upper and lower triangular matrices into which A is decomposed are retained for use in solving for additional constant vectors.

Notes:

1. Some insight into the selection of a value for ACC may be gained from the following discussion.
After obtaining a solution vector $X_0$, a residual vector is computed, and a solution vector $X_r$ is obtained using the residual vector as a right-hand side constant vector. If the largest element in $X_r$ is less than or equal to the product of ACC and the largest value in the solution vector $X_0$, convergence has been achieved.

Using the test matrix given in Section 1, an accuracy of better than eight significant digits was achieved with MAXIT = 2 iterations and ACC = 0.001, but IT was returned as 0 (no convergence). When MAXIT was specified as 3, with ACC = 0.001, IT was returned with a value of 3 (convergence in 3 iterations).

2. This subroutine calls subroutine SIMEQA.
SUBROUTINE SOLVIT(A,N,M,B,X,ACC,MAXIT,IT,IN,W)
C A INPUT MATRIX, SQUARE
C N NO. OF ROWS OF A
C M NO. OF ROWS OF THE ARRAY IN WHICH MATRIX A IS STORED
C B VECTOR OF CONSTANTS (RIGHT HAND SIDE)
C X THE SOLUTION VECTOR
C ACC DESIRED ACCURACY
C MAXIT MAXIMUM NO. OF ITERATIONS (10 IS MORE THAN ENOUGH)
C IT FLAG SET =-1 IF A IS SINGULAR
C =0 IF CONVERGENCE IS NOT ACHIEVED
C = NO. OF ITERATIONS TO CONVERGE
C IN =1 FOR FIRST ENTRY
C =2 FOR SUBSEQUENT ENTRIES WITH NEW B
C W WORKING STORAGE* N*(N+5) WORDS
DIMENSION W(N,M)
IF (IN .EQ. 1) CALL SIMEGA(A,N,M,ACC,MAXIT,IT,IN,W(1,1),
W(1,N+1),W(1,N+2),W(1,N+3),W(1,N+4),W(1,N+5))
CALL SIMQSB(B,X)
RETURN
END
SUBROUTINE SIMEQ(A, NN, MM, EPS, MAXIT, IT, I1N, AB, FACT, ITOP, Y, Z)
DOUBLE PRECISION SUM
DIMENSION A(1:N, 1:M), B(1:N), Z(1:M), Y(1:M), AB(1:N, 1:M), FACT(1:N, 1:M), ITOP(1:N)
GO TO +C0
ENTRY SIMEQB(B, X)
DIMENSION B(1:N, 1:M), X(N)
GO TO (100, 250, IN
100 N=NN
NP=N+1
DO 120 I=1,N
DO 110 J=1,N
A(I,J)=AA(I,J)
110 CONTINUE
120 CONTINUE
C NORMALIZE EACH ROW
DO 150 I=1,N
IF(A(I,J)*LE*0.0) K=0
IF(A(I,J)*LE*0.0) GO TO 1000
K=ALOG(A(I,J))
1000 DO 130 J=2,N
IF(A(I,J)*LE*0.0) GO TO 130
K=MAX(K, ALOG(A(I,J)))
130 CONTINUE
FACT(I)=E**(-K)
DO 140 J=1,N
A(I,J)=A(I,J)*FACT(I)
AA(I,J)=A(I,J)
140 CONTINUE
150 CONTINUE
C GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
DO 220 M=1,NM
TOP= ABS(A(M,M))
IMAX=M
MP=M+1
DO 160 I=MP,N
IF (TOP*GE*ABS(A(I,M))) GO TO 160
TOP= ABS(A(I,M))
IMAX=I
160 CONTINUE
IF (TOP*GT*0.0) GO TO 170
IT=1
GO TO 400
170 ITOP(M)=IMAX
IF (IMAX*LE*M) GO TO 190
DO 180 I=1,N
TEMP=A(M,I)
A(M,I)=A(IMAX,I)
A(IMAX,I)=TEMP
TEMP=AA(M,I)
180 CONTINUE
GO TO 220
400 RETURN
END
AA(M,I) = AA(IMAX,I)
AA(IMAX,I) = TEMP

CONTINUE

MP = M + 1
DO 210 I = MP, N
EM = A(I, M) / A(M, M)
A(I, M) = EM
IF (EM * NEQ 0.) GO TO 210
DO 200 J = MP, N
A(I, J) = A(I, J) * A(M, J) * EM
CONTINUE

CONTINUE

CONTINUE

ITOP(N) = N
IF (A(N, N) * NEQ 0.) GO TO 250
IT = 1
GO TO 400

C AT THIS POINT A = (L+I)*U WITH L AND U LOWER AND UPPER TRIANGULAR MATRICES.
C PROCESS RIGHT HAND SIDE

ITMAX = MAXIT
IT = 0
DO 260 I = 1, N
B(I) = BB(I) * FACT(I)
X(I) = 0
CONTINUE

DO 270 I = 1, N
IMAX = ITOP(I)
IF (IMAX * EQ I) GO TO 265
TEMP = B(I)
B(I) = B(IMAX)
B(IMAX) = TEMP
Z(I) = B(I)
BB(I) = B(I)
CONTINUE

C COMPUTE Z, WHERE L*Z = B

DO 300 I = 2, N
K = I - 1
SUM = 0
DO 290 J = 1, K
SUM = SUM + A(I, J) * Z(J)
CONTINUE

Z(I) = Z(I) - SUM
CONTINUE

C SOLVE FOR X, WHERE U*X = Z

DO 330 K = 1, N
I = NP - K
SUM = 0.
IF (K * EQ 1.) GO TO 320
IP = I + 1

39
DO 310 J=IP,N
SUM=SUM+A(I,J)*Y(J)
310 CONTINUE
320 Y(I)=(Z(I)-SUM)/A(I,I)
330 CONTINUE
C ITERATIVE IMPROVEMENT
DO 340 I=1,N
X(I)=X(I)+Y(I)
340 CONTINUE
IF (ITMAX .NE. 0) GO TO 345
IT=0
GO TO 400
345 IT=IT+1
ITMAX=ITMAX+1
IF (IT .GT. 1) GO TO 360
TOP=ABS(X(I))
DO 350 I=2,N
IF (ABS(X(I)) .GT. TOP) TOP=ABS(X(I))
350 CONTINUE
TEMP=TOP*EPS
360 TOP=ABS(Y(I))
DO 370 I=2,N
IF (ABS(Y(I)) .GT. TOP) TOP=ABS(Y(I))
370 CONTINUE
C TEST FOR CONVERGENCE
IF (TOP .LE. TEMP) GO TO 400
DO 390 I=1,N
SUM=0.
DO 380 J=1,N
SUM=SUM+A(I,J)*X(J)
380 CONTINUE
Z(I)=BB(I)-SUM
390 CONTINUE
GO TO 280
400 RETURN
END
**SMEQ1**


**Method:** Gauss elimination with partial pivoting (see Appendix A).

**Usage:**

```
DIMENSION A(n,n), B(n), Z(n)

CALL SMEQ1(A, B, Z, N, KR, NERROR)
```

where

- A = coefficient matrix
- B = constant vector
- Z = solution vector
- n = maximum permissible number of equations
- N = number of equations to be solved
NERRO R = an error indicator. NERROR is returned with a value of zero if A is non-singular and with a non-zero value if A is singular. In the latter case, the numerical value of NERROR is an aid in locating the point in the subroutine from which control is returned.

The original value of A is destroyed.

Comments: Several minor changes were made to the original version of SMEQ1:

1. The calls to subroutines PAGEHD and ZZBOMB were deleted.

2. Calls to ZZBOMB were replaced by RETURN statements.

3. The variable NERROR, originally an argument of subroutine ZZBOMB, was added to the argument list of SMEQ1.

These changes were made for the purpose of obtaining a "stand-alone" equation solver not requiring any supporting subroutines.
SUBROUTINE SMEQI (A,B,Z,N,KR,NERROR)
DIMENSION A(KR,N),B(1),Z(1)
DATA TOL/1.E-15/

SOLUTION OF LINEAR SIMULTANEOUS ALGEBRAIC EQUATIONS; A*Z = B.
GAUSS ELIMINATION METHOD; FORWARD SOLUTION TRANSFORMS ORIGINAL SYSTEM
INTO TRIANGULAR FORM; BACK SOLUTION THEN GIVES RESULT.
LARGEST PIVOTAL DIVISOR IS USED TO AVOID DIVISION BY SMALL NUMBERS.
The rows are interchanged when necessary to accomplish this.
If no pivot can be found exceeding 1.E-15, the matrix is considered
singular and the program stopped.
CODED BY CARL BODLEY; AUGUST 1968.
MODIFIED FOR CONTRACT NAS8-25922, MAY 1971.

SUBROUTINE ARGUMENTS
A = INPUT SQUARE MATRIX OF COEFFICIENTS; SIZE(N,N) *DESTROYED*
B = INPUT RIGHT HAND SIDE VECTOR; SIZE(N) *DESTROYED*
Z = OUTPUT RESULT VECTOR; SIZE(N)*
N = INPUT NUMBER OF EQUATIONS*
KR = INPUT ROW DIMENSION OF A IN CALLING PROGRAM*

IF (N *GT* 1) GO TO 5
NERROR = 1
IF (ABS(A(I,1)) *LE* TOL) RETURN
NERROR = 0
Z(1) = B(1)/A(1,1)
RETURN

FORWARD SOLUTION:
5 DO 25 L=1,N
AMAX = TOL
DO 10 I=1,N
IF (ABS(A(I,L)) *LT* ABS(AMAX)) GO TO 10
AMAX = A(I,L)
IMAX = I
10 CONTINUE
NERROR = 2
IF (ABS(AMAX) *LE* TOL) RETURN
DO 15 J=L,N
SAVE = A(IMAX,J)
A(IMAX,J) = A(L,J)
15 A(L,J) = SAVE/AMAX
SAVE = B(IMAX)
B(IMAX) = B(L)
B(L) = SAVE/AMAX
IF (L *EQ* N) GO TO 40
LP1 = L + 1
DO 25 I=LP1,N
DO 20 J=LP1,N
20 A(I,J) = A(I,J) - A(I,L)*A(L,J)
25 B(I) = B(I) - A(I,L)*B(L)

C BACK SOLUTION

40 Z(N) = B(N)
   NM1 = N - 1
   DO 45 L=1,NM1
       I = N - L
       Z(I) = B(I)
   DO 45 K=I,NM1
       J = K + 1
   45 Z(I) = Z(I) - A(I,J)*Z(J)

C

NERROR=0
RETURN
END
DPSIM


Method: Double precision Gauss elimination with partial pivoting (see Appendix A).

Usage:

    DOUBLE PRECISION A(n,n+1),DET
    .
    .
    .
    CALL DPSIM(A,N,DET)

where

A = coefficient matrix augmented by the constant vector added as an additional column

n = maximum permissible number of equations

DET = determinant of A

N = number of equations to be solved

The original value of A is destroyed, and the solutions are returned in the last row of A in reverse order: $A_{N1} = X_{N}$, $A_{N2} = X_{N-1}$, ..., $A_{NN} = X_1$
The array size in the DOUBLE PRECISION statement in DPSIM must conform to the following:

\[
\text{DOUBLE PRECISION } A(n, n+1), \text{DET, SIGN, AMAX, ABPIV, DIV, SS}
\]
SUBROUTINE DPSIhiAjN>DET)
DOUBLE PRECISION A(100,101), DET, SIGN, AMAX, ABPIV, DIV, SS
MM=N+1
SS=1.0DDO
DET=1.0DDO
NN=N+1
DO 8 K=1,NN
AMAX=C*CDU
DO 2 I=K,NN
SIGN=A(I,K)
ABPIV=DABS(SIGN)
IF (ABPIV-AMAX)2<2<1
1 AMAX=ABPIV
DIV=SIGN
IMAX=I
2 CONTINUE
IF (AMAX>3,3)4
3 DET=0.0DDO
RETURN
4 IF (IMAX-K)5<7,5
5 DO 6 J=K,MM
AMAX=A(IMAX,J)
A(IMAX,J)=A(K,J)
A(K,J)=AMAX
6 CONTINUE
SS=SS
7 KK=K+1
DO 8 I=KK,NN
AMAX=A(I,K)/DIV
DO 8 J=KK,MM
A(I,J)=A(I,J)-AMAX*A(K,J)
8 CONTINUE
DO 9 I=1,NN
9 DET=DET*A(I,I)
DET=DET*SS
JJ=N
DO 12 J=1,N
A(N,J)=A(J)MM/A(J,J)
KK=N
IF (KK)13,13,10
10 DO 11 I=JJ,NN
A(KK,MM)=A(KK,MM)*A(N,J)*A(KK,J)
11 KK=KK+1
12 JJ=JJ+1
13 RETURN
END
SHORT

Usage:

DIMENSION A(n,n + 1)
.
.
.
.
CALL SHORT(A,N)

where

A = coefficient matrix augmented by the constant vector attached as an additional column

n = maximum permissible number of equations

N = number of equations to be solved

The original value of A is destroyed, and the solution vector is returned in the last column of A.

The array size in the DIMENSION statement in SHORT must conform to the following:

DIMENSION A(n,n + 1)
SUBROUTINE SHORT(A,N)
DIMENSION A(100,101)
DO 2 I=1,N
DO 2 J=I,N
R*A(I,J+1)/A(I,I)
DO 1 K=2,N
1 A(K=1,J+1)=A(K,J+1)=R*A(K,I)
2 A(N,J+1)=R
RETURN
END
SOLVE1


**Method:** Gauss-Jordan elimination (see Appendix A). Zero divisors are avoided by row interchanges.

**Usage:**

```fortran
DIMENSION A(n,n + 1)
.
.
.
CALL SOLVE1 (N,A)
```

where

- \( A \) = coefficient matrix augmented by the constant vector attached as an additional column
- \( n \) = maximum permissible number of equations
- \( N \) = number of equations to be solved
The original value of A is destroyed, and the solution vector is returned in the first column of A.

The array sizes in the DIMENSION statement in SOLVE1 must conform to the following:

```
DIMENSION A(n,n + 1), B(n,n)
```

In the event A is singular, output is produced. If the output device designation is anything other than 3, the statement

```
WRITE (3,10)
```

must be modified accordingly.

**Comment**: The original version is a program. Only those changes necessary to convert it to a subroutine were made.
SUBROUTINE SOLVE1(N, A)
SOLUTION OF SIMULTANEOUS EQUATIONS BY GAUSS-JORDAN ELIMINATION
ZERO DIAGONAL ELEMENTS ARE PERMITTED.
THE ORIGINAL ELEMENTS OF A, THE AUGMENTED MATRIX, ARE LOST.
THE SOLUTIONS ARE STORED IN THE FIRST COL OF THE AUGMENTED MATRIX.

DIMENSION A(100,101), B(100,100)
M = N + 1
5 IF(A(1,1)) 11, 6, 11
6 K = M-1
50 9 I=2,K
IF(A(I,1)) 7, 9, 7
7 DO 8 J=1,M
   TEMP = A(I,J)
   A(I,J) = A(I,J)
   A(I,J) = TEMP
8 GO TO 11
9 CONTINUE
WRITE(3,10)
10 FORMAT(/// 19H NO UNIQUE SOLUTION)
   GO TO 18
11 DO 12 J=2,M
   DU 12 I=2,N
12 B(I=1,J=1) = A(I,J)*A(1,J)*A(I,J)/A(1,1)
   DO 13 J=2,M
13 B(N,J=1) = A(1,J)/A(1,1)
   M = M-1
   DU 14 J=1,M
   DU 14 I=1,N
14 A(I,J) = B(I,J)
   IF(M=1) 5,18,5
18 RETURN
END
Source: Written by T. S. Wu, Department of Civil Engineering, Memphis State University

Method: Gauss-Jordan elimination (see Appendix A)

Usage:

DIMENSION A(n,n + r)

.
.
.

CALL WUIGJ(A,N,NSOL,DETA)

where

A = coefficient matrix augmented by the constant vectors attached as additional columns

n = maximum permissible number of equations

r = maximum permissible number of constant vectors

N = number of equations to be solved

NSOL = number of constant vectors to be solved. If NSOL is specified as zero, the inverse of A is computed and returned as A.

DETA = determinant of A
The original value of $A$ is destroyed, and the solution vectors replace the corresponding constant vectors.

The DIMENSION statement in WUIGJ must conform to the following:

\[
\text{DIMENSION } A(n, n + r)
\]
SUBROUTINE HUIUJ(A, N, NSOL, DETAT)
DIMENSION A(100, 105)
DETA=1.
NCOL=N+NSOL
DO 10 K=1, N
   IF (A(K, K)) EQ 0.0
      30 DETA=G
      RETURN
      20 DETA=DETA*A(K, K)
      DO 40 J=1, NCOL
         IF (J=K) GO TO 50
         50 A(K, J) = A(K, J)/A(K, K)
      CONTINUE
      A(K, K) = 1./A(K, K)
      DO 10 I=1, N
         IF (I=K) GO TO 60
         60 DO 70 J=1, NCOL
            IF (J=K) GO TO 80
            80 A(I, J) = A(I, J) = A(I, K)*A(K, J)
      CONTINUE
      A(I, K) = A(I, K)*A(K, K)
      10 CONTINUE
      RETURN
END
WUGJN

Source: Written by T. S. Wu, Department of Civil Engineering, Memphis State University

Method: Gauss-Jordan elimination (see Appendix A)

Usage:

DIMENSION A(n,n + r)
.
.
.

CALL WUGJN (A,N,M)

where

A = coefficient matrix augmented by the constant vectors attached as additional columns

n = maximum permissible number of equations

r = maximum permissible number of constant vectors

N = number of equations to be solved

M = number of constant vectors to be solved

The original value of A is destroyed, and the solution vectors replace the corresponding constant vectors.
The DIMENSION statement in WUGJN must conform to the following:

    DIMENSION A(n,n + r)
CALL WUGJN(A,100,101)
SUBROUTINE WUGJN(A,N,M)
DIMENSION A(100,101)
DO 100 K=1,N
K1=K+1
DO 105 J=K1,M
105 A(K,J)=A(K,J)/A(K,K)
IF (K1=M) 110,110,100
110 DO 100 I=1,N
IF (K=I) 120,120,100
120 DO 130 J=K1,M
130 A(I,J)=A(I,J)/A(I,K)*A(K,J)
100 CONTINUE
RETURN
END
SLVE


Method: Gauss-Jordan elimination (see Appendix A). To prevent division by zero or very small numbers, subsequent equations are added to the current one until the divisor becomes larger than some pre-assigned value.

Usage:

```
DIMENSION A(n,n), U(n)

... ...

CALL SLVE(A,U,NM,EPS)
```

where

A = coefficient matrix

U = Constant vector on entry

U = solution vector upon return

n = maximum permissible number of equations
NM = number of equations to be solved

EPS = smallest acceptable value for a divisor in the elimination process

The original value of A is destroyed.

The DIMENSION statement in SLVE must conform to the following:

DIMENSION A(n,n), U(n)
SUBROUTINE SLVE(A,U,NM, EPS)

REFERENCE = NUMERICAL AND MATRIX METHODS IN STRUCTURAL MECHANICS, BY P. C. WANG

DIMENSION A(100,100), U(100)
DO 9015 I=1,NM
  K=I
  IF (I=NM) 9021, 9007, 9021

  TEST TO SEE WHETHER DIAGONAL ELEMENT IS ZERO

9021 IF (ABS(A(I,I))=EPS) 9006, 9006, 9007
  ADD SUBSEQUENT EQUATIONS TO THE CURRENT ONE

9006 K=K+1
  U(I)=U(I)+U(K)
  DO 9023 J=1,NM
    A(I,J)=A(I,J)+A(K,J)
  GO TO 9021

9007 DIV=A(I,I)
  U(I)=U(I)/DIV

  DIVIDE ALL THE ELEMENTS OF I-TH EQ. BY A(I,I)

  DO 9009 J=1,NM
    A(I,J)=A(I,J)/DIV

  REDUCE THE I-TH ELEMENT OF THE OTHER EQUATIONS TO ZERO

  DO 9015 MM=1,NM
    DELT=A(MM,I)
    IF (ABS(DELT)=EPS) 9015, 9015, 9016
  9016 IF (MM=I) 9010, 9015, 9010
  9010 U(MM)=U(MM)-U(I)*DELT
  DO 9011 J=1,NM
  9011 A(MM,J)=A(MM,J)-A(I,J)*DELT
  CONTINUE
  RETURN
END
GASSIM


Method: Gauss-Jordan elimination with partial pivoting (see Appendix A)

Usage:

```
DIMENSION A(n,n),B(n,r)
.
.
.
CALL GASSIM(A,N,NN,DET,B)
```

where

- $A = \text{coefficient matrix}$
- $B = \text{matrix of right-hand side constants}$
- $n = \text{maximum permissible number of equations}$
- $r = \text{maximum permissible number of right-hand side constant vectors}$
- $N = \text{number of equations to be solved}$
NN = number of constant vectors to be solved

DET = determinant of A

The solution vectors are returned in the B array, replacing the corresponding constant vectors.

The original value of A is destroyed.

The DIMENSION statement in GASSIM must conform to the following:

DIMENSION A(n,n), B(n,r)
SUBROUTINE GASSIM(A,N,NN,DET,B)
DIMENSION A(100,100),B(100,5)
EQUIVALENCE (D,ID)(E,IE)
DATA IG, 2500,CCOO/
DET=1
DO 2 K=1,N
IF (K.EQ.N) GO TO 3
TEST=ABS(A(K,K))
KP1=K+1
L=K
DO 4 I=KP1,N
IF (TEST.GE.ABS(A(I,K))) GO TO 4
TEST=ABS(A(I,K))
L=I
4 CONTINUE
IF (L.EQ.K) GO TO 3
DO 5 J=1,N
TEMP=A(L,J)
A(L,J)=A(K,J)
A(K,J)=TEMP
5 CONTINUE
DO 15 J=1,NN
TEST=ABS(A(J,K))
L=J
DO 4 CONTINUE
IF (L.EQ.K) GO TO 3
DO 6 J=K+1,N
A(K,J)=A(J,K)
6 CONTINUE
DET=-DET
3 DET=DET*A(K,K)
A(K,K)=1/A(K,K)
IF (K.EQ.N) GO TO 19
DO 6 J=KP1,N
A(K,J)=A(K,J)*A(K,K)
6 CONTINUE
19 DO 16 J=1,NN
B(K,J)=B(K,J)*A(K,K)
16 CONTINUE
IF (I.EQ.K) GO TO 7
FAC=A(I,K)
DO 8 J=KP1,N
E=A(I,J)
D=E+FAC*A(K,J)
IF (IABS(E)-IABS(ID).*GT.*IG) C=0.
A(I,J)=D
8 CONTINUE
DO 18 J=1,NN
B(I,J)=B(I,J)+FAC*B(K,J)
18 CONTINUE
7 CONTINUE
2 CONTINUE
RETURN
C*************************************************************************
LAST CARD IN SUBROUTINE*************************************************************************
END
GJR

Source: University of Alabama at Huntsville Research Institute

Method: Gauss–Jordan elimination with partial pivoting
(See appendix A)

Usage:

.DIMENSION A(n,n + r), JC(1)

.

.

.

CALL GJR(A, NC, NR, N, L, NN, JC)

where

A = the coefficient matrix augmented by the constant vectors attached as additional columns

JC = an error indicator. If JC(1) is returned with a value of N, the number of equations to be solved, a solution is obtained. If JC(1) is returned as anything else, the matrix is singular, no solution is obtained, and control is returned to statement NN in the calling program.
n = maximum permissible number of equations

r = maximum permissible number of constant vectors

NC = value of n + r in the DIMENSION statement in the calling program

NR = the value of n in the DIMENSION statement in the calling program

N = number of equations to be solved

L = N plus the number of constant vectors to be solved

NN = statement number of the statement in the calling program to which return is to be made in the event of a singularity.

The original value of A is destroyed, and the solution vectors replace the corresponding constant vectors.
SUBROUTINE 3JR (A, NC, NR, NC, LN, NN, JC)
DIMENSION A(NR, NC), JC(1)

SEARCH FOR PIVOT ROW

DO 91 I = 1, N
M = I
IF (I .EQ. N) GO TO 60
X = A(I, I)
DO 30 J = 1, N
IF (X .GT. ABS(A(J, I))) GO TO 30
X = ABS(A(J, I))
K = J
30 CONTINUE
IF (K .EQ. I) GO TO 60

INTERCHANGE ROW I AND ROW K

DO 50 J = M, L
X = A(I, J)
A(I, J) = A(K, J)
A(K, J) = X
50

TEST FOR SINGULARITY

60 IF (ABS(A(I, I)) .GT. 0.) GO TO 72

MATRIX IS SINGULAR

150 JC(1) = I - 1
RETURN
72 X = A(I, I)
A(I, I) = 1.

NORMALIZATION OF THE I-TH ROW

DO 80 J = M, L
A(I, J) = A(I, J) / X
80

TEST OVERFLOW SWITCH: IF ON, RETURN NEGATIVE VALUE OF I IN JC(1)

CALL OVERFL(IFL)
IF (IFL .EQ. 1) GO TO 150
80 CONTINUE

REDUCTION OF ALL REMAINING ROWS

DO 91 K = 1, N
IF (K .EQ. I) GO TO 91
91

RETURN
\[ x = a(k, i) \]
\[ a(k, i) = 0 \]
\[ \text{DO 90 } j = m, l \]
\[ a(k, j) = a(k, j) + x \cdot a(l, j) \]

TEST OVERFLOW SWITCH IF ON,
RETURN NEGATIVE VALUE OF I IN JC(1)

CALL OVERFLO(FL)
IF(FL.EQ.1) GO TO 150
90 CONTINUE
91 CONTINUE
JC(1) = N
RETURN
END
SIMULT

Source: Written by J. E. Key, George C. Marshall Space Flight Center

Method: Gauss-Jordan elimination (see Appendix A). The technique used to select the pivotal elements minimizes the number of non-zero elements generated.

Usage:

COMMON/PARAM/NAA
COMMON/A/A(n,m)
COMMON/ICC/ICC(n),ICOL(n,m),PLOAD(n)
COMMON/INDEX/NU,NELE,NNOD,NA,NI,NLNOD,
    NUDIS,NUP2,KK,NNN
COMMON/ZTEST/ZTEST

DOUBLE PRECISION A,PLOAD
    *
    *
    *
    CALL SIMULT

where

A = matrix containing the non-zero elements of the coefficient matrix. Each row is compacted to the left; the remaining elements (if any) in the row are zeros.
n = maximum permissible number of equations

m = maximum permissible number of elements in any row of the compacted coefficient matrix

ICC = a vector giving the number of non-zero coefficients in each row of A

ICOL = a matrix giving the position of each non-zero element in the coefficient matrix. For example, if ICOL(6,4) has a value of 7, the element occupying row 6 and column 7 in the full matrix is non-zero. The second subscript of ICOL means that the element being referenced is the fourth non-zero coefficient in the row.

PLOAD = constant vector

NU = number of equations to be solved

ZTEST = a small number used to test for ill-conditioning

NAA = value of m in the COMMON statements in the calling program. Arrays A and ICOL will possibly expand during the solution, but there is no way to predict the maximum row dimension in advance. This will be a function of the
pivotal elements selected. If NAA has been specified with too small a number, an error message will be printed. The only guide to the selection of NAA that can be given here is to make it at least as large as the initial number of elements in any row of the compacted coefficient matrix.

The original value of A is destroyed, and the solution vector is returned in the first column of A.

This subroutine requires subroutine RITE, which is called in the event A is singular, and it produces output. The output device designation is specified in both SIMULT and RITE in the statement

INTEGER OUT/108/

The array sizes in the COMMON and DIMENSION statements in SIMULT must conform to the following:

COMMON/A/A(n,m)
COMMON/ICC/ICC(n),ICOL(n,m),PLOAD(n)
DIMENSION IRC(n)

Comment: The only variable in the labeled COMMON block "INDEX" that is used in SIMULT is NU, the number of equations to be solved. The user may wish to remove the other variables from this statement in both the calling program and in SIMULT.
SUBROUTINE SIMULT

C -NAA = MAXIMUM NUMBER OF COLUMNS IN EQUATION SOLVER
C NU = NUMBER OF ROWS IN SYSTEM OF EQUATIONS TO BE SOLVED
C PLOAD = CONSTANT VECTOR OF EQUATIONS TO BE SOLVED
C A = MATRIX CONTAINING NON-ZERO COEFF* OF SYSTEM
C ICC = NUMBER OF NON-ZERO COEFF* IN ANY GIVEN ROW OF A
C ICOL = RECORDS POSITION OF NON-ZERO COEFF* IN SYSTEM
C A(I,J) = SOLUTION VECTOR RETURNED IN FIRST COL* OF A.

COMMON/PARAM!/ NAA
DOUBLE PRECISION C*X;A>PLOAD
INTEGER PIVROW,PIVCOL,OPROW
COMMON/A/ A(50,10)
COMMON/ICC/ ICC(50),ICOL(50,10),PLOAD(50)
COMMON/INDEX/NU,NELE,NNOD,NA,NI,NLNGD,NUDIS,NUP2, KK, NNN
COMMON/ZTEST/ZTEST
DIMENSION IRC(2000)
INTEGER OUT/108/

C INITIALIZE ROW ENTRY COUNTER
DO 99 I=1,NU
99 IRC(I)=0

C COUNT ROW AND COLUMN ENTRIES
JELE=0
DO 101 I=1,NU
DO 102 J=1,NAA
ICC(I)=ICOL(I,J)
IF(ICC(I).EQ.0) GO TO 101
JELE=JELE+1
ICC(I)=J
IRC(IC)=IRC(IC)+1
102 CONTINUE
101 CONTINUE
JCOL=0
JELIM=0
DO 800 I=1,NU
IF(ICC(I) GT JCOL) JCOL=ICC(I)
800 CONTINUE
DO 66 LKJ=1,NU

C SELECT ROW WITH MINIMUM ENTRIES
IK=100000
DO 103 I=1,NU
IF(ICC(I) GT IK OR ICC(I) LE 0) GO TO 103
PIVROW=I
IK=ICC(I)
103 CONTINUE

C SELECT SMALLEST AVAILABLE COLUMN FROM PIVROW
IK=100000
IC=ICC(PIVROW)
DO 104 I=1,IC
II=ICOL(PIVROW,I)
IF(IRC(II) GT IK OR IRC(II) LE 0) GO TO 104

72
PIVCOL = II
IK=IRC(I)
IY=I

104 CONTINUE
C NORMALIZE PIVROW
X=A(PIVROW, IY)
IC=ICC(PIVROW)
DO 5 J=1, IC
      A(PIVROW, J)=A(PIVROW, J)/X
      PLOAD(PIVROW)=PLOAD(PIVROW)/X
C SELECT ROWS THAT CAN BE OPERATED ON
DO 106 I=1, NU
      IF(I.EQ.PIVROW.AND.IC(PIVCOL).EQ.1) GO TO 107
      IF(I.EQ.PIVROW) GO TO 106
      IC=ABS(IC(I))
      DO 105 J=1, IC
      IF(ICOL(I, J)=PIVCOL) 105, 77, 106
C IF YOU CAN GET TO THIS POINT OPROW CONTAINS PIVOTAL ELEMENT
77 OPROW = I
      JKOP=1
      JKPI=1
      C=A(OPROW, J)
      X=PLOAD(PIVROW)+C+PLOAD(OPROW)
      PLOAD(OPROW)=X
79 CONTINUE
      IF(ICOL(OPROW, JKPI).EQ.0) GO TO 106
      IF(ICOL(OPROW, JKOP).EQ.0) GO TO 80
      IF(ICOL(OPROW, JKPI)=ICOL(OPROW, JKOP)) 80, 81, 82
C OPROW DOES NOT CONTAIN THIS ELEMENT: ADD ELEMENT TO OPROW
80 ICC(I)=ICC(I)+1
      IF(ICC(I).LE.0) ICC(I)=ICC(I)+2
      IC=ABS(ICC(I))
      IF(I.GT.ICOL) JCOL=II
      IF(I.EQ.NAA) GO TO 85
      WRITE(OUT, 400) II
      STOP
85 JKL=JKOP+1
90 CONTINUE
IX=II+1
A(OPROW, IX)=A(OPROW, IX)
ICOL(OPROW, IX)=ICOL(OPROW, IX)
II=IX
      IF(I.EQ.JKL) GO TO 90
      X=A(PIVROW, JKPI)*C
      A(OPROW, JKOP)=X
      ICOL(OPROW, JKOP)=ICOL(PIVROW, JKPI)
      IX=ICOL(OPROW, JKOP)
      IRC(IX)=IRC(IX)+1
      GO TO 83
PIVROW AND OPROW CONTAIN THIS ELEMENT. SHIFT BOTH AND OPERATE ON OPROW.

\[ \text{IF}(\text{IX} \neq \text{PIVCOL}) \text{GO TO 11} \]
\[ \text{X} = a(\text{PIVROW}, \text{JKPI}) \times c + a(\text{OPROW}, \text{JKOP}) \]
\[ a(\text{OPROW}, \text{JKOP}) = x \]

TEST OPROW TO SEE WHAT WAS ELIMINATED

\[ \text{ATEST} = \text{ABS}(X) \times \text{ZTEST} \]
\[ \text{IF}(\text{ATEST} \geq 0.0) \text{GO TO 83} \]

\[ \text{IRC}(\text{IX}) = \text{IRC}(\text{IX}) - 1 \]
\[ \text{JELIM} = \text{JELIM} + 1 \]
\[ \text{ICC}(\text{OPROW}) = \text{ICC}(\text{OPROW}) - 1 \]
\[ \text{IF}(\text{ICC}(\text{OPROW})) = 0 \times 140, 141, 142 \]

CONTINUE

\[ \text{IB} = 1 \]

\[ \text{DO 10 M = 1,NU} \]
\[ \text{II} = \text{ICC}(M) \]
\[ \text{IF}(\text{II} \geq \text{IB}) \text{IB} = \text{IB} \]

CONTINUE

\[ \text{IB} = \text{IB} + 1 \]

WRITE(OUT, 999) LKJ, PIVROW, PIVCOL, OPROW

\[ \text{FORMAT}(1 \times \text{M} \times \text{SINGULAR', 'NO CYCLES COMPLETED' =1,15,3' PIVROW='I5', 'PIVCOL='I5', 'OPROW='I5') \]

WRITE(OUT, 801) (M, IRC(M), ICC(M), M = 1,NU)

\[ \text{FORMAT}(1 \times \text{COL', 'ROW NO', 'NO. COL', 'ENTRIES', 'NO. ROW ENTRIES' =1/16(I7,215)) \]

WRITE(OUT, 6)

CALL RITE(1,NU,IB, NAA, A, ICOL)

WRITE(OUT, 200)

CALL RITE(2,NU,IB, NAA, A, ICOL)

\[ \text{FORMAT}(1 \times \text{COEFFICIENT MATRIX'})) \]

STOP

CONTINUE

\[ \text{ICC}(\text{OPROW}) = \text{ICC}(\text{OPROW}) + 2 \]

\[ \text{IX} = \text{ABS}(\text{ICC}(\text{OPROW})) \]

\[ \text{DO 131 NK = JKOP, IX} \]
\[ a(I,NK) = a(I,NK+1) \]

\[ \text{131 ICOL}(I,NK) = \text{ICOL}(I,NK+1) \]
\[ \text{IX} = \text{IX} + 1 \]
\[ a(I,IX) = 0 \]
\[ \text{ICOL}(I,IX) = 0 \]
\[ \text{JKPI} = \text{JKPI} + 1 \]

GO TO 79

\[ \text{83 JKPI} = \text{JKPI} + 1 \]

PIVROW DOES NOT CONTAIN THIS ELEMENT. SHIFT OPROW AND CONTINUE.

\[ \text{JKOP} = \text{JKOP} + 1 \]

GO TO 79

CONTINUE

CONTINUE
107 CONTINUE
C ELIMINATES PIVROW AND PIVCOL FROM BEING CONSIDERED AGAIN:
   ICC(PIVROW) = ICC(PIVROW)
   IRC(PIVCOL) = IRC(PIVCOL)
66 CONTINUE
C UNSCRAMBLE AND STORE SOLUTION IN FIRST COLUMN OF A:
DO 350 I = 1, NU
   II = ICol(I,1)
   A(II,1) = PLOAD(I)
350 CONTINUE
400 FORMAT ('OII = ',I4,' THIS IS GREATER THAN VALUE OF NAA ')
RETURN
END
SUBROUTINE RITE(IDUM, NR, NC, MC, A, ICOL)
DOUBLE PRECISION A
DIMENSION A(50, 10), ICOL(50, 10)
INTEGER OUT, ICOL
IPRINT = 12
IF (IDUM .NE. 1) IPRINT = 30
IPR = IPRINT = 1
DO 35 K = 1, NC, IPRINT
MAX = K + IPR
IF (MAX .GT. NC) MAX = NC
IF (K .NE. 1) WRITE (OUT, 103)
IF (IDUM .EQ. 1) GO TO 45
WRITE (OUT, 101) (I, I = K, MAX)
DO 20 J = 1, NR
20 WRITE (OUT, 104) J, (ICOL(J, I), I = K, MAX)
GO TO 35
45 WRITE (OUT, 102) (I, I = K, MAX)
DO 40 J = 1, NR
40 WRITE (OUT, 105) J, (A(J, I), I = K, MAX)
35 CONTINUE
RETURN
101 FORMAT (6X, 30I4)
102 FORMAT (6X, 12I10)
103 FORMAT (I1)
104 FORMAT (', ', 30I4)
105 FORMAT (', ', 15, 12G10.3)
END
SOLVE3


Method: Double precision Gauss elimination (see Appendix A)

Usage:

COMMON /A/ A(n,h + 1), R(n)
COMMON NEQ, ITEM(n,h + 1)
DOUBLE PRECISION A, R
.
.
.
CALL SOLVE3

where

A = upper triangular portion of the symmetric coefficient matrix stored as a compacted rectangular array. All of the zero elements in a given row are shifted to the right side of the row.

n = maximum permissible number of equations

h = maximum permissible number of non-zero elements in any row
R = constant vector on entry

R = solution vector upon return

NEQ = number of equations to be solved

ITEM = a "pointer" matrix which gives the location (in terms of the full \( n \times n \) coefficient matrix) of the non-zero elements. For example, if ITEM(6,4) has a value of 7, the element occupying row 6 and column 7 in the full matrix is non-zero. The second subscript of ITEM simply means that the element being referenced is the fourth non-zero coefficient in the row.

Each row of ITEM is searched until a zero element is encountered. Therefore, the \((w + 1)\)st column of ITEM, where \( w \) is the actual minimum number of non-zero elements in any row, must consist of zeros. Note that ITEM must contain \( h + 1 \) columns, even if the excess columns contain zeros.

The original value of A is destroyed.

The array sizes in the COMMON and DIMENSION statements in SOLVE3 must conform to the following:
COMMON /A/ A(n,h + 1), R(n)
COMMON NEQ, ITEM(n,h + 1)
DIMENSION IMET(n)

If $h + 1$ has a value of anything other than 4, the statement

$$\text{NT} = 4$$

must be modified accordingly.

In the event of an input error, SOLVE3 produces output. The output device designation is specified in the statement

```
INTEGER OUT/108/
```

**Note:** Several changes were made to the original version of this subroutine:

1. The name was changed from SOLVE to SOLVE3.

2. Variables not used in this subroutine were removed from the COMMON statements.

3. A DIMENSION statement was added.

4. The output device designation was changed from a constant to a variable.

5. A conversion to double precision arithmetic was made by the addition of the statement

   ```
   DOUBLE PRECISION A, R, TEMP
   ```
SUBROUTINE SOLVE3
SOLVES SPARSE MATRIX

COMMON/A/ A(3500, 4), R(3500)
COMMON NEQ, ITEM(3500, 4)
DIMENSION IMET(3500)
DOUBLE PRECISION A, R, TEMP
NT = 4
INTEGER OUT/108/

IF NEQ NEGATIVE SKIP
FORMATION OF FULL ITEM MATRIX

IF (NEQ < 0) GO TO 360

OTHERWISE WORK THROUGH
EQUATIONS AND FILL ITEM AS NEEDED

DO 220 M = 1, NT
220 IMET(M) = ITEM(1, M)
DO 340 N = 2, NEQ
DO 280 M = 1, NT
IF (IMET(M) = N + 1) 225, 280, 225
225 DO 240 L = 1, NT
IF (ITEM(N, L)) 230, 260, 230
230 IF (ITEM(N, L) = IMET(M)) 240, 280, 240
240 CONTINUE
WRITE(OUT/100) N, 100 FORMAT(43H ALLOWABLE SPACE EXCEEDED IN EQUATION TABLE, I4)
STOP
260 ITEM(N, L) = IMET(M)
280 CONTINUE
300 DO 320 M = 1, NT
320 IMET(M) = ITEM(N, M)
340 CONTINUE
360 NEQ = IABS(NEQ)
NEQM = NEQ - 1

LOOP ON EQUATIONS

DO 520 I = 1, NEQM
MODIFY RHS VECTOR

R(I) = R(I) / A(I, 1)

LOOP ON ROW TO BE ELIMINATED

DO 460 M = 2, NT
IN = ITEM(I, M)
IF(IN)365, 480, 365

SEEK APPROPRIATE ROWS

365 DO 380 N=1, NT
   IA=ITEM(IN, N)
   IF(IA)370, 400, 370
370 IMET(IA)=N
380 CONTINUE
400 CONTINUE
   TEMP=A(I, M)/A(I, 1)

LOOP ON COLUMN TO BE ELIMINATED

DO 420 N=1, NT
   IA=ITEM(IN, N)
   IF(IA)405, 440, 405
405 IF(IA=IN)420, 410, 410
410 IM=IMET(IA)

MODIFY TERM OF MATRIX

   A(IN, IM)=A(IN, IM)-TEMP*A(I, N)
420 CONTINUE

MODIFY LOAD VECTOR

   R(IN)=R(IN)-R(I)*A(I, M)
460 CONTINUE
480 CONTINUE

RESET ROW FOR BACK-SUBSTITUTION

DO 500 M=2, NT
   A(I, M)=A(I, M)/A(I, 1)
500 CONTINUE
520 CONTINUE
   R(NEQ)=R(NEQ)/A(NEQ, 1)

BACK-SUBSTITUTE

DO 560 IB=1, NEQM
   I=NEQ-IB
   DO 540 M=2, NT
      J=ITEM(I, M)
      IF(J)540, 560, 540
540 R(I)=R(I)-A(I, M)*R(J)
560 CONTINUE
RETURN
END
SOLVE2


Method: Gauss-Seidel iteration with over-relaxation (see Appendix A).

Usage:

COMMON/CONTR/TITLE(20),NP,NE,NDF,NCN,
NLD,NMAT,NEQ,LI,NT4

COMMON CORD(100,2),NOP(200,4),IMAT(200),
ORT(25,2),NBC(25),NFIX(25),R(n),
A(n,b),ITEM(n,b+l),DIS(n)
.
.
.

CALL SOLVE2

where

NEQ = number of equations to be solved

R = constant vector on entry

R = solution vector upon return
A = the full coefficient matrix stored as a compacted rectangular array. All of the zero elements in a given row are shifted to the right side of the row. The diagonal terms in the full matrix are the first elements in each row of the compacted matrix, followed by the remaining terms in their original order.

n = maximum permissible number of equations

ITEM = a "pointer" matrix which gives the location (in the full n x n coefficient matrix) of the non-zero elements. For example, if ITEM(6,4) has a value of 7, the element occupying row 6 and column 7 in the full matrix is non-zero. The second subscript of ITEM simply means that the element being referenced is the fourth non-zero coefficient in the row.

b = maximum permissible number of non-zero elements in any row

The array sizes in the COMMON statement in SOLVE2 must conform to the following:

```
COMMON CORD(100,2),NOP(200,4),IMAT(200),
ORT(25,2),NBC(25),NFIX(25),R(n),
A(n,b),ITEM(n,b + 1),DIS(n)
```
If \( b+1 \) is anything other than 20, the statement

\[
NT = 20
\]

must be modified accordingly.

The original value of \( A \) is retained, but \( ITEM \) is modified. Output is produced by this subroutine. The output device designation is specified in the statement

\[
\text{INTEGER OUT/108/}
\]

Each row of the pointer matrix \( ITEM \) is searched until a zero element is encountered. Therefore, the \((b+1)\)st column of \( ITEM \) must consist of zeros, where \( b \) is the actual number of non-zero elements in any row.

A tolerance and over-relaxation factor are built into subroutine SOLVE2. These may be varied by modifying the statements

\[
\text{TOLER = .1E-6}
\]

\[
\text{RELAX = 1.8}
\]

accordingly.

The maximum number of iterations is given in the statement

\[
\text{NCYC = 15000}
\]

Comments: Several minor changes were made to the original version of this subroutine:

1. The name was changed from SOLVE to SOLVE2.
2. The statement
   \[ \text{NCYC} = \text{NEQ}/2 \]
   was changed to
   \[ \text{NCYC} = 15000 \]

3. The statement
   \[ \text{DIS}(N) = \text{DIS}(N)+\text{RLAX} \times \text{DX} \]
   was changed to
   \[ \text{DIS}(N) = \text{DIS}(N)+\text{RELAX} \times \text{DX} \]
   (this was a typographical error in the original version).

4. The statement
   \[ \text{TOLER} = .1 \times 10^{-3} \]
   was changed to
   \[ \text{TOLER} = .1 \times 10^{-6} \]

The user may wish to streamline this subroutine somewhat by deleting the extraneous entries from the COMMON statements.
SUBROUTINE SOLVEZ
SUBROUTINE FOR ITERATIVE SOLUTION OF EQUATIONS
INTEGER OUT/108/
COMMON/CONTR/TITLE(20),NP,NE,NB,NDF,NCN,LD,NMAT,NEG,LI,NT4
COMMON CRD(lOO,2).NOP(2O0,4),IMAT(2OO),ORT(25,2),NBC(25),NFiX(25)
1R(2OO),A(2O0,2O),ITEM(2O0,2O),DIS(2O0)
NT=2O

BUILT IN RELAXATION AND TOLERANCE
TOLER=1E-6
RELAX=1.8

NEGATIVE NEG SKIPS SETTING UP
OF INITIAL DATA

IF(NEG.LT.O) GO TO 310
DO 300 N=1,NEQ
DO 250 M=1,NT
IF(ITEM(N,M).NE.0) GO TO 250
ITEM(N,1) CONTAINS COUNT OF
NUMBER OF OFF DIAGONAL TERMS
ITEM(N,1)=M=1
GO TO 260
250 CONTINUE
260 CONTINUE
300 CONTINUE
310 NEG=IABS(NEG)

SET MAX NO OF CYCLES
NCYC=15000
IF(NCYC.LT.25) NCYC=25

UNKNOWN S ARE SET TO ZERO

DO 320 N=1,NEQ
IF(A(N,1).NE.0) A(N,1)=1/A(N,1)
32O DIS(N)=O

LOOP ON CYCLES

DO 500 NC=1,NCYC
SUM=O.
SUMD=O.

THEN ON EQUATIONS

DO 450 N=1,NEQ
FX=R(N)
NUM=ITEM(N,1)
DO 330 M=2,NUM
L=ITEM(N,M)
330 FX=FX=AIN(M)*DIS(L)

FX IS TOTAL UNBALANCE OF RHS
DX IS THE CHANGE

DX=AIN,M)*FX-DIS(N)
DIS(N)=DIS(N)+RELAX*DX

SUM AND SUMD ARE CONVERGENCE
PARAMETERS

SUM=SUM+ABS(DX)
SUMD=SUMD+ABS(DIS(N))
450 CONTINUE

SKIP OUT OF LOOP IF CONVERGED

ND=NC
IF(SUM<LT*SUMD*TOLER) GO TO 550
500 CONTINUE

MOVE FINAL RESULTS TO R

550 DO 600 N=1,NEQ
600 R(N)=DIS(N)

PRINT LAST VALUE OF SUM ETC

WRITE(OUT,10)ND,SUM,SUMD
10 FORMAT(18HO LAST CYCLE NO== ,II0
1/ 18H (SN=SN+1)/SN = E10.3
2/ 18H SN = E10.3)
RETURN
END
GSEID


Method: Gauss-Seidel iteration (see Appendix A).

Usage:

DIMENSION A(n,n + l), X(n)
.
.
.

CALL GSEID(N, MAX, A, X, TOL, INDEX)

where

A = coefficient matrix augmented by the constant vector attached as an additional column

X = trial solution vector on entry

X = solution vector upon return

n = maximum permissible number of equations

N = number of equations to be solved

MAX = maximum number of iterations to be performed
TOL = a convergence criterion. If the difference between the value of each unknown and its value in the previous cycle is less than TOL, convergence is considered to have been achieved.

INDEX = a convergence indicator. INDEX is returned with a value of 2 if convergence has been achieved and as 1 if it has not.

The original value of A is retained.

The array sizes in GSEID must conform to the following:

DIMENSION A(n,n + 1),X(n)

Comment: The original version is a program. Statements necessary to convert it to a subroutine were added.
SUBROUTINE GSEID(N, MAX, A, X, TOL, INDEX)

N = NO. OF EQUATIONS
MAX = LIMIT TO ITERATIONS
TOL = TOLERANCE
INDEX IS RETURNED AS 1 IF NO CONVERGENCE

DIMENSION A(100,101), X(100)
INDEX=2
NP1=N+1
DO 30 I=1,MAX
DIFF=G.
DO 20 J=1,N
SAVE=X(J)
X(J)=A(J,NP1)
DO 21 K=1,N
IF(J=K)22,22,22
22 X(J)=X(J)-A(J,K)*X(K)
21 CONTINUE
X(J)=X(J)/A(J,J)
IF(DIFF=ABS(X(J)-SAVE))23,23,20
23 DIFF=ABS(X(J)-SAVE)
20 CONTINUE
IF(TOL=DIF130,40,40
30 CONTINUE
INDEX=1
40 RETURN
END
2.2 In-core, Non-banded Inversion Subroutines
AINVER

Method: Gauss-Jordan elimination (see Appendix A).

Usage:

DIMENSION A(m,m)

. 

. 

. 

CALL AINVER(A,N,N1)

where

A = matrix to be inverted

m = maximum permissible order of A plus one

N = order of matrix to be inverted

N1 = value of m in the DIMENSION statement in the calling program

The original value of A is destroyed, and the inverse is returned in A.
SUBROUTINE AINV(A,N,N11)
DIMENSION A(N11,N)
INTEGER P
P=N+1
A(1,P)=1.
DO 4 I=2,N
4 A(I,P)=0.
DO 40 K=1,N
DO 5 J=1,N
5 A(P,J)=A(1,J+1)/A(1,1)
DO 10 I=2,N
DO 10 J=1,N
10 A(I-1,J)=A(I,J+1)=A(I,1)*A(P,J)
DO 40 J=1,N
40 A(N,J)=A(P,J)
RETURN
END
BINVER

Method: Gauss-Jordan elimination (see Appendix A)

Usage:

DIMENSION A(n,n + l),B(n + l,n),C(n,n)

... ...

CALL BINVER(C,A,B,N,N1)

where

C = matrix to be inverted

B = inverse of C

A = working storage

n = maximum permissible order of C

N = order of matrix to be inverted

N = value of n in the DIMENSION statement in the calling program

N1 = N + 1

The original value of matrix C is retained.
SUBROUTINE BINVER(C,A,B,N,N1)
DIMENSION A(N,N1),B(N1,N),C(N,N)

MATRIX A HAS DIMENSIONS OF N X (N+1) .
MATRIX B (THE INVERTED MATRIX) HAS DIMENSIONS OF (N+1) X N .
THE ORIGINAL VALUE OF MATRIX C IS RETAINED.
N1 IS N + 1

DO 15 I=1,N
DO 15 J=1,N
13 A(I,J) = C(I,J)
K = N + 1
A(I,K) = 1
DO 10 MC=2,N
10 A(MC,K) = 0
DO 13 L=1,N
DO 11 JC=1,N
11 B(K,JC) = A(I,JC+1)/A(I,1)
DO 12 I=1,N
DO 12 J=1,N
12 B(I,J) = A(I,J+1) = A(I,1)*B(K,J)
DO 13 I=1,N
DO 13 J=1,N
A(I,J) = B(I+1,J)
13 B(I,J) = B(I+1,J)
DO 20 I=1,N
DO 20 J=1,N
20 C(I,J) = B(I,J)
RETURN
END
MATIN1

Method:  Gauss-Jordan elimination (see Appendix A)

Usage:

DIMENSION A(n,m)

.

.

.

CALL MATIN1(A,N)

where

A = matrix to be inverted

n = maximum permissible order of A

m = 2n

N = order of matrix to be inverted

The original value of A is destroyed, and the inverse is returned in A.

The array size in the DIMENSION statement in MATIN1 must conform to the following:

DIMENSION A(n,m)
SUBROUTINE MATIN1(A,N)
DIMENSION A(100,200)
K1 = N + 1
KN = 2*N
DO 1 IZ = 1,N
IX = IZ + N
DO 1 JZ = K1,KN
A(IZ,JZ) = 0*
1 IF(IX.EQ.JZ) A(IZ,JZ) = 1*
DO 2 IZ = 1,N
KZ = IZ
4 IF(ABS(A(IZ,IZ)).EQ.0.0) GO TO 6
IF(A(IZ,IZ)).LT.0001) GO TO 6
GO TO 2
6 DO 3 JZ = 1,KN
3 A(IZ,JZ) = A(IZ,JZ) + A(KZ+1,JZ)
KZ = KZ + 1
GO TO 4
2 CONTINUE
DO 5 IZ = 1,N
DIV = A(IZ,IZ)
DO 9 JZ = 1,KN
9 A(IZ,JZ) = A(IZ,JZ)/DIV
DO 7 K = 1,N
IF(K.EQ.IZ) GO TO 7
XMULT = A(K,IZ)
DO 8 L = 1,KN
A(K,L) = A(K,L) = XMULT*A(IZ,L)
8 CONTINUE
7 CONTINUE
5 CONTINUE
DO 10 IZ = 1,N
DO 10 JZ = 1,N
K2 = JZ + N
10 A(IZ,JZ) = A(IZ,K2)
RETURN
END
MATIN4

Method: Gauss-Jordan elimination (see Appendix A)

Usage:

DIMENSION ARRAY(n,n)

.. .. ..

CALL MATIN4 (ARRAY,N)

where

ARRAY = matrix to be inverted

n = maximum permissible order of ARRAY

N = order of matrix to be inverted

N = value of n in the DIMENSION statement in the calling program

The original value of A is destroyed, and the inverse is returned in A.
SUBROUTINE MAT(list ( ARRAY* P4 i
DIMENSION ARRAY(N,N)
DO 604 I=1,N
STORE = ARRAY(I,I)
ARRAY(I,I)=1.0
DO 601 J=1,N  
601 ARRAY(I,J) = ARRAY(I,J)/STORE 
DO 604 K=1,N  
IF(K-I)602,604,602  
602 STORE = ARRAY(K,I)
ARRAY(K,I) = 0.0 
DO 603 J=1,N  
603 ARRAY(K,J)=ARRAY(K,J)-STORE*ARRAY(I,J)
604 CONTINUE 
RETURN 
END
**MIV**

**Source:** Numerical and Matrix Methods in Structural Mechanics, by P. C. Wang. Copyright (c) 1966 by John Wiley & Sons, Inc. Reprinted by permission.

**Method:** Gauss-Jordan elimination. To prevent division by zero or a very small number, subsequent equations are added to the current one until the divisor becomes larger than some preassigned value.

**Usage:**

```plaintext
DIMENSION A(n,n), U(n,n)

COMMON A,U

CALL MIV(NM)
```

where

- **A** = matrix to be inverted
- **U** = inverse of **A**
- **n** = maximum permissible order of **A**
- **NM** = order of matrix being inverted
The array sizes in the DIMENSION statement in MIV must conform to the following:

```
DIMENSION A(n,n), U(n,n)
```

The original value of A is destroyed, and the inverse is stored in both A and U.

**Comment:** The user may wish to remove U from the DIMENSION statement in the calling program, and from the COMMON statements in both MIV and the calling program.
SUBROUTINE MIV(NM)
DIMENSION A(100,100),U(100,100)
COMMON A,U
DO 9001 I=1,NM
DO 9001 J=1,NM
U(I,J)=0.
IF(I.EQ.J) U(I,J)=1.0
9001 CONTINUE
EPS=0.0000001
DO 9015 I=1,NM
K=1
IF(I.NE.NM)9021,9007,9021
9021 IF(A(I,I).EQ.EPS)9005,9006,9007
9006 K=K+1
DO 9023 J=1,NM
U(I,J)=U(I,J)+U(K,J)
9023 A(I,J)=A(I,J)+A(K,J)
GO TO 9021
9007 DIV=A(I,I)
DO 9009 J=1,NM
U(I,J)=U(I,J)/DIV
9009 A(I,J)=A(I,J)/DIV
DO 9015 MM=1,NM
DELT=A(MM,I)
IF(ABS(DELT).EQ.EPS)9015,9015,9016
9016 IF(MM-I).EQ.0,9010,9010
9010 DO 9011 J=1,NM
U(MM,J)=U(MM,J)-U(I,J)*DELT
9011 A(MM,J)=A(MM,J)-A(I,J)*DELT
9015 CONTINUE
DO 9033 I=1,NM
DO 9033 J=1,NM
9033 A(I,J)=U(I,J)
RETURN
END
INVT


Method: Gauss-Jordan elimination with full pivoting (see Appendix A)

Usage:

DIMENSION A(n,n)

.

.

.

CALL INVT(A,N,M,DETER)

where

A = matrix to be inverted

n = maximum permissible order of A

N = order of matrix being inverted

M = 0 for inversion

M = number of constant vectors if a direct solution of the system is desired.

DETER = determinant of A. If DETER is returned as zero, A is singular.
The original value of A is destroyed, and the inverse is returned in A.

The array sizes in the DIMENSION statement in INVT must conform to the following:

\[
\text{DIMENSION } A(n,n), \text{IPIV}(n), \text{INDEX}(n,2)
\]
SUBROUTINE INVT(A,N,M,DETER)

MATRIX INVERSION AND SIMULTANEOUS EQUATIONS SOLVER

A=INPUT MATRIX FOR INVERSION OR AUGMENTED MATRIX FOR SIM. Eqs.

N=ORDER OF COEFFICIENT MATRIX

M=0 FOR INVERSION ONLY

P=NUMBER OF CONSTANT VECTORS

DETER=DETERMINANT OF COEFFICIENT MATRIX

DIMENSION A(100,101),IPIV(100),INDEX(100,2)

5 DETER=1.0
6 SIGN=N
7 TOL=0.0
8 DO 10 I=1,N
9 DO 10, J=1,N
10 TOL=TOL+ABS(A(I,J))
11 TOL=(TOL+1.0E-7)/(SIGN*SIGN)
14 SIGN=1.0
15 DO 20 J=1,N
20 IPIV(J)=0
25 NN=N+M
30 DO 182 K=1,M
35 AMAX=C.O
40 DO 76 I=1,N
45 IF (IPIV(I)=1)50,76,50
50 DO 75 J=1,N
51 IF (IPIV(J)=1)55,75,75
55 IF (AMAX=ABS(A(I,J)))60,75,75
60 IR=I
65 IC=J
70 AMAX=ABS(A(I,J))
75 CONTINUE
76 CONTINUE
80 IPIV(IC)=IPIV(IC)+1
85 IF (IR=IC)90,115,90
90 SIGN=-SIGN
95 DO 110 L=1,NN
100 AMAX=A(IR*L)
105 A(IR*L)=A(IC*L)
110 A(IC*L)=AMAX
115 INDEX(K,1)=IR
120 INDEX(K,2)=IC
125 AMAX=A(IC*IC)
130 DETER=DETER*AMAX
135 IF (DETER)140,255,140
140 A(IC*IC)=1.0
145 DO 150 L=1,NN
150 A(IC*L)=A(IC*L)/AMAX
155 DO 181 L=1,N
160 IF (L=IC)165,181,165
165 AMAX=A(L,IC)
170 A(L,IC)=0.0

105
175 DO 180 I=1,NN
176 A(L,I)=A(L,I)=A(IC,I)*AMAX
177 IF (I=NN)178,178,180
178 IF (ABS(A(L,I))=TOL)179,179,180
179 A(L,I)=0.0
180 CONTINUE
181 CONTINUE
182 CONTINUE
185 DO 235 I=1,N
190 L=N+1-I
195 IR=INDEX(L,1)
200 IC=INDEX(L,2)
205 IF (IR-IC)<235,210
210 DO 230 K=1,N
220 AMAX=A(K,IR)
225 A(K,IR)=A(K,IC)
230 A(K,IC)=AMAX
235 CONTINUE
240 DETER=SIGN*DETER
245 RETURN
250 M=1
255 RETURN
260 END
**INVERT**

**Source:** George C. Marshall Space Flight Center Computation Laboratory Programmer Procedures Manual

**Method:** Gauss elimination with partial pivoting (see Appendix A).

**Usage:**

```
DIMENSION A(n,n), TEMP(m)
.
.
.
.
CALL INVERT(A,N,ERROR,TEMP)
```

where

- **A** = matrix to be inverted
- **TEMP** = working storage
- **n** = maximum permissible order of **A**
- **m** = 2n
- **N** = order of matrix to be inverted
- **N** = value of **n** in the DIMENSION statement in the calling program.
ERROR = an error indicator. If A is singular, ERROR is returned with a non-zero value.

The original value of A is destroyed, and the inverse is returned in A.

In the event A is singular, a message to that effect is printed. The output device designation is specified in the statement

INTEGER OUT/108/
SUBROUTINE INVERT (A, N, ERROR, TEMP)
INTEGER OUT/108/
DIMENSION A(N,N), TEMP(1)
INTEGER IROW, ERROR
TOL=5.2E-8
ERROR=0
L=N+1
N2=N+N
M=N+1
DO 16 I=1,N
K=N+1
16 TEMP(K)=I
DO 9 IJK=1,N
M=M+1
AMAX=ABS(A(I,1))
IROW=1
IF(M.EQ.1) GO TO 3
DO 2 I=1,M
IF(AMAX.EQ.0) GO TO 2
AMAX=ABS(A(I,1))
IROW=I
2 CONTINUE
3 IF(AMAX) 14,14,4
4 T=1.0/A(IROW,1)
DO 5 I=2,N
5 TEMP(I-1)=A(IROW,I)*T
DO 6 J=1,N
A(IROW,J)=A(I,J)
6 DO 7 I=2,N
7 ELE=A(I,1)*TEMP(J-1)
A(I,1,J-1)=A(I,J)-ELE
CK=(ABS(A(I,J)))/ABS(ELE))
IF(CK=1.0) 18,17,19
18 CK=1.0=CK
GO TO 21
19 CK=CK=1.0
21 IF(CK=TL) 17,17,15
17 A(I-1,J-1)=0.0
15 CONTINUE
7 A(I-1,N)=A(I,1)*T
DO 8 J=2,N
8 A(N,J-1)=TEMP(J-1)
A(N,N)=T
IROW=IROW+N
K=TEMP(IROW)
TEMP(IROW)=TEMP(L)
DO 12 I=2,N
9 J=N+1
12 TEMP(J-1)=TEMP(J)
9 TEMP(N2)=K
   M=N+1
   DO 13 I=1,M
      IROW =N+I
   11 K=TEMP(IROW)
      IF(K.EQ.I) GO TO 13
      KORD=K+N
      TEMP(IROW)=TEMP(KORD)
      TEMP(KORD)=K
   10 DO 10 J=1,N
      TEMP(J)=A(J,I)
      A(J,I)=A(J,K)
   10 A(J,K)=TEMP(J)
      GO TO 11
   13 CONTINUE
1000 RETURN
14 WRITE(OUT,100)
   ERROR=4
100 FORMAT (/120X,'MATRIS IS SINGULAR, NO INVERSE OBTAINABLE//')
   GO TO 1000
END
INVITIT


Method: Decomposition with partial pivoting, iterative improvement, and initial scaling of each row (see Appendix A).

Usage:

```
DIMENSION A(n,n),B(n,n),W(n,n+6)
.
.
.
CALL INVITIT (A,N,M,B,MAXIT,W)
```

where

A = matrix to be inverted

B = inverse of A

W = working storage

n = maximum permissible order of A

N = order of matrix being inverted

M = value of n in the DIMENSION statement in the calling program
MAXIT = maximum number of iterations. If MAXIT = 0, no iterative improvement will be performed. If A is singular, MAXIT is returned as -1.

This subroutine calls subroutine SIMEQA.

The original value of A is destroyed.
SUBROUTINE INVTIT(A, N, M, B, MAXIT, W)

A = INPUT MATRIX, SQUARE
N = NUMBER OF ROWS IN MATRICES A AND B (B IS THE INVERSE MATRIX)
M = NUMBER OF ROWS OF THE ARRAYS WHERE MATRICES A AND B ARE STORED
B = ARRAY WHERE THE INVERSE OF MATRIX A IS TO BE STORED
MAXIT = MAXIMUM NUMBER OF ITERATIONS TO BE ATTEMPTED.
PROGRAM SETS MAXIT=1 IF MATRIX IS SINGULAR
W = WORKING STORAGE, N*(N+6) WORDS

DIMENSION W(N*M), B(M*M)
CALL SIMEGA(A, N, M, 1, E=8, MAXIT, IT, IN, W(1,2), W(1, N+2), W(1, N+3), W(1, N+5), W(1, N+6))

IN=1
DO 110 J=1, N
DO 100 I=1, N
W(I, J) = 0.
100 CONTINUE
W(J, J) = 1.
CALL SIMEQB(W, B(1, J))
IF (IT = LT 0) GO TO 120
IN=2
110 CONTINUE
GO TO 130
120 MAXIT=1
130 RETURN
END
SUBROUTINE SIMEGA(AA,NN,MM,EPS,MAXIT,IT,IN,AB,FACT,ITOP,Y,Z)
DOUBLE PRECISION SUM
DIMENSION A(0,NN),(1)Z(1),Y(1),AA(MM),FACT(1),ITOP(1)
GO TO 400
ENTRY SIMGB(BB,X)
DIMENSION BB(1),X(1)
GO TO (100,250),IN
100 N=NN
NM=N-1
NP=N+1
DO 120 I=1,N
DO 110 J=1,N
A(I,J)=AA(I,J)
110 CONTINUE
120 CONTINUE
C NORMALIZE EACH ROW
DO 150 I=1,N
IF(A(I,1)*LE.O*) K=0
IF(A(I,1)*LE.O*) GO TO 1000
K=ALOG(A(I,1))
1000 DO 130 J=2,N
IF(A(I,J)*LE.O*) GO TO 130
K=MAXO(K,ALOG(A(I,J)))
130 CONTINUE
FACT(I)=EXP(-K)
DO 140 J=1,N
A(I,J)=A(I,J)*FACT(I)
AA(I,J)=A(I,J)
140 CONTINUE
150 CONTINUE
C GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING*
DO 220 M=1,NM
TOP=ABS(A(M,M))
IMAX=M
MP=M+1
DO 160 I=MP,N
IF (TOP*GE*ABS(A(I,M))) GO TO 160
TOP=ABS(A(I,M))
IMAX=I
160 CONTINUE
IF (TOP*GT*O*) GO TO 170
IT=1
GO TO 400
170 ITOP(M)=IMAX
IF (IMAX*LE*MP) GO TO 190
DO 180 I=1,N
TEMP=A(M,I)
A(M,I)=A(IMAX,I)
A(IMAX,I)=TEMP
TEMP=AA(M,I)
180 CONTINUE
AA(M+I)=AA(IMAX+I)
AA(IMAX+I)=TEMP
16C CONTINUE
19C MP=M+1
DO 210 I=MP,N
EM=A(I,M)/A(M,M)
A(I,M)=EM
IF (EM *EQ* 0.) GO TO 210
DC 200 J=MP,N
A(I,J)=A(I,J)-A(M,J)*EM
200 CONTINUE
210 CONTINUE
C
ITOP(N)=N
IF (A(N,N) *NE* 0.) GO TO 250
IT=1
GO TO 400
C AT THIS POINT A=(L+I)*U WITH L AND U LOWER AND UPPER
C TRIANGULAR MATRICES.
C PROCESS RIGHT HAND SIDE
250 ITMAX=MAXIT
IT=0
DO 260 I=1,N
B(I)=BB(I)*FACT(I)
X(I)=0
260 CONTINUE
DO 270 I=1,N
IMAX=ITOP(I)
IF (IMAX *EQ* I) GO TO 265
TEMP=B(I)
B(I)=B(IMAX)
B(IMAX)=TEMP
265 Z(I)=B(I)
BB(I)=B(I)
270 CONTINUE
C COMPUTE Z, WHERE L*Z=B
280 DO 300 I=2,N
K=I-1
SUM=0
DC 290 J=1,K
SUM=SUM+A(I,J)*Z(J)
290 CONTINUE
Z(I)=Z(I)+SUM
300 CONTINUE
C SOLVE FOR X, WHERE U*X=Z
DO 320 K=1,N
I=NP-K
SUM=0.
IF (K *EQ* 1) GO TO 320
IP=I+1
320 CONTINUE
DO 310 J = IP, N
SUM = SUM + A(I, J) * Y(J)
310 CONTINUE
Y(I) = (Z(I) - SUM) / A(I, I)
330 CONTINUE
C ITERATIVE IMPROVEMENT
DO 340 I = 1, N
X(I) = X(I) + Y(I)
340 CONTINUE
IF (ITMAX .NE. 0) GO TO 345
IT = 0
GO TO 400
345 IT = IT + 1
ITMAX = ITMAX + 1
IF (IT .GT. 1) GO TO 360
TOP = ABS(X(1))
DO 350 I = 2, N
IF (ABS(X(I)) .GT. TOP) TOP = ABS(X(I))
350 CONTINUE
TEMP = TOP * EPS
360 TOP = ABS(Y(I))
DO 370 I = 2, N
IF (ABS(Y(I)) .GT. TOP) TOP = ABS(Y(I))
370 CONTINUE
C TEST FOR CONVERGENCE
IF (TOP .LE. TEMP) GO TO 400
DO 390 I = 1, N
SUM = 0
DO 380 J = 1, N
SUM = SUM + AA(I, J) * X(J)
380 CONTINUE
Z(I) = BB(I) - SUM
390 CONTINUE
GO TO 280
400 RETURN
END
MATIN3

Source: Written by W. T. Segui, Department of Civil Engineering, Memphis State University

Method: Partitioning (See Reference 6).

Usage:

```
DIMENSION A(n,n)
   .
   .
   .

CALL MATIN3(A,NO)
```

where

- A = matrix to be inverted
- n = maximum permissible order of A
- NO = order of matrix to be inverted

The original value of A is destroyed, and the inverse is returned in A.

The array sizes in the DIMENSION statement in MATIN3 must conform to the following:

```
DIMENSION A(n,n),C(n),D(n),B3(n,n)
```
SUBROUTINE MATINV(A,NO)
DIMENSION A(100,100),C(100),D(100),B3(100,100)
A(1,1) = 1/A(1,1)
N1 = NO = 1
DO 6 M=1,N1
N = M + 1
DO 10 I=1,M
C(I) = 0.
10 D(I) = 0.
E = 0.
DO 1 I=1,M
DO 3 J=1,M
C(I) = A(I,J)*A(J,N) + C(I)
3 D(I) = A(N,J)*A(J,I) + D(I)
1 E = A(N,I)*C(I) + E
F = A(N,N) = E
A(N,N) = 1/F
DO 4 I=1,M
A(I,N) = -1*C(I)*A(N,N)
4 A(N,I) = -1*A(N,N)*D(I)
DO 5 I=1,M
DO 5 J=1,M
B3(I,J) = C(I)*A(N,J)
5 DO 6 I=1,M
DO 6 J=1,M
6 A(I,J) = A(I,J) = B3(I,J)
RETURN
END
INV1


Method: Bordering (see Reference 7).

Usage:

DIMENSION A(n,n),Z(n,n)
.
.
.
CALL INV1(A,Z,N,KR,NERROR)

where

A = matrix to be inverted

Z = inverse of A

n = maximum permissible order of A

N = order of matrix being inverted

KR = value of n in the DIMENSION statement in the calling program
NERROR = an error indicator. NERROR is returned with a value of zero if A is non-singular and with a non-zero value if A is singular. In the latter case, the numerical value of NERROR is an aid in locating the point in the subroutine from which control is returned.

The original value of A is retained. Matrices A and Z may share the same core locations, but in this case the original value of A is lost.

The array sizes in the DIMENSION statement in INV1 must conform to the following:

```
DIMENSION A(1),Z(1),IX(n),B(n),G(n),DETR(n)
```

If n is anything other than 150, the statement

```
IF(N.GT.150)RETURN
```

must be modified accordingly.

This subroutine produces output. Input/output device designations are specified in the statement

```
DATA NIT,NOT/105,108/
```

(Note that the input device designation is superfluous, since no input is required).

Comments: Several minor changes were made to the original version of this subroutine:

1. The calls to subroutines PAGEHD and ZZBOMB were deleted.
2. Calls to ZZBOMB were replaced by RETURN statements.

3. The variable NERROR, originally an argument of ZZBOMB, was added to the argument list of INV1.

These changes were made for the purpose of obtaining a "stand-alone" subroutine not requiring any supporting subroutines.
SUBROUTINE INV1 (A*,Z*,N*,KR*,NERRCR)
DIMENSION A(*), Z(*), IX(150), IX(150), G(150), DETR(150)
DATA NIT*NOT/105*,108/
C C MATRIX INVERSION (A*1 = Z). BORDERING METHOD.
C THE DETERMINANT RATIO DET(I+1) / DET(I) IS PRINTED. DET(I) IS THE
C DETERMINANT OF THE FIRST I BY I SUB-MATRIX OF A.
C THE INVERSION CHECK Z*A IS CALCULATED AND PRINTED.
C MATRICES A*Z MAY SHARE SAME CORE LOCATIONS. (Z*A CHECK IS INVALID).
C THE MAXIMUM SIZE IS
C N = 150
C CODED BY BOB DILLON. FEBRUARY 1965.
C MODIFIED FOR CONTRACT NAS8-25922, MAY 1971.
C C SUBROUTINE ARGUMENTS
C A = INPUT MATRIX TO BE INVERTED. SIZE(N,N).
C Z = OUTPUT RESULT MATRIX. SIZE(N,N).
C N = INPUT SIZE OF MATRICES A*Z. MAX=150.
C KR = INPUT ROW DIMENSION OF A*Z IN CALLING PROGRAM.
C 2000 FORMAT (// 10X,10(7X,1H1,I2,1H1))
2001 FORMAT (// 10X,45HSUBROUTINE INV1 HAS CALCULATED THE DATA BELOW
* // 10X,44HTHE DETERMINANT RATIOS DET(I+1) / DET(I) ARE
* // (13X,10E11•3))
2002 FORMAT (///10X,37HTHE (A*1)*A) INVERSION CHECK GIVES
* //10X,25HTHE DIAGONAL ELEMENTS ARE // (13X,10F11•8))
2003 FORMAT (///10X,35HTHE MAXIMUM OFF-DIAGONAL ELEMENT IS
* E11•3, 2X, 4HAT ( I 3, 1H, I 3, 1H ) )
C
IF (N GT 150) RETURN
C DO 160 I=2,N
160 IX(I) = I
C INVERT FIRST NON-ZERO ELEMENT IN FIRST COLUMN.
DO 190 I=1,N
190 IF (A(I) NE 0) GO TO 220.
RETURN
C C START INVERSION WITH ROW I.
220 DETR(I) = A(I)
Z(I) = 1. / A(I)
IF (N EQ 1) RETURN
C
IX(I) = 1
IX(1) = I
C BORDERING LOOP.
DO 630 L=2,N
    K = L
L1 = L = 1

250 S = 0.

MIXL = KR * (IX(L) = 1)
LL = IX(L) + MIXL

DO 450 I=1,L1

MIXI = KR * (IX(I) = 1)
LI = IX(L) + MIXI

B(I) = C.
G(I) = C.

DO 440 J=1,L1

MIXJ = KR * (IX(J) = 1)
IJ = IX(I) + MIXJ

JL = IX(J) + MIXL

B(I) = B(I) + Z(IJ) * A(JL)

JL = IX(L) + MIXJ

B(I) = B(I) = G(I) = A(LJ) * Z(JI)

450 S = S + A(LI) * B(I)

AL = A(LL) + S

IF (AL < GE. 0.) GO TO 480

ALBAR = ABS (AL / A(LL))

GO TO 490

480 ALBAR = ABS (AL)

490 IF (ALBAR < GE. 1E-6) GO TO 550

C

C INTERCHANGE ROWS AND COLUMNS.

K = K + 1

IF (K .GT. N) GO TO 540

IX = IX(L)
IX(L) = IX(K)
IX(K) = IX(L)

GO TO 250

540 IF (ALBAR < GE. 1E-5) GO TO 550

C

C COMPUTE INVERSION CHECK Z*A.

XOFF = 0.0

NERROR=3

RETURN

550 Z(LL) = 1. / AL

DETR(L) = AL

DO 570 I=1,L1

IL = IX(I) + MIXL

LI = IX(L) + KR * (IX(I) = 1)

Z(IL) = G(I) * Z(LL)

Z(IL) = G(I) * Z(LL)

DO 570 J=1,L1

IJ. = IX(I) + KR * (IX(J) = 1)

570 Z(IJ) = Z(IJ) + G(J) * Z(IL)

530 CONTINUE
DO 720  I=1,N
DO 710  J=1,N
X = 0.0
KJA = KR * (J=1)
DO 703  K=1,N
IK = 1 + KR*(K=1)
KJ = K + KJA
703  X = X + Z(IK) * A(KJ)
IF (I .NE. J) GO TO 705
G(I) = X
GO TO 710
705  IF (ABS(X) .LT. ABS(XOFF)) GO TO 710
XOFF = X
IOFF = I
JOFF = J
710  CONTINUE
720  CONTINUE

C PRINT THE DETERMINANT RATIO AND INVERSION CHECK.
WRITE (NIO,2000) (JC, JC=1,10)
WRITE (NIO,2001) (DETR(I), I=1,N)
WRITE (NIO,2002) (G (I), I=1,N)
WRITE (NIO,2003) XOFF,IOFF,JOFF
NERROR=0
RETURN

C END
INV1A


Method: Bordering with iterative improvement (see Reference 7).

Usage:

```
DIMENSION A(n,n),Z(n,n),WORK(n,n)

.

.

.

CALL INV1A(A,Z,N,KR,WORK,NERROR)
```

where

A = matrix to be inverted

Z = inverse of A

n = maximum permissible order of A

WORK = working storage

N = order of matrix being inverted
KR = value of n in the DIMENSION statement in the calling program

NERROR = an error indicator. NERROR is returned with a value of zero if A is non-singular and with a non-zero value if A is singular. In the latter case, the numerical value of NERROR is an aid in locating the point in the subroutine from which control is returned.

The original value of A is retained.

The array sizes in the DIMENSION statement in INVLA must conform to the following:

\[ \text{DIMENSION A(l),Z(l),WORK(l),G(n),IX(n),} \]
\[ \text{COL(n),B(n),DETR(n)} \]

If n is anything other than 120, the statement

\[ \text{IF(N.GT.120)RETURN} \]

must be modified accordingly.

This subroutine produces output. Input/output device designations are specified in the statement

\[ \text{DATA NIT,NOT/105,108/} \]

(Note that the input device designation is superfluous, since no input is required).

Comments: Several minor changes were made to the original version of this subroutine:
1. The calls to subroutines PAGEHD and ZZBOMB were deleted.

2. Calls to ZZBOMB were replaced by RETURN statements.

3. The variable NERROR, originally an argument of ZZBOMB, was added to the argument list of INV1A.

These changes were made for the purpose of obtaining a "stand-alone" subroutine not requiring any supporting subroutines.
SUBROUTINE INVIA (A, Z, N, KR, WORK, NERROR)
DIMENSION A(1), Z(1), WORK(1), G(120), IX(120), COL(120), B(120),
* DETR(120)
DATA NIT, NGT/1Q8, 108/

MATRIX INVERSION (A**+1 = Z) - ITERATIVE IMPROVEMENT USING BORDERING
METHOD. (THIS PROGRAM USES 3 N x N SPACES, WHERE SUBROUTINE INV1 USES
ONLY 2).

THE DETERMINANT RATIO DET(I+1) / DET(I), (FROM THE FIRST INVERSION),
IS PRINTED. DET(I) IS THE DETERMINANT OF THE FIRST I x I SUBMATRIX
OF A.

THE FINAL INVERSION CHECK Z*A IS CALCULATED AND PRINTED.

THE MAXIMUM SIZE IS
N = 120

CODED BY BOB DILLON. FEBRUARY 1965.
MODIFIED FOR CONTRACT NASA-25922, MAY 1971.

SUBROUTINE ARGUMENTS

A = INPUT MATRIX TO BE INVERTED. SIZE(N x N).
Z = OUTPUT RESULT MATRIX. SIZE(N x N).
N = INPUT SIZE OF MATRICES A, Z, WORK. MAX = 120.
KR = INPUT ROW DIMENSION OF A, Z, WORK IN CALLING PROGRAM.
WORK = INPUT WORKSPACE MATRIX. SIZE(N x N).

2000 FORMAT ('/ 10X,10(7X,1H(*12,1H))')
2001 FORMAT ('/ 10X,10H THE DETERMINANT RATIOS DET(I+1) / DET(I) ARE
* / (13X,10E11,3))
2002 FORMAT ('/ 10X,37H THE (A**+1)*(A) INVERSION CHECK GIVES
* / (13X,10F11,8))
2003 FORMAT ('/ 10X,35H THE MAXIMUM OFF-DIAGONAL ELEMENT IS
* / E11,3, 2X, 4HAT ( I3, 1H, I3, 1H )
2007 FORMAT ('/ 10X,39H INVERSION IMPROVEMENT DOES NOT CONVERGE /
10X,20H THE FINAL DELTA IS (E17,8,1H) /
* 10X,25H NUMBER OF ITERATIONS IS (I2,1H))
2008 FORMAT ('/ 10X,20H THE FINAL DELTA IS (E17,8,1H) /
10X,25H NUMBER OF ITERATIONS IS (I2,1H))

C

IF (N * GT * 120) RETURN
NERROR = 0

C

INITIAL INVERSION (STATEMENTS FROM INV1).
DO 5 I=2,N
5 IX(I) = I
DO 10 I=1,N
10 IF (A(I) * NE * 0.0) GO TO 15
RETURN
15 DETR(1) = A(I)

NERROR = 1
Z(I) = 1 / A(I)
IF (N - EQ 1) RETURN
IX(I) = 1
IX(1) = 1
DO 65 L=2, N
K = L
L1 = L = 1
20 S = 0
MIXL = KR * (IX(L) = 1)
LL = IX(L) + MIXL
DO 30 I=1, L1
MIXI = KR * (IX(I) = 1)
L1 = IX(L) + MIXI
B(I) = 0
G(I) = 0
DO 25 J=1, L1
MIXJ = KR * (IX(J) = 1)
IJ = IX(I) + MIXJ
JL = IX(J) + MIXL
B(I) = B(I) = Z(IJ) * A(JL)
JI = IX(J) + MIXI
LJ = IX(L) + MIXJ
25 G(I) = G(I) = A(JL) * Z(JI)
30 S = S + A(LI) * B(I)
AL = A(LL) + S
IF (A(LL) - EQ C) GO TO 35
ALBAR = ABS (AL / A(LL))
GO TO 40
35 ALBAR = ABS (AL)
40 IF (ALBAR GE 1E-6) GO TO 55
K = K + 1
IF (K GT N) GO TO 45
IX L = IX(L)
IX(L) = IX(K)
IX(K) = IX L
GO TO 20
45 IF (ALBAR GE 1E-8) GO TO 55
RETURN
55 Z(LL) = 1 / AL
DET(L) = AL
DO 60 I=1, L1
IL = IX(I) + MIXL
LI = IX(L) + KR * (IX(I) = 1)
Z(IL) = B(I) * Z(LL)
Z(LI) = G(I) + Z(LL)
DO 60 J=1, L1
IJ = IX(I) + KR * (IX(J) = 1)
60 Z(IJ) = Z(IJ) + B(J) * Z(IL)
65 CONTINUE
129
C PRINT THE DETERMINANT RATIOS.
   WRITE (NOT,2000) (JC, JC=1,10)
   WRITE (NOT,2000) (DETR(I), I=1,N)
C
C INVERSION IMPROVEMENT LOOP.
E = 1.E+8
DO 465 LIJ=1,10
465 CONTINUE
C MULTIPLe (Z*A).
   MJ = 0
   DO 185 J=1,N
   DO 130 I=1,N
   IK = I
   COL I = 0.
   DO 125 K=1,N
   KJ = K + MJ
   COL I = COL I + Z(IK) * A(KJ)
   IK = IK + KR
125 CONTINUE
   IJ = I + MJ
   WORK(IJ) = COL I
130 CONTINUE
   MJ = MJ + KR
185 CONTINUE
C INVERT (Z*A).
   DO 205 I=2,N
205 IX(I) = I
   DO 210 I=1,N
210 IF (WORK(I) .NE. 0.) GO To 215
   NERROR=4
RETURN
215 DETR(I) = WORK(I)
   WORK(I) = 1. / WORK(I)
   IX(I) = 1
   IX(1) = 1
   DO 265 L=2,N
   L1 = L - 1
   MIXL = KR * (IX(L) - 1)
   LL = IX(L) + MIXL
   DO 230 I=1,L1
   MIXI = KR * (IX(I) - 1)
   LI = IX(L) + MIXI
   B(I) = 0.
   BL(I) = C.
   DO 225 J=1,L1
   MIXJ = KR * (IX(J) - 1)
   LJ = IX(I) + MIXJ
   JL = IX(J) + MIXL
225 CONTINUE
230 CONTINUE
265 CONTINUE
$B(I) = B(I) - W(IJ) \times W(JL)$
$JI = IX(J) + MIXI$
$LJ = IX(L) + MIXJ$

225 $G(I) = G(I) - W(LJ) \times W(IJ)$

230 $S = S + W(IJ) \times B(I)$

AL = WORK(LL) + S

IF (WORK(LL) * EQ. 0.) GO TO 235
ALBAR = ABS (AL / WORK(LL))
GO TO 240

235 ALBAR = ABS (AL)

240 IF (ALBAR * GE. * 1E-6) GO TO 255

K = K + 1

IF (K * GT. * N) GO TO 245

IX L = IX(L)

IX(L) = IX(K)

IX(K) = IX L

GO TO 220

245 IF (ALBAR * GE. * 1E-8) GO TO 255

RETURN

255 WORK(LL) = 1. / AL

DETR(L) = AL

DO 260 I=1,L1

IL = IX(I) + MIXL

LI = IX(L) + KR * (IX(I) = 1)

WORK(IL) = B(I) * WORK(LL)

WORK(LI) = G(I) * WORK(LL)

DO 260 J=1,L1

IJ = IX(I) + KR * (IX(J) = 1)

260 WORK(IJ) = WORK(IJ) + G(J) * WORK(IJ)

255 CONTINUE

C MULTIPLY ((Z*A)**-1)*Z AND TEST CONVERGENCE.

DEL IJ = 0.

MJ = 0

DO 385 J=1,N

DU 330 I=1,N

IK = I

COL I = 0.

DO 325 K=1,N

KJ = K + MJ

COL I = COL I + WORK(IK) * Z(KJ)

IK = IK + KR

325 CONTINUE

COL(I) = COL I

330 CONTINUE

AMAX = 0.

DO 360 I=1,N

IF (AMAX * GE. * ABS(COL(I))) GO TO 360

IJ = I + MJ

DEL I = ABS ((COL(I) = Z(IJ)) / COL(I))
AMAX = ABS(COL(I))

360 CONTINUE
IF (DEL IJ .GE. DEL I) GO TO 375
DEL IJ = DEL I
375 DO 380 I=1,N
   IJ = I + MJ
   Z(IJ) = COL(I)
380 CONTINUE
MJ = MJ + KR
385 CONTINUE
D = DEL IJ
IF (D .LE. 1.E-7) GO TO 415
IF (D .GE. E) GO TO 410
E = 0
405 CONTINUE
GO TO 415
C
410 WRITE (NCT,2007) E,LIJ
GO TO 420
415 WRITE (NCT,2008) D,LIJ
C
C CALCULATE AND PRINT THE INVERSION CHECK Z*A*
420 XOFF = 0.0
   DO 440 I=1,N
   DO 435 J=1,N
      X = 0.0
      KJA = KR * (J-1)
   DO 425 K=1,N
      IK = I + KR*(K-1)
      KJ = K + KJA
   425 X = X + Z(IK) * A(KJ)
   IF (I .NE. J) GO TO 430
   G(I) = X
   GO TO 435
430 IF (ABS(X) .LT. ABS(XOFF)) GO TO 435
   XOFF = X
   IOFF = I
   JOFF = J
435 CONTINUE
440 CONTINUE
   WRITE (NCT,2002) (G(I), I=1,N)
   WRITE (NCT,2003) XOFF,IOFF,JOFF
RETURN
END

132

Method: Rank annihilation (see Reference 2).

Usage:

```
DIMENSION A(n,n),Z(n,n)
.
.
.
CALL INV2(A,Z,N,KR,NERROR)
```

where

- \( A \) = matrix to be inverted
- \( Z \) = inverse of \( A \)
- \( n \) = maximum permissible order of \( A \)
- \( N \) = order of matrix being inverted
- \( KR \) = value of \( n \) in the DIMENSION statement in the calling program
NERROR = an error indicator. NERROR is returned with a value of zero if A is non-singular and with a non-zero value if A is singular. In the latter case, the numerical value of NERROR is an aid in locating the point in the subroutine from which control is returned.

The original value of A is retained.

The array sizes in the DIMENSION statement in INV2 must conform to the following:

\[
\text{DIMENSION } A(KR,1), Z(KR,1), W(n), U(n), IV(n), \\
\quad \text{IRE}(n), \text{BIN}(n)
\]

If \( n \) is anything other than 150, the statement

\[
\text{IF}(N.\ Gt.\ 150)\ \text{RETURN}
\]

must be modified accordingly.

This subroutine produces output. Input/output device designations are specified in the statement

\[
\text{DATA } NIT, NOT/105,108/
\]

(Note that the input device designation is superfluous, since no input is required).

Comments: Several minor changes were made to the original version of this subroutine:

1. The calls to subroutines PAGEHD and ZZBOMB were deleted.

2. Calls to ZZBOMB were replaced by RETURN statements.
3. The variable NERROR, originally an argument of ZZBOMB, was added to the argument list of INV2.

These changes were made for the purpose of obtaining a "stand-alone" subroutine not requiring any supporting subroutines.
SUBROUTINE INV2(A,Z,N,KR,ERROR)
DIMENSION A(KR,1),Z(KR,1),W(150),U(150),IV(150),
+ IRE(150),BIN(150)
DATA NIT,NOT/105,108/
C
MATRIX INVERSION (A = Z) RANK ANNihilation METHOD
ALGORITHM FORMULATED BY CARL BODLEY
THE INVERSION CHECK Z*A IS CALCULATED AND PRINTED
THE MAXIMUM SIZE IS
N = 150
CODED BY CARL BODLEY JANUARY 1967
MODIFIED FOR CONTRACT NAS8-25922 MAY 1971.
SUBROUTINE ARGUMENTS
A = INPUT MATRIX TO BE INVERTED SIZE(N,N)
Z = OUTPUT RESULT MATRIX SIZE(N,N)
N = INPUT SIZE OF MATRICES A,Z MAX=150
KR = INPUT ROW DIMENSION OF A,Z IN CALLING PROGRAM
C
2000 FORMAT(/10X,10(7X,1H,I2,1H))
2001 FORMAT(/10X,45HSUBROUTINE INV2 HAS CALCULATED THE DATA BELOW )
2002 FORMAT(/10X,37HTHE (A**-1)*(A) INVERSION CHECK GIVES
* ///10X,25HTHE DIAGONAL ELEMENTS ARE // (13X,1OF11.8))
2003 FORMAT(/10X,35HTHE MAXIMUM OFF=DIAGONAL ELEMENT IS
* E11-3, 2X, 4HAT (13,1H,13,1H)
C
IF(N.GT.150) RETURN
IF(N.EQ.1.AND.A(1,1).EQ.0.0) RETURN
IF(N.EQ.1.AND.A(1,1).NE.0.0) GO TO 98
C
GENERATE INITIAL ROW INDICES
IT=1
GO TO 90
91 IT=2
90 DO 5 I=1,N
IRE(I)=1
5 IV(I)=1
C
CONDITION A FOR MAXIMUM DIAGONAL ELEMENTS
NM1=N-1
DO 6 L=1,NM1
S MAX=0.0
DO 8 J=L,N
LA=IRE(J)
I=L
K=LA
IF(IT.EQ.2)I=LA
IF(IT.EQ.2)K=L
C
136
IF (ABS (A(K,I)) .LE. SMAX) GO TO 8
JMAX = I
SMAX = ABS (A(K,I))
8 CONTINUE
LS = IRE (L)
IRE (L) = IRE (JMAX)
6 IRE (JMAX) = LS
DO 7 L = 1, N
LA = IRE (L)
BIN (L) = A (LA, L)
IF (IT .EQ. 2) BIN (L) = A (L, LA)
7 IF (BIN (L) .EQ. 0.0) BIN (L) = 1.0
C C GENERATE INITIAL Z AND ABAR C
DO 10 L = 1, N
LA = IRE (L)
I = L
K = LA
IF (IT .EQ. 2) I = LA
IF (IT .EQ. 2) K = L
DO 15 J = 1, N
M = J
M1 = LA
IF (IT .EQ. 2) M = LA
IF (IT .EQ. 2) M1 = J
15 Z (M,M1) = 0.0
Z (I,K) = 1.0 / BIN (L)
10 A (K,I) = A (K,I) / BIN (L)
C C INVERSION LOOP USES ROW OF ABAR WITH MAXIMUM S C
DO 35 L = 1, N
SMAX = 0.0
DO 23 J = L, N
LA = IV (J)
S = 1.0
DO 26 K = 1, N
26 S = S + A (LA, K) * Z (K, LA)
IF (ABS (S) .LE. SMAX) GO TO 23
LMAX = J
SMAX = ABS (S)
23 CONTINUE
IF (SMAX .GT. 1.0 * CE .EQ. 35) GO TO 60
IF (IT .EQ. 2) RETURN
GO TO 65
60 LS = IV (L)
IV (L) = IV (LMAX)
IV (LMAX) = LS
LA = IV (L)
DO 25 I = 1, N
DO 25 J=1,N
25 W(I)=W(I)+A(LA,J)*Z(J,I)
S=1.0+H(LA)
DO 30 I=1,N
30 U(I)=Z(I,LA)
DO 35 I=1,N
35 Z(I,J)=Z(I,J)-U(I)*W(J)/S

C
C   RESTORE A
65 DO 40 L=1,N
L*1=IRE(L)
I=L
K=LA
IF (I*EQ*2) I=LA
IF (I*EQ*2) K=L
40 A(K,I)=A(K,I)+BIN(L)
   IF (SMAX*LE*1.0E-35) GO TO 91

C
C   COMPUTE INVERSION CHECK Z*A
X*FF=0.0
DO 50 J=1,N
DO 45 I=1,N
X=0.0
DO 46 K=1,N
46 X=X+Z(I,K)*Z(K,J)
   IF (I*NE*J) GO TO 47
U(I)=X
GO TO 45
47 IF (ABS(X)*LT*X*FF) GO TO 45
X*FF=X
I*FF=I
J*FF=J
45 CONTINUE
50 CONTINUE

C
C   PRINT INVERSION CHECK AND MAXIMUM OFF DIAGONAL ELEMENT
WRITE (NOT,2000) (JC*JC-1*10)
WRITE (NOT,2001)
WRITE (NOT,2002) (U(I),I=1,N)
WRITE (NOT,2003) X*FF,I*FF,J*FF
NERROR=0
RETURN
95 Z(I,J)=I*O/A(I,J)
NERROR=0
RETURN
END
INV3


Method: Choleski decomposition (see Appendix A).

Usage:

DIMENSION A(n,n),Z(n,n)

.  
.  
.  

CALL INV3(A,Z,N,KR,NERRO)

where

A = matrix to be inverted

Z = inverse of A

n = maximum permissible order of A

N = order of matrix being inverted

KR = value of n in the DIMENSION statement in the calling program
NERRO= an error indicator. NERROR is returned with a value of zero if A is non-singular and with a non-zero value if A is singular. In the latter case, the numerical value of NERROR is an aid in locating the point in the subroutine from which control is returned.

The original value of A is retained. Matrices A and Z may share the same core locations, but in this case the original value of A is lost.

This subroutine calls subroutines DECOM1 and INV4.

The array sizes in the DIMENSION statement in INV3 must conform to the following:

```
DIMENSION A(KR,1),Z(KR,1),W1(n),W2(n),DETR(n)
```

If n is anything other than 150, the error detection statement

```
IF(N.GT.150)RETURN
```

in INV3 must be modified accordingly.

Output is produced by INV3 and may be produced by DECOM1 in the event A is singular. In each of these two subroutines, input/output device designations are specified in the statement

```
DATA NIT, NOT/105,108/
```

(Note that the input device designation is superfluous, since no input is required).
Comments: Several minor changes were made to subroutines INV3, DECOM1, and INV4:

1. The calls to subroutines PAGEHD and ZZBOMB were deleted.

2. Calls to ZZBOMB were replaced by RETURN statements.

3. The variable NERROR, originally an argument of ZZBOMB, was added to the argument lists of INV3, DECOM1, and INV4.

These changes were made for the purpose of making INV3 independent of subroutines PAGEHD and ZZBOMB.
SUBROUTINE INV3(A,Z,N,KR,NERROR)
DIMENSION A(KR,N),Z(KR,N),W1(150),W2(150),DETR(150)
DATA NI,NOT/1C5,lOa/
C
C MATRIX INVERSION (A**-1 = Z). METHOD USES TRIANGULAR DECOMPOSITION
AND TRIANGULAR INVERSION. MATRIX A SHOULD BE SYMMETRIC, POSITIVE
DEFINITE. UPPER HALF OF MATRIX A IS USED TO CALCULATE Z. FULL
MATRICES A,Z ARE USED FOR INVERSION CHECK.
C THE DETERMINANT RATIO DET(I+1)/DET(I) IS PRINTED. DET(I) IS THE
DETERMINANT OF THE FIRST I BY I SUB-MATRIX OF A.
C THE INVERSION CHECK Z*A IS CALCULATED AND PRINTED.
C MATRICES A,Z MAY SHARE SAME CORE LOCATIONS. (Z*A CHECK IS INVALID).
C CALLS FORMA SUBROUTINES DCOM1, INV4.
C THE MAXIMUM SIZE IS
N=150
CODED BY CARL BODLEY. MARCH 1969.
MODIFIED FOR CONTRACT NASB-25922. MAY 1971.
C
SUBROUTINE ARGUMENTS
A = INPUT MATRIX TO BE INVERTED. SIZE(N,N).
Z = OUTPUT RESULT MATRIX SIZE(N,N).
N = INPUT SIZE OF MATRICES A,Z. MAX=150.
KR = INPUT ROW DIMENSION OF A,Z IN CALLING PROGRAM.

2000 FORMAT(1X,10X,1G17,1H(,I2,1H))
2001 FORMAT(1X,10X,45H) SUBROUTINE INV3 HAS CALCULATED THE DATA BELOW
* 10X,45H THE DETERMINANT RATIOS DET(I+1) / DET(I) ARE
* (13X,10E11,3))
2002 FORMAT(1X,35H) THE (A**-1)*(A) INVERSION CHECK GIVES
* (13X,10F11,8))
2003 FORMAT(1X,35H) THE MAXIMUM OFF-DIAGONAL ELEMENT IS
* E11*3,2X,4HAT (.I3,1H, I3, 1H)

C
NERROR=1
IF (N*GT*150) RETURN
C
NERROR=0
CALL DCOM1(A,Z,N,KR,NERROR)
DO 5 I=1,N
5 DETR(I)=Z(I,I)*Z(I,I)
CALL INV4(Z,Z,N,KR,NERROR)
DO 40 L=1,N
DO 20 I=L,N
20 WI(I)=Z(L,I)
DO 30 I=L,N
30 W2(I)=0.0
DO 30 K=L,N
30 W2(I)=W2(I) + Z(I,K)*WI(K)
DO 40 K=1,N
40 Z(K,L)=W2(K)

142
CALCULATE INVERSION CHECK Z*A*
  XOFF=0.0
  DO 120 I=1,N
  DO 110 J=1,N
  X=0.0
  DO 105 K=1,N
  105 X=X+Z(I,K)*A(K,J)
  IF(I*NE.*J) GO TO 108
  W1(I)=X
  GO TO 110
  108 IF(ABS(X)*LT*ABS(XOFF)) GO TO 110
  XOFF=X
  IOFF=I
  JOFF=J
  110 CONTINUE
  120 CONTINUE

PRINT THE DETERMINANT RATIOS AND INVERSION CHECK*
  WRITE(NOT,2000) (JC,JC=1,10)
  WRITE(NOT,2001)(DETR(I),I=1,N)
  WRITE(NOT,2002)(W1(I),I=1,N)
  WRITE(NOT,2003) XOFF,IOFF,JOFF
RETURN
END
SUBROUTINE DCOY1(A,Z,N,KR,NERROR)
DIMENSION A(KR,1),Z(KR,1)
DATA EPS/1.1E-35/
DATA NIT,NOT/105,108/

C
C DECOMPOSE MATRIX (A) TO FORM UPPER TRIANGULAR MATRIX (Z) SUCH THAT
C A = Z(TRANS)*Z* CHOLESKI SQUARE ROOT METHOD.
C MATRIX (A) SHOULD BE REAL, SQUARE, SYMMETRIC, POSITIVE DEFINITE.
C UPPER HALF OF MATRIX (A) IS USED.
C MATRICES (A) AND (Z) MAY SHARE SAME CORE LOCATIONS.
C CODED BY RL WOHLEN, OCTOBER 1970.
C MODIFIED FOR CONTRACT NAS8-25922, MAY 1971.

C SUBROUTINE ARGUMENTS
C A = INPUT MATRIX TO BE DECOMPOSED. SIZE (N,N).
C Z = OUTPUT MATRIX. SIZE (N,N).
C N = INPUT SIZE OF MATRICES A,Z.
C KR = INPUT ROW DIMENSION OF MATRICES A,Z IN CALLING PROGRAM.

3001 FORMAT(5H11 = 13)

C
NERROR=1
IF(A(1,1)LT EPS) RETURN
Z(1,1)=SQRT(A(1,1))
IF(N.EQ.1) RETURN
DO 5 J=2,N
5 Z(1,J)=A(1,J)/Z(1,1)
C
NERROR = 2
 DO 30 I=2,N
 IM1=I-1
 IP1=I+1
 Z(I,1)=A(1,I)
 DO 10 K=1,IM1
10 Z(I,I)=Z(I,I)-Z(K,I)**2
 IF(Z(I,I)LT EPS) GO TO 998
 NERROR=0
 Z(I,1)=SQRT(Z(I,1))
 IF(I.EQ.N) GO TO 40
 DO 30 J=IP1,N
 Z(I,J)=A(I,J)
 DO 20 K=1,IM1
20 Z(I,J)=Z(I,J)-Z(K,I)*Z(K,J)
30 Z(I,J)=Z(I,J)/Z(I,1)
C
40 DO 50 I=2,N
 IM1=I-1
 DO 50 J=1,IM1
50 Z(I,J)=C*0
 RETURN

144
998 WRITE (*NOT, 3001) I
END
SUBROUTINE INV (A, Z, N, KR, NERROR)

DIMENSION A(KR,1), Z(KR,1)

DATA EPS/1.E-35/

MATRIX INVERSION (A **-1 = Z). MATRIX A IS ASSUMED TO BE
UPPER TRIANGULAR.
MATRICES A, Z MAY SHARE SAME CORE LOCATIONS.
CODED BY RL WOHLER, JANUARY 1971.
MODIFIED FOR CONTRACT NAS8-25522, MAY 1971.

SUBROUTINE ARGUMENTS
A = INPUT MATRIX TO BE INVERTED. SIZE(N,N).
Z = OUTPUT RESULT MATRIX. SIZE(N,N).
N = INPUT SIZE OF MATRICES A, Z.
KR = INPUT ROW DIMENSION OF A, Z IN CALLING PROGRAM.

NERROR = 1
DO 10 I=1,N
IF (ABS(A(I,I)) .LT. EPS) RETURN
10 Z(I,I) = 1./A(I,I)
NERROR = 0
IF (N .EQ. 1) RETURN

N1 = N - 1
DO 25 I=1,N1
I1 = I + 1
DO 25 J=I1,N
Z(I,J) = Z(I,J) .+ A(I,J)
IF (J .EQ. I1) GO TO 23
J1 = J - 1
DO 25 K=I1,J1
25 CONTINUE

Z(I,J) = Z(I,J) .+ Z(I,K) .A(K,J)
23 Z(I,J) = Z(I,J) .Z(J,J)
DO 30 I=2,N
I1 = I + 1
DO 30 J=I1,N1
30 Z(I,J) = 0.
RETURN
END
**DPMNVT**

**Source:** George C. Marshall Space Flight Center Computation Laboratory Programmer Procedures Manual

**Method:** Double precision Gauss elimination with partial pivoting (see Appendix A)

**Usage:**

```
DIMENSION A(n,n),TEMP(m)

DOUBLE PRECISION A,TEMP

.  

.  

.  

CALL DPMNVT (A,N,ERROR,TEMP)
```

where

- **A** = matrix to be inverted
- **TEMP** = working storage
- **n** = maximum permissible order of **A**
- **m = -2n**
- **N** = order of matrix to be inverted
N = the value of n in the DIMENSION statement in the calling program

ERROR = an error indicator. If A is singular, ERROR is returned with a non-zero value.

The original value of A is destroyed, and the inverse is returned in A.

In the event A is singular, a message to that effect is printed. The output device designation is specified in the statement

    INTEGER OUT/108/
SUBROUTINE DPIAVT (A, N, ERROR, TEMP)  
DOUBLE PRECISION A, TEMP, AMAX, T  
DIMENSION A(1:N), TEMP(1)  
INTEGER I0UT/1CB/  
INTEGER IROw, ERROR  
ERROR=0  
L=N+1  
N2=N+N  
N=N+1  
DO 16 I=1,N  
K=N+1  
16 TEMP(K)=1  
DO 9 IJK=1,N  
M=I  
AMAX=ABS(A(I,1))  
IROw=1  
IF(M*EG=1) GO TO 3  
DO 2 I=1,M  
IF(AMAX=ABS(A(I,1))) 1,1,2  
1 AMAX=ABS(A(I,1))  
IROw=1  
2 CONTINUE  
3 IF(AMAX) = 1+4  
4 T=1:00/A(IROw,1)  
DO 5 I=2,N  
5 TEMP(I-1)=A(IROw,1)*T  
DO 6 J=1,N  
6 A(IROw,J)=A(1,J)  
DO 7 I=2,N  
7 TEMP(I-1)=A(I,J)  
DO 15 J=2,N  
15 A(I-1,J-1)=A(I,J)=A(I,1)*TEMP(J-1)  
7 A(I-1,N)=A(I,1)*T  
DO 8 J=2,N  
8 A(N,J-1)=TEMP(J-1)  
A(N,N)=T  
IROw=IROw+N  
K=TEMP(IROw)  
TEMP(IROw)=TEMP(L)  
DO 12 I=2,N  
9 TEMP(J-1)=TEMP(J)  
12 TEMP(J)=K  
M=N+1  
DO 13 I=1,M  
13 TEMP=TEMP+I  
11 IF(K*EG=1) GO TO 13  
KORD=K+N  
TEMP(IROw)=TEMP(KORD)  
TEMP(KORD)=K  
999 RETURN  
END
DO 10 J=1,N
    TEMP(J)=A(J,J)
    A(J,J)=A(J,K)
  10    A(J,K)=TEMP(J)
    GO TO 11
13    CONTINUE
1000  RETURN
14    WRITE (GUT,100)
      ERROR=4
      GO TO 1000
100    FORMAT (/'20X,'42H MATRIX IS SINGULAR, NO INVERSE OBTAINABLE///)
    END
Source: Reference 8

Method: Double precision Gauss-Jordan elimination with full pivoting (see Appendix A).

Usage:

DOUBLE PRECISION A(n,n)
INTEGER ORDER
LOGICAL SING

CALL DPINV(A,ORDER,SING)

where

A = matrix to be inverted

n = maximum permissible order of A

ORDER = order of matrix to be inverted

SING = a logical variable which is returned as .TRUE. if the matrix is singular and .FALSE. if the matrix is non-singular
The array sizes in the DOUBLE PRECISION statement in DPINV must conform to the following:

\[
\text{DOUBLE PRECISION } A(n,n), B(n), C(n), \text{PIVOT}, Z
\]

The original value of A is destroyed, and the inverse is returned in A.
SUBROUTINE DPINV(A,ORDER,SING)
THE ARGUMENTS ARE AS FOLLOWS

A IS THE MATRIX TO BE INVERTED. A IS DOUBLE
PRECISION. THE INVERSE OF A IS RETURNED IN
A.

ORDER IS AN INTEGER VARIABLE. THE ORDER OF THE
MATRIX TO BE INVERTED.

SING IS A LOGICAL VARIABLE. WHEN THE ROUTINE
RETURNS SING IS *FALSE* IF THE MATRIX WAS
NON-SINGULAR AND *TRUE* IF THE MATRIX WAS
SINGULAR. IF SING IS *TRUE* THE MATRIX A
CONTAINS GARBAGE.

INTEGER ORDER
DOUBLE PRECISION A(100,100),B(100),C(100),PIVOT,Z
N=ORDER
LOGICAL SING
INTEGER IF(100),IQ(100)
DO 1 K=1,N

C DETERMINATION OF PIVOT ELEMENT
PIVOT=0.
DO 100 I=K,N
DO 100 J=K,N
100 CONTINUE
IF(DABS(A(I,J))-DABS(PIVOT)) 100,100,101
101 PIVOT=A(I,J)
IP(K)=I
IQ(K)=J
100 CONTINUE
IF (PIVOT) 102,900,102
102 IPK=IP(K)
IQK=IQ(K)

C EXCHANGE OF THE PIVOTAL ROW WITH THE KTH ROW
IF(IPK<K) 200,299,200
200 DO 201 J=1,N
Z=A(IPK,J)
A(IPK,J)=A(K,J)
201 A(K,J)=Z
299 CONTINUE

C EXCHANGE OF THE PIVOTAL COLUMN WITH THE KTH COLUMN
IF(IQK<K) 300,399,300
300 DO 301 I=1,N
Z=A(I,QK)
A(I,QK)=A(I,K)
301 A(I,K)=Z
399 CONTINUE

C JORDAN ELIMINATION STEP
DO 400 J=1,N
IF (J=K) GO TO 404
402 D(J) = D(J) / PIVOT
C(J) = 1
GO TO 404
403 B(J) = A(K+J) / PIVOT
C(J) = A(J+K)
404 A(K+J) = 0
405 A(J+K) = 0
DO 406 I = 1, N
DO 405 J = 1, N
405 A(I+J) = A(I+J) + C(I) * B(J)
CONTINUE
C
REORDERING THE MATRIX
K = N
DO 500 KSUM = 1, N
IPK = IP(K)
IQK = IG(K)
IF (IPK = K) GO TO 501
501 DO 503 I = 1, N
Z = A(I, IPK)
A(I, IPK) = A(I, K)
503 A(I, K) = Z
502 IF (IQK = K) GO TO 504
504 DO 506 J = 1, N
Z = A(IQK, J)
A(IQK, J) = A(K, J)
506 A(K, J) = Z
500 K = K - 1
SING = *FALSE*
RETURN
900 SING = *TRUE*
RETURN
END
DPINVT


Method: Double-precision Gauss-Jordan elimination with full pivoting. (See Appendix A).

Usage:

DIMENSION A(n,n)

DOUBLE PRECISION A

CALL DPINVT (A,N,M,DETER)

where

A = matrix to be inverted

n = maximum permissible order of A

N = order of matrix being inverted

M = 0 for inversion

M = number of constant vectors if a direct solution of the system is desired.
DETER = determinant of A. If DETER is returned as zero, then A is singular.

The original value of A is destroyed, and the inverse is returned in A.

The array sizes in the DIMENSION statement in DPINVT must conform to the following:

DIMENSION A(n,n),IPIV(n),INDEX(n,2)
SUBROUTINE DPINVTA(N, M, DET, A)
C
MATRIX INVERSION AND SIMULTANEOUS EQUATIONS SOLVER
C
A=INPUT MATRIX FOR INVERSION OR AUGMENTED MATRIX FOR SIMULTANEOUS EQUATIONS
C
N=ORDER OF COEFFICIENT MATRIX
C
M=0 FOR INVERSION ONLY
C
M=NUMBER OF CONSTANT VECTORS
C
DETER=DETERMINANT OF COEFFICIENT MATRIX
C
DIMENSION A(100,100), IPIV(100), INDEX(100,2)
C
DOUBLE PRECISION A, DETER, AMAX

5 DETE R=1.0D0

14 SIGN=1.0

15 DO 20 J=1,N

20 IPIV(J)=0

25 NN=N+M

30 DO 182 K=1,N

35 AMAX=0.0D0

40 DO 76 I=1,N

45 IF (IPIV(I)=1) GO TO 76

50 DO 75 J=1,N

55 IF (AMAX=DABS(A(I,J)) 160, 75, 75

60 IR=I

65 IC=J

70 AMAX=DABS(A(I,J))

75 CONTINUE

76 CONTINUE

60 IPIV(IC)=IPIV(IC)+1

85 IF (IR=IC) GO TO 95, 130, 90

90 SIGN=-SIGN

95 DO 110 L=1,NN

100 AMAX=A(IR,L)

105 A(IR,L)=A(IC,L)

110 A(IC,L)=AMAX

115 INDEX(K,1)=IR

120 INDEX(K,2)=IC

125 AMAX=A(IC,IC)

130 DETER=DETER*AMAX

135 IF (DETER) 140, 255, 140

140 A(IC,IC)=1.0

145 DO 150 L=1,NN

150 A(IC,L)=A(IC,L)/AMAX

155 DO 181 L=1,N

160 IF (L=IC) 165, 181, 165

165 AMAX=A(L,IC)

170 A(L,IC)=0.0D0

175 DO 180 I=1,NN

176 A(I,L)=A(I,L)=A(IC,I)*AMAX

180 CONTINUE

181 CONTINUE

182 CONTINUE
185 DO 235 1=1,N
190  L=N+I+1
195  IR=INDEX(L,1)
200  IC=INDEX(L,2)
205  IF (IR=IC)210,235,210
210  DO 230 K=1,N
220  AMAX=A(K,IR)
225  A(K,IR)=A(K,IC)
230  A(K,IC)=AMAX
235 CONTINUE
240  DETER=SIGN*DETER
245  RETURN
250  M=1
255  RETURN
     END
SYMVRT

Usage:

DIMENSION A(n,n)

.

.

.

CALL SYMVRT(A,N,ISING)

where

A = matrix to be inverted

n = maximum permissible order of A

N = order of matrix to be inverted

ISING = an error indicator. If A is non-singular, ISING is returned with a value of zero. If A is singular, ISING is returned with a value equal to the row number of the diagonal element which is zero.

The original value of A is destroyed and the inverse is returned in A.

The array sizes in the DIMENSION statement in SYMVRT must conform to the following:

DIMENSION A(n,n),B(n),T(n)
SUBROUTINE SYMVRT(A,N,ISING)
SYMMETRIC INVERSION SUBROUTINE
DIMENSION A(100,100),T(100),B(100)
IF(N*4+1) GO TO 100
I=N
ISING=0
DO 16 ID=1,I
T(ID)=1.
IDM1=ID+1
IDP1=ID+1
IF(IDM1)10,14,10
10 DO 11 J=ID,I
DO 11 K=1,IDM1
11 A(ID,J)=A(ID,J)=A(K,ID)*A(K,J)*T(K)
12 ISING=ID
IF(A(ID,ID))13,12,12
13 T(ID)=-1.
A(ID,ID)=ABS(A(ID,ID))
14 A(ID,ID)=SQRT(A(ID,ID)*T(ID))
IF(IDP1=I)16,16,19
16 DO 17 J=IDP1,I
17 A(ID,J)=A(ID,J)/A(ID,ID)
A(ID,ID)=A(ID,ID)*T(ID)
CONTINUE
19 A(IX,IX)=A(IX,IX)*T(IX)
IXM1=IX+1
DO 24 I=1,IXM1
21 I=I+1
DO 20 J=I,IX
20 A(I,J)=A(I,J)/A(I,I)
23 IXM=IX+1
22 A(I,J)=A(I,J)/A(I,I)
A(IX,IX)=T(IX)/A(IX,IX)
DO 23 I=2,IX
K=I+1
L=I+1
IF(K=IX)24,24,26
21 DO 22 J=K,IX
22 A(L,J)=A(L,J)*A(I,J)*A(L,I)
23 A(L,I)=A(L,I)*A(I,I)
DO 27 I=1,IX
S(I)=0.
DO 29 K=1,IX
29 B(I)=B(I)+A(I,K)**2*T(K)
S=I+1
IF(S=IX)26,26,32
26 DO 27 J=S,IX
A(I,J)=0.
DO 27 K=J,IX
27 A(I,J) = A(I,J) + A(I,K) * A(J,K) * T(K)
32 DO 28 I = 1, M
28 A(I,1) = B(I)
    DO 35 J = 1, N
    DO 35 I = 1, N
35 A(I,J) = A(I,J)
    GO TO 101
101 CONTINUE
100 A(1,1) = 1 / A(1,1)
30 RETURN
   END
SOLEQ


Method: Gauss elimination (see Appendix A).

Usage:

    DIMENSION SK(n,h), Rl(n)

    .

    .

    .

    CALL SOLEQ (SK,Rl,NSZF)

where

    SK = upper half-band of coefficient matrix stored as a rectangular array

    Rl = constant vector on entry

    Rl = solution vector upon return

    n = maximum permissible number of equations

    h = maximum permissible half-band width

    NSZF = number of equations to be solved
The array sizes in the DIMENSION statement in SOLEQ must conform to the following:

\[
\text{DIMENSION SK(n,h), R1(n)}
\]

If \( n \) is anything other than 50, the statement

\[
\text{NBAND = 50}
\]

must be modified accordingly.

The original value of \( SK \) is destroyed.

Comments: Several minor changes were made to the original version of this subroutine:

1. The name was changed from SOLVE to SOLEQ.

2. The COMMON statements were replaced by a DIMENSION statement, and subroutine arguments were added.
SUBROUTINE SCLEG(SK,R1,NSZF)

SPECIFICATION STATEMENTS

DIMENSION SK(200,50),R1(200)
NAND=50

REDUCE MATRIX

DO 300 K=1,NSZF
I=N
DO 290 L=2,NBAND
I=I+1
IF(SK(N,L))240,290,240
240 C=SK(N,L)/SK(N,1)
J=0
DO 270 K=L,NBAND
J=J+1
IF(SK(N,K))260,270,260
260 SK(I,J)=SK(I,J)=C*SK(N,K)
270 CONTINUE
280 SK(N,L)=C

AND LOAD VECTOR FOR EACH EQUATION

R1(I)=R1(I)-C*R1(N)
290 CONTINUE
300 R1(N)=R1(N)/SK(N,1)

BACK=SUBSTITUTION

L=NSZF
350 L=N-1
IF(L)500,500,360
360 L=N
DO 400 K=2,NBAND
L=L+1
IF(SK(N,K))370,400,370
370 R1(N)=R1(N)-SK(N,K)*R1(L)
400 CONTINUE
GO TO 350
500 RETURN
END
SYMSOL


Method: Gauss elimination (see Appendix A). All computations are done in double precision.

Usage:

DIMENSION A(n,h),B(n),C(h)

DOUBLE PRECISION A,B,C

. .

CALL SYMSOL (A,B,C,NN,MM)

where

A = upper half-band of the coefficient matrix stored as a rectangular array

n = maximum permissible number of equations

h = maximum permissible half-band width

B = constant vector on entry
B = solution vector upon return

C = working space needed in the subroutine

NN = value of n in the DIMENSION statement in the calling program

NN = number of equations to be solved

MM = value of h in the DIMENSION statement of the calling program

MM = half-band width of the system to be solved

The original value of A is destroyed.
SUBROUTINE SYMBSOL(A,B,C,NN,MM)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(1N,J1 MM)B(JN,J1 MM)

A IS THE COEFFICIENT MATRIX
B IS THE RIGHT HAND MEMBER
C IS A WORKING SPACE NEEDED IN THE SUBROUTINE
NN IS THE TOTAL NUMBER OF EQUATIONS
MM IS THE HALF-BANDWIDTH OF THE SYSTEM

N=0
100 N=N+1

REDUCE N TH EQUATION

1. DIVIDE RIGHT SIDE BY DIAGONAL ELEMENT
   
   B(N)=B(N)/A(N,1)

2. CHECK FOR LAST EQUATION
   
   IF(N=NN) 150,300,150

   3. DIVIDE N TH EQUATION BY DIAGONAL ELEMENT
      
      150 DO 200 K=2,MM
      200 A(N,K)=A(N,K)/A(N,1)

4. REDUCE REMAINING EQUATIONS
   
   DO 260 L=2,MM
   1=N+L-1
   IF(1=NN) 260,240,240
   240 J=0
   DO 250 K=L,MM
   250 A(I,J)=A(I,J)-C(L)*A(N,K)
   B(I)=B(I)-C(L)*B(N)
   260 CONTINUE
   GO TO 100

BACK SUBSTITUTION

300 N=N-1

1. CHECK FOR FIRST EQUATION

   IF(N) 350,500,350
2. CALCULATE UNKNOWNS B(N)

350 DO 400 K=2, NN
   L=N+K-1
   IF(NN-L) 400, 370, 370
   370 B(N) = B(N) - A(N,K)*B(L)
400 CONTINUE
   GO TO 300
500 RETURN
END
TRIMSS

Source: Army Computation Center, Redstone Arsenal, Huntsville, Alabama

Method: Gauss elimination with partial pivoting (see Appendix A). All computations are performed in double precision.

Usage:

DIMENSION A(n,b+1)
DOUBLE PRECISION A, D

CALL TRIMSS(A, N, NLD, NRD, NED, D, R, E)

where

A = the full band of the coefficient matrix, stored as a rectangular array, augmented by the constant vector (attached as an additional column).

n = maximum permissible number of equations

b = maximum permissible band width

N = number of equations to be solved
NLD = maximum number of band elements left of the diagonal term in any row

NRD = maximum number of band elements right of the diagonal term in any row

NED = NLD + NRD + 1

D = determinant of A

R = rank of A

E = an error indicator

E = 0.0 if a solution is obtained

E = 1.0 if A is singular and no solution attempted

E = 2.0 if a solution was attempted and A was found to be ill-conditioned or singular.

The solution vector is returned in the first column of A. The original value of A is destroyed.

The array size in the DIMENSION statement in TRIMSS must conform to the following:

DIMENSION A(n,b+1)
SUBROUTINE TRIMSSI (N,ND,NLD,NRD,NED,DER,E)
CATEGORY
  MATHEMATICAL
PURPOSE
SOLVE BAND MATRIX SIMULTANEOUS EQUATIONS.
DESCRIPTION
THIS SUBROUTINE WILL SOLVE AN N X N SYSTEM OF SIMULTANEOUS
EQUATIONS WHOSE COEFFICIENT MATRIX IS OF THE BAND FORM,
(I.E. IT HAS ELEMENTS ONLY ABOUT THE MAIN DIAGONAL AND
ZEROES ELSEWHERE). ONLY THE BAND ELEMENTS NEED BE
STORED IN ORDER TO SOLVE THE SYSTEM OF EQUATIONS. THIS
TECHNIQUE OF STORING ONLY THE BAND ELEMENTS ENABLES
ONE TO SOLVE LARGE SYSTEMS OF SIMULTANEOUS EQUATIONS IN
RELATIVELY FEW CORE STORAGE LOCATIONS. THIS TECHNIQUE
MAY ALSO BE USED TO SOLVE SYSTEMS OF SIMULTANEOUS EQUATIONS
WITH NO ZERO ELEMENTS, BUT IN THIS CASE, THERE WOULD BE
NO SAVINGS IN CORE LOCATIONS. THE TECHNIQUE USED TO FIND
THE SIMULTANEOUS SOLUTIONS IS GAUSSIAN ELIMINATION,
MODIFIED TO TAKE ADVANTAGE OF THE REDUCED MATRIX. THE
ROUTINE ALSO USES PARTIAL PIVOTING TO REDUCE ROUNDOFF ERROR.
INPUT
1 A
FIRST LOCATION OF COEFFICIENT MATRIX, I.E. A(1,1).
THE BAND ELEMENTS IN EACH ROW MUST BE LEFT
JUSTIFIED AND EXTEND TO THE RIGHT M PLACES
(M=MIN(N,NLD+NRD+1)). IF IN ANY PARTICULAR ROW
THERE ARE ONLY K BAND ELEMENTS AND K IS LESS
THAN M, THEN THE M-K RIGHT MOST ELEMENTS OF THAT
ROW WILL BE SET TO ZERO. THE ROW WHOSE LEFT
MOST COLUMN IN THE FULL BLOWN MATRIX CONTAINS
A NON-ZERO ELEMENT MUST BE THE FIRST ROW OF THE
REDUCED MATRIX AND ETC. THE COLUMN TO THE
IMMEDIATE RIGHT OF THE REDUCED MATRIX (FORMED AS
ABOVE) MUST CONTAIN THE RIGHT HAND SIDE OF THE
EQUATION SET IN QUESTION. IT SHOULD NOW BE
OBVIOUS THAT AN N X N+1 FULL BLOWN SYSTEM WOULD
BE REDUCED BY THE ABOVE METHOD TO AN N X M+1
SYSTEM.
2 N
NUMBER OF SIMULTANEOUS EQUATIONS TO BE SOLVED.
3 ND VARIABLE DIMENSION INTEGER MUST BE EQUAL TO
ROW DIMENSION OF A IN CALLING PROGRAM.
4 NLD MAXIMUM NUMBER OF NON-ZERO ELEMENTS TO THE LEFT
OF PRINCIPAL DIAGONAL IN ANY ROW OF SYSTEM TO
BE DETERMINED.
5 NRD MAXIMUM NUMBER OF NON-ZERO ELEMENTS TO THE RIGHT
OF PRINCIPAL DIAGONAL IN ANY ROW OF SYSTEM TO
BE DETERMINED.
6 NED NED=MIN(N,NLD+NRD+1)
OUTPUT
1 A
THE FIRST COLUMN OF A CONTAINS THE SOLUTION
VECTOR.
CONTAINS DETERMINANT OF A
CONTAINS RANK OF A
E=0, SOLUTION O.K., E=1, A SINGULAR
E=2, SOLUTION ATTEMPTED, BUT A ILL CONDITIONED
OR SINGULAR. IN THIS CASE SOLUTIONS SHOULD BE
CHECKED TO ASSURE VALIDITY.

REMARKS
ONE OF THE AREAS IN WHICH THIS ROUTINE SHOULD BE VALUABLE
IS THAT OF SOLUTION OF ELLIPTIC BOUNDARY VALUE PROBLEMS.

REFERENCE
ANY GOOD TEXT ON NUMERICAL ANALYSIS
SOME TEXTS ON ELLIPTIC BOUNDARY VALUE PROBLEMS.

CONTACT
CHIEF, THEORETICAL PROBLEMS SECTION
SCIENTIFIC DIGITAL PROGRAMMING BRANCH
ARMY COMPUTATION CENTER

SUBROUTINE TRIMSS(A,L,LD,N,R,DE,EE)
DIMENSION A(3500,4)
DOUBLE PRECISION A,Y,W,S
DOUBLE PRECISION A,Y,W,S
X1 = 1
L1 = 1
E=0
R = 0
D=0.
ND1=NED+1
M=NL0
NM1=N-1
DO 2 I=1,NN
IF (I*GT*(N-NL0)) MM=M+1
NN=I+M-1
DO 2 II=1,NN
IF (ABS(A(I+1)) .GE. ABS(A(II+1))) GO TO 2
D=-0
DO 3 J=1,ND1
Y=A(I,J)
A(I,J)=A(I+1,J)
3 A(I+1,J)=Y
2 CONTINUE
D=D*A(I,1)
IF (A(I,1) .EQ. 0) GO TO 10
GO TO (5,13),L1
13 IF (ABS(ABS((X1-A(I,1))/X1)-1) .LT. 1.E-7) E = 2
X1 = A(I,1)
5 R = R + 1
L1 = 2
DO 4 J=2,ND1
4 A(I,J)=A(I,J)/A(I,1)
K=I+1
NN=1+M
DO 1 II=K,NN
H=A(II,1)
DO 6 J=1,NED
6 A(II,J)=A(II,J+1)+A(I,J+1)*H
A(II,N+1)=A(II,NED)
1 A(II,NED)=0.
D=D*A(N,1)
IF(A(N,1) .EQ. 0.) GO TO 10
IF(AES(ABS((X1*A(N,1))/X1)-1.) .LT. 1.E-7)E=2.
5 R = R + 1*
A(N,1)=A(N,N+1)/A(N,1)
K=N+1
NN=2
6 IF(NN .GT. NED) NN=NN+1
J=K+1
S=0.
DO 7 I=2,NN
S=S+A(J,1)*A(K,I)
7 J=J+1
A(K,1)=A(K,N+1)*S
NN=NN+1
K=K+1
IF(K .NE. 0) GO TO 8
RETURN
10 E=1*
RETURN
END
BANSLV

Source:  Introduction to the Finite Element Method, by
         C. S. Desai and J. F. Abel, Van Nostrand Reinhold
         Company, 1972

Method:  Gauss elimination with decomposition (see Appendix A).

Usage:

        DIMENSION AK(n,h),R(n)
        DOUBLE PRECISION AK,R
              .
              .
              .
        CALL BANSLV(1,AK,R,NEQ,IBAND,NDIM,MDIM)
        CALL BANSLV(2,AK,R,NEQ,IBAND,NDIM,MDIM)

where

        AK = upper half-band of coefficient matrix stored
             in a rectangular array
        n = maximum permissible number of equations
        h = maximum permissible half-band width
        R = constant vector on entry
R = solution vector upon return

NEQ = number of equations to be solved

IBAND = half-band width of the system to be solved

NDIM = value of n in the DIMENSION statement in the calling program

MDIM = value of h in the DIMENSION statement in the calling program

The first call triangularizes the coefficient matrix AK; the second solves for the right-hand side constant vector. If other right-hand sides are to be solved, BANSLV is again called using the value 2 for the first argument (the triangularized coefficient matrix will have been retained).

At the time of this writing, permission to print the program listing had not been received from the publisher.
DECOM and SOLV

**Source:** Written by J. R. Admire, George C. Marshall Space Flight Center.

**Method:** Cholesky decomposition (see Appendix A). All computations are done in double precision.

**Usage:**

```fortran
COMMON/A/ A(n,h), Q(n)
DOUBLE PRECISION A,Q

CALL DECOM (N,NB)
CALL SOLV (N,NB)
```

where

- A = upper half-band of the coefficient matrix stored as a rectangular array
- n = maximum permissible number of equations
- h = maximum permissible half-band width
- Q = constant vector on entry into SOLV
Q = solution vector upon return from SOLV

N = number of equations to be solved

NB = half-band width of the system to be solved

Subroutine DECOM decomposes A into an upper triangular matrix, and SOLV uses this to obtain a solution vector for a given constant vector. SOLV may be called repeatedly if solutions corresponding to several constant vectors are desired.

The original value of A is destroyed, but the upper triangular matrix will be retained after SOLV has been executed.

The array sizes in the COMMON statements in both DECOM and SOLV must conform to the following:

COMMON/A/ A(n,h),Q(n)
SUBROUTINE DECO(N, NB)
DOUBLE PRECISION A, S
COMMON/A/ A(200, 50), Q(200)
DOUBLE PRECISION S, C1, C2, C3
DO 40 I = 1, N
KF = I + 1
JMAX = I + NS + 1
IF (JMAX .GT. N) JMAX = N
DO 40 J = I + 1, JMAX
JJ = J + I
SUM = A(I, JJ)
KS = J + NS + 1
IF (KS .LT. 1) KS = 1
IF (KS .LT. KF) GO TO 15
DO 10 K = KS, KF
IK = I + K
JK = J + K
C1 = A(K, IK)
C2 = A(K, JK)
10 SUM = SUM + C1 * C2
15 IF (I = J) GO TO 20
30 CONTINUE
C3 = SQRT(SUM)
A(I, JJ) = C3
GO TO 40
30 A(I, JJ) = SUM / C3
40 CONTINUE
RETURN
END
SUBROUTINE SOLVIN(N,NB)
DOUBLE PRECISION A(I)
DOUBLE PRECISION SUM,C1,C2,C3
COMMON/A(I(200,5))/Q(200)
DO 21 I=1,N
SUM=Q(I)
KS=I-NB+1
IM=I-1
IF(KS*LT*1) KS=1
IF(KS*GT*IM) GO TO 20
DO 10 K=KS,IM
IK=I-K+1
C1=A(K,IK)
C2=Q(K)
10 SUM=SUM+C1*C2
20 C3=A(I,I)
21 Q(I)=SUM/C3
I=N+1
DO 41 I1=1,N
I=I-1
IP=I+1
JMAX=I+NB-1
SUM=Q(I)
IF(JMAX*GT*N) JMAX=N
IF(IP*GT*JMAX) GO TO 40
DO 30 J=IP,JMAX
J1=J-1+1
C1=A(I,J1)
C2=Q(J)
30 SUM=SUM+C1*C2
40 C3=A(I,I)
41 Q(I)=SUM/C3
RETURN
END
CHOLES

Source: 1969 Vanderbilt University short course on matrix and finite element methods.

Method: Cholesky decomposition (see Appendix A).

Usage:

DIMENSION A(m)

... 

CALL CHOLES (A,N,MM,IB,NT)

where

A = lower half-band of coefficient matrix, stored in a one-dimensioned array, followed by the constant vector

m = maximum permissible number of elements in A

N = number of equations to be solved

MM = half-band width

IB = number of constant vectors to be solved minus one
$NT = 1$ if both decomposition and substitution are desired

$NT \neq 1$ if only substitution is desired (on the second or later entry)

Upon return, the solution vector occupies the space used by the constant vector on entry.

The original value of $A$ is destroyed.

The total number of elements in $A$ is given by

$$m = MM \cdot MM/2 + MM/2 + (N-MM)MM + N.$$
SUBROUTINE CHOLES(A,N,NM,IBINT)

THIS SUBROUTINE SOLVES A SYSTEM OF LINEAR EQUATIONS WHOSE COEFFICIENT MATRIX IS SYMMETRICAL AND BANDED AND IS STORED IN A LINEAR ARRAY. THIS ROUTINE WAS PRESENTED BY VANDERBILT UNIVERSITY DURING THEIR SHORT COURSES ON MATRIX AND FINITE ELEMENT METHOD IN 1969.

THE ARGUMENTS ARE AS FOLLOWS:

A = COEFFICIENT MATRIX + CONSTANT VECTOR
N = NUMBER OF EQUATIONS
NM = HALF-BAND WIDTH + DIAGONAL ELEMENT
I3 = NUMBER OF CONSTANT VECTORS = 1
NT = 1 IF DECOMPOSITION AND SUBSTITUTION DESIRED
GT 1 IF SUBSTITUTION ONLY

DIMENSION A(I1)
MUD=NM+1
NS=MUD*NM/2
NM=N+NM+NS

IF(IN+GT+1) GO TO 31
30 DO 20 J=1,N
20 IF (J=MUD) 1,1/2
2 IN=J*MUD
L=IN+(J-1)*MUD+NS
GO TO 7
1 IN=1
L=IN+(J-1)*J/2
7 IF (J=N+MUD) 133,103,105
105 N3=N
GO TO 104
103 M3=J+MUD
104 S1=0
JI=J-1
J2=J+1
IF(JL=0)
GO 6 K=IN+1
TI=A(L)
S1=S1+TI**2
6 L=L+1
4 TI=A(L)
TI=SRT(TI*S1)
A(L)=TI
IF(J=NT) 19,20,2C
15 DO 18 I=J2,MS
SUM=0
IF((I=MUD) 66,66,71
71 IN=I=MUD
LL=IN+(I-1)*MUD+NS
GO TO 5
66 IN=1
LL=IN+(I-1)*I/2
5 IF(JL=18,18,8

183
6 IF (IN=J1) 53, 53, 18
53 DO 17 K=1,IN
   LM=L*K+J
   SUM=SUM+A(LL)*A(LM)
17 LL=LL+1
18 A(LL)=A(LL)-SUM/A(L)
20 CONTINUE
31 NR=NR+1
   NB=NB+1
   DO 60 K=1,NB
      A(NB)=A(NB)/A(1)
60 DO 60 I=2,N
   IF (I=MUD) 21, 21, 22
22 IN=IN+1
   KS=IN*MUD+NS
   M5=MUD
   GO TO 27
21 IN=0
   MB=I=1
   KS=M5*I/2
27 SUM=0
   DO 61 J=1,M5
      JR=IN+J
      L=JR+KS
      JR=JR+NB+1
61 SUM=SUM+A(L)*A(JR)
   ID=I+KS
60 A(JR+1)=(A(JR+1)+SUM)/A(ID)
55 NB=NB+N
   DO 72 K=1,NB
      A(NB)=A(NB)/A(NM)
72 DO 72 I=2,N
   IF (I=MUD) 41, 41, 95
95 ID=I+(I=MUM)*MUD+NS
   GO TO 42
41 ID=I+(I-1)*I/2
42 IF (I=N*MUM) 43, 43, 45
45 M5=I=1
   GO TO 76
43 M5=MUD
76 SUM=0
   DO 81 J=1,M5
      JR=I+J
      IF (JR=MUD) 98, 98, 99
99 L=I+(JR=MUM)*MUD+NS
   GO TO 82
82 L=I+(JR=1)*JR/2
82 JR=NB*N+JR
B1 SUM=SUM+A(L)*A(JR)
JR=NB+N+1
30 A(JR)=(A(JR)-SUM)/A(ID)
75 NB=NB+N
RETURN
END
SOLBAN

Source: Written by J. E. Key, George C. Marshall Space Flight Center

Method: Symmetric decomposition (see Appendix A). All computations are done in double precision.

Usage:

COMMON/A/A(n,h), Q(n)
DOUBLE PRECISION A,Q
.
.
.
CALL SOLBAN (NU,NB)

where

A = upper half-band of the coefficient matrix stored as a rectangular array

n = maximum permissible number of equations

h = maximum permissible half-band width

Q = constant vector on entry

Q = solution vector upon return
\[ NU = \text{number of equations to be solved} \]
\[ NB = \text{half-band width of the system to be solved} \]

The original value of A is destroyed.

The array sizes in the COMMON statement in SOLBAN must conform to the following:

\[ \text{COMMON/A/ A(n,h), Q(n)} \]
SUBROUTINE SOLBAN(NU,BW)
COMMON/*/ A(260,5C1),Q(260)
DOUBLE PRECISION A,Q,C1,SUM,SUB
INTEGER H,J,H1,BW,10

DO 4 I=1,NU
DO 6 J=1,BW
SUB=C1
SUM=0
H1=I+1
M=BW+1
IF(H1*LE*M) H=H1
IF(J*EQ*1) GO TO 10
IF(H*LE*C) GO TO 17
C COMPUTE OFF DIAGONAL TERMS
DO 11 N=1,H
K=I-M
N=J+M
C=M+1
IF(C*GT*BN) GO TO 11
IF(N*GT*3K) GO TO 11
SUM=SUM+A(K,N)*A(K,N)*A(K,1)
11 CONTINUE
17 A(I,J)=(A(I,J)/SUM)/A(I,1)
GO TO 6
C COMPUTE DIAGONAL TERMS
10 CONTINUE
IF(H*LE*C) GO TO 16
DO 13 M=1,H
K=I-M
N=J+M
C=K+1
SUM=SUM+C1*C1+A(K,1)
C=I=K+1
SUM=SUM+G(K)*A(K,C)
13 CONTINUE
16 A(I,1)=A(I,1)*SUM
C(I)=Q(I)*SUB
6 CONTINUE
4 CONTINUE
DO 12 I=1,NU
I=NU-III+1
SUM=0
H1=III-1
H=NU-1
IF(H1*LE*M) H=H1
IF(H*LT*1) GO TO 15
DO 14 K=I,H
J=I+K
JJ=J-I+1

188
SUM = SUM + A(I,J)*G(J)

14 CONTINUE

15 Q(I) = Q(I)/A(I,1)*SUM

12 CONTINUE

RETURN

END
DISPL

Source: Written by T. J. Chung, University of Alabama at Huntsville Research Institute

Method: Symmetric decomposition (see Appendix A). All computations are done in double precision.

Usage:

\[
\text{DOUBLE PRECISION FORCT}
\]
\[
\text{DIMENSION FORCT(n)}
\]
\[
\text{INTEGER TA, TB}
\]
\[
\ldots
\]
\[
\ldots
\]
\[
\ldots
\]
\[
\text{REWIND TB}
\]
\[
\text{CALL DISPL(N, IHB, FORCT, TA, TB)}
\]

where

\[
\text{FORCT = solution vector}
\]

\[
n = \text{maximum permissible number of equations}
\]

\[
\text{TA = number assigned to a sequential tape or disk file to be used as a scratch file}
\]

\[
\text{TB = number assigned to the sequential tape or disk file in which the lower half-band of the}
\]
coefficient matrix and the constant vector are stored. These coefficients are stored by rows, the first row consisting of the first diagonal element and a right-hand side constant. The second row will contain two coefficients from the lower half-band and a right-hand side constant; the third will contain three lower half-band coefficients and a right-hand side constant. This pattern will continue until the number of lower half-band coefficients is the same as the half-band width. The number of coefficients per row will then be the same for the remaining equations.

\[ N = \text{number of equations to be solved} \]

\[ \text{IHB} = \text{half-band width of the system to be solved} \]

The array sizes in the DIMENSION statement in DISPL must conform to the following:

\[ \text{XK}(h,h), \text{FORCT}(n) \]

where \( h \) is the maximum permissible half-band width.

If \( h \) is anything other than 50, the statements

\[ \text{DO 80 I = 1,50} \]
\[ \text{DO 80 J = 1,50} \]

must be modified accordingly.
SUBROUTINE DISPLIN(IHB, FORCT, TA, TB)

Gaussian-Cholesky solution technique is used

This method works for positive-definite matrices only.

The lower half-band and force vector must be stored by rows on disk TB.

Disk TA is a scratch file.

A maximum of a half band square block of coefficients is required in core storage at any given time. The dimensions of XK are each equal to the maximum half band width. The dimension of FORCT is the maximum number of equations.

The force vector (FORCT) is replaced by the displacement vector (i.e., the force vector is destroyed).

Definition of arguments -

N = number of equations
IHB = half-band width (including diagonal term)
FORCT = force vector (initially)
   = displacement vector when this subroutine has been executed.

INTEGER ORDER, TA, TB
DIMENSION XK(50,50), FORCT(1000)
DOUBLE PRECISION XK, FORCT, SUM

READ TA

Determine the number of blocks of coefficients required in core.

ORDER=N
AORDER=IHB
AORDER=ORDER
BLOCK=AORDER/AIHB
NBLOCK=ORDER/IHB
ABLOCK=NBLOCK
IF (BLOCK. GE. ABLOCK) NBLOCK=NBLOCK+1

Factor and forward substitution of stiffness matrix assembled in a half band square block at a time.

DO 60 I=1,50
DO 60 J=1,50

193
**CHANGE OF XK SUBSCRIPTS FOR THE THIRD AND/OR LATER BLOCKS**

1.

```
DO 2 I = 2, M
DO 2 J = 2, I
2 XK(I,J) = XK(I,N-1+J)
```

9.

```
DO 4 I = 1, MM
IF (NN*NE*1) GO TO 101
READITB(XK(I,J), J=1, I), FORCT(I)
GO TO 102
```

101 CONTINUE

```
READ FROM DISK (XK(I,J), J=1, M) AND FORCT(I)
```

102 CONTINUE

```
DO 6 J = 1, N
SUB = 0.
SUM = 0.
IF (NN*NE*1) GO TO 7
IF (NN*GE*2) GO TO 8
7 IF (I*EQ*J) GO TO 20
IF (J*GT*1) GO TO 6
```

**OFF-DIAGONAL FACTOR FOR THE FIRST BLOCK**

```
J1 = J - 1.
IF (J1*EQ*0) GO TO 21
DO 22 K = 1, J1
22 SUM = SUM + XK(I,K)*XK(J,K)*XK(K,J)
21 XK(I,J) = (XK(I,J)*SUM)/XK(J,J)
GO TO 6
```

**DIAGONAL FACTOR FOR THE FIRST BLOCK**

```
I1 = I - 1.
IF (I1*EQ*0) GO TO 23
DO 24 K = 1, I1
24 SUM = SUM + XK(I,K)*XK(I,K)*XK(K,K)
24 SUB = SUB + XK(I,K)*FORCT(K)
23 XK(I,J) = XK(I,J) + SUB
FORCT(I) = FORCT(I) - SUB
GO TO 6
```

194
OFF-DIAGONAL FACTOR FOR BLOCKS OTHER THAN THE FIRST:

8 IF(J*EG*M) GO TO 10
   II=I+J
   IJ=I+J
   IF(IJ=M*GT*O) IJ=I+J=M
   IF(II=M*GT*O)II=M
   Ji=J=1
   IF(J11*EG*O)GO TO 12
   DO 11 K=1,J11
   IK=I+K
   K1=IK
   MJK=M-J+K
   IF(MJK*GT*I1+AND*I1+J*LE*M) MJK=I*K
   IF(I+K=I*GT*O)IK=M
   IF(I+K=M*GT*O)K1=I+K=M
11 SUM = SUM + XK(I*J)*XK(IJ,MJK)*XK(KI,K1)
12 XK(I,J) = (XK(I,J)-SUM)/XK(IJ,II)
   GO TO 6

DIAGONAL FACTOR FOR BLOCKS OTHER THAN THE FIRST:

10 MI=M-1
   DO 13 K=1,MI
   IK=I+K
   K1=IK
   IF(IK=GT*M) IK=I+K=M
   IF(K1=GT*M) K1=M
   SUM = SUM + XK(I*J)*XK(I,J)*XK(KI,K1)
13 SUB = SUB + XK(I*J)*FORCT(IK)
   XK(I,J) = XK(I,J) - SUM
   FORCT(I) = FORCT(I) - SUB
   CONTINUE
   WRITE(I,TM) (XK(I,J),J=1,M),FORCT(I)
4 CONTINUE
5 CONTINUE

BACK SUBSTITUTION STARTS

REWIND TB
BACKSPACE TA
MM = M*MM
DO 52 NN=1,NSLOCK
   IF(NN=GT*1)MM=MM
   DO 52 I=1,MM
30 MI=MM-I+1
   SUM=0
   READ(TA) (XK(MI,J),J=1,M),FORCT(MI)
   IF(NN=EG*NSLOCK*AND*I*EG*M) GO TO 39
TO READ FROM THE BOTTOM UP:

BACKSPACE TA
BACKSPACE TA
33 CONTINUE
  IF (NN.LT.NBLOCK) GC TO 33

FIRST BLOCK (TRIANGULAR) REARRANGED:
  II=II+1
  IF (II.EQ.0) GO TO 33
  DO 34 J=1,MI
   JMI=J+I-II
  34 XK(MI,K=J+1) = XK(MI,JMI)
  33 IF (NN.GT.1) GO TO 36

DISPLACEMENTS FOR THE LAST BLOCK:
  II=II-1
  IF (II.EQ.0) GO TO 50
  DO 51 K=1,II
   51 SUM = SUM + XK(MI+K,M-K)*FCRT(MI+K)
  50 FCRT(MI) = FCRT(MI)/XK(MI,MI) = SUM
  GO TO 52

DISPLACEMENTS FOR BLOCKS OTHER THAN THE LAST:
  56 II = NN + I - 1
   IF (II.GT.M-1 OR NN.GT.2) II=M-1
  31 DO 55 K=1,II
   MK=MI+K
   IF (MK.GT.M) MK=M-K
  55 SUM = SUM + XK(MK,M-K)*FCRT(MK)
  52 FCRT(MI) = (FCRT(MI)/XK(MI,MI))*SUM

NOW THE DISPLACEMENTS ARE STORED ON DISK TO BE IN REVERSE ORDER WHICH WILL BE READ AND STORED IN CORE WITH THE ORDER PROPERLY RESTORED:

REWIND TB
  DO 56 I=1,N
   J=N-I+1
  56 READ(TB) FCRT(J)
RETURN
END
BANSOL


Method: Gauss elimination (see Appendix A).

Usage:

COMMON/BANARG/MM,NUMBLK,B(n),A(m,p),NN,IT,JT
DOUBLE PRECISION A,B

CALL BANSOL

where

MM = half-band width of the system being solved

NUMBLK = number of blocks of coefficients which constitute the upper half-band

B = solution vector

A = a block of coefficients of the upper half-band

NN = depth of each block (i.e., the number of rows in each block)
\[ n = \text{the product of } \text{NUMBLK and } \text{NN (or larger)} \]

\[ m = \text{twice the block depth } \text{NN (or larger)} \]

\[ p = \text{the larger of } \text{MM and } \text{NUMBLK} \]

\[ IT = \text{number assigned to a sequential tape or disk} \]
\[ \text{file to be used as a scratch file} \]

\[ JT = \text{number assigned to the sequential tape or} \]
\[ \text{disk file in which the upper half-band and} \]
\[ \text{constant vector are written.} \]

The upper half-band and constant vector must be written on file JT in blocks. The depth of each block must be at least as large as the half-band width, but no larger than half the first dimension of A (i.e., A is actually an array containing two of these blocks of coefficients). Each block has the same depth, which will possibly necessitate generation of dummy zero elements to completely fill the last block. Each row of a block consists of a right-hand side constant followed by a row of A coefficients, the number of which equals the half-band width.

The second dimension of A must be equal to or greater than the half-band width and equal to or greater than the number of blocks. The dimension of the B array must be equal to or greater than the product the block depth and the number of blocks.
The COMMON statement in BANSOL must conform to the following:

```
COMMON/BANARG/MM, NUMBLK, B(n), A(m,p), NN, IT, JT
```
SUBPROGRAM SUBROUTINE BANSOL

SUBROUTINE BANSOL
COMMON /BARGY/, M, NUMBLK, BI (200), AI (4, 100), IT, JT
DOUBLE PRECISION A, B, C
N1 = N1 + 1
N1 = N1 + N1
REIND IT
REIND JT
NB = 0
GO TO 150

REDUCE EQUATIONS BY BLOCKS
SHIFT BLOCK OF EQUATIONS

150 NS = NS + 1
DO 125 N = 1, MM
N1 = N1 + N
BN (N) = 0(N)
BN (N1) = 0
DC 125 M = 1, MM
AN (N, M) = AN (N, M1)
125 AN (N1, M) = 0

READ NEXT BLOCK OF EQUATIONS INTO CORE

IF (NUMBLK = NB) 150, 200, 150
190 READ (IJ 1) (BN (N), AN (N, M), M = 1, MM), N = N1, NH)
IF (NB) 200, 100, 200

REDUCE BLOCK OF EQUATIONS

200 DO 250 N = 1, MM
IF (AN (N, 1)) 225, 300, 225
225 BN (N) = BN (N) / AN (N, 1)
DO 275 L = 2, MM
IF (AN (N, L)) 230, 275, 230
230 CN = AN (N, L) / AN (N, 1)
I = N + L - 1
J = 0
DO 250 K = 1, MM
J = J + 1
250 AN (I, J) = AN (I, J) - CN * AN (N, K)
BN (I) = BN (I) - AN (N, L) * BN (N)
AN (N, L) = C
275 CONTINUE
300 CONTINUE

WRITE BLOCK OF REDUCED EQUATIONS ON DISK IT
C
IF (NUMBLK=NB) 375,400,375
375 WRITE (IT) (B(N), (A(N,M), M=2, NN), N=1, NN)
GO TO 100
C
BACK=SUBSTITUTION
C
400 DO 450 M=1,NN
   N=NN+1=M
   DO 425 K=2,MM
       L=N+K-1
   425 B(N)=B(N)-A(N,K)*B(L)
   NM=N+NN
   B(NM)=B(N)
450 A(NM,NB)=B(N)
   NB=NB+1
   IF (NB) 475,500,475
475 BACKSPACE IT
   READ (IT) (B(N), (A(N,M), M=2, MM), N=1, NN)
   BACKSPACE IT
   GO TO 400
C
ORDER UNKNOWNS IN B ARRAY
C
500 K=0
   DO 600 NB=1,NUMBLK
       DO 600 N=1,NN
           NM=N+NN
           K=K+1
   600 B(K)=A(NM,NB)
   RETURN
   END
III. DISCUSSION

3.1 Non-banded Methods

This discussion of the results obtained from the solution of the test system given in Section I applies only to those results, and should not be extrapolated to necessarily include other types of systems.

All of the in-core, non-banded subprograms were employed to solve a 100th order system of the type given in Eq. (1.1). The results are summarized in Tables 1 and 2. The execution times given in all tables are approximate, and those times less than one second are listed as one second. The two error measures tabulated are those discussed in Section 1 of this report. The comment, "no solution" means that the subroutine was still executing after a total job time of five minutes and was terminated.

Superior accuracy was achieved when either double precision arithmetic or iterative improvement was used. In most instances the accuracy obtained with the direct solvers was independent of the solution method used. Exceptions were subroutines INVT, which uses Gauss-Jordan elimination with full pivoting, and INV2, which uses the method of rank annihilation. A large time penalty is paid for this above-average accuracy, however. In general, the use of double
Table 1.
In-core, Non-banded Equation Solvers
(Single Precision Except as Noted)

<table>
<thead>
<tr>
<th>Subprogram Name</th>
<th>Execution Time, sec.</th>
<th>Error Measure</th>
<th>Maximum Error</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMEQ</td>
<td>1</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SIMEQ3</td>
<td>14</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>WUGEL</td>
<td>16</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>GAUSEL</td>
<td>11</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SIMEQ2</td>
<td>17</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>FACTOR, LU</td>
<td>14</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>GAUSS</td>
<td>12</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SOLVIT</td>
<td>3</td>
<td>0.00970</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SMEQL</td>
<td>3</td>
<td>0.00000170</td>
<td>0.00000229</td>
<td>1 iteration</td>
</tr>
<tr>
<td>SMEQL1</td>
<td>5</td>
<td>0.0</td>
<td>0.0</td>
<td>2 iterations</td>
</tr>
<tr>
<td>DPSIM</td>
<td>20</td>
<td>0.00968</td>
<td>0.137</td>
<td>Double precision</td>
</tr>
<tr>
<td>SHORT</td>
<td>19</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SOLVE1</td>
<td>31</td>
<td>0.00969</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>WUIGJ</td>
<td>48</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>WUGJN</td>
<td>25</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SLVE</td>
<td>34</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>GASSIM</td>
<td>31</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>GJR</td>
<td>42</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>SIMULT</td>
<td>1</td>
<td>0.159x10^{-13}</td>
<td>0.174x10^{-12}</td>
<td>Double precision</td>
</tr>
<tr>
<td>SOLV 3</td>
<td>1</td>
<td>0.00968</td>
<td>0.137</td>
<td>2199 Gauss-Seidel iterations</td>
</tr>
<tr>
<td>SOLV 2</td>
<td>25</td>
<td>0.00588</td>
<td>0.00583</td>
<td>200 Gauss-Seidel iterations</td>
</tr>
<tr>
<td>GSEID</td>
<td>103</td>
<td>5.51</td>
<td>84.1</td>
<td></td>
</tr>
<tr>
<td>Subprogram Name</td>
<td>Execution Time, sec.</td>
<td>Error Measure</td>
<td>Maximum Error</td>
<td>Comments</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>AINVER</td>
<td>57</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>BINVER</td>
<td>103</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>MATIN1</td>
<td>69</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>MATIN4</td>
<td>45</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>MIV</td>
<td>25</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>INVVT</td>
<td>197</td>
<td>0.0000549</td>
<td>0.000900</td>
<td></td>
</tr>
<tr>
<td>INVERT</td>
<td>85</td>
<td>---</td>
<td>---</td>
<td>50 equations</td>
</tr>
<tr>
<td></td>
<td>177</td>
<td>---</td>
<td>---</td>
<td>70 equations</td>
</tr>
<tr>
<td></td>
<td>300+</td>
<td>---</td>
<td>---</td>
<td>100 equations, no solution</td>
</tr>
<tr>
<td>INVVTIT</td>
<td>63</td>
<td>0.00970</td>
<td>0.137</td>
<td>No iterative improvement</td>
</tr>
<tr>
<td></td>
<td>117</td>
<td>0.0000170</td>
<td>0.000229</td>
<td>1 iteration</td>
</tr>
<tr>
<td></td>
<td>262</td>
<td>0.0</td>
<td>0.0</td>
<td>2 iterations</td>
</tr>
<tr>
<td>MATIN3</td>
<td>38</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>INV1</td>
<td>110</td>
<td>0.00968</td>
<td>0.137</td>
<td></td>
</tr>
<tr>
<td>INV1A</td>
<td>48</td>
<td>---</td>
<td>---</td>
<td>50 equations</td>
</tr>
<tr>
<td></td>
<td>154</td>
<td>---</td>
<td>---</td>
<td>70 equations</td>
</tr>
<tr>
<td></td>
<td>300+</td>
<td>---</td>
<td>---</td>
<td>100 equations, no solution</td>
</tr>
<tr>
<td>INV2</td>
<td>121</td>
<td>0.000334</td>
<td>0.00455</td>
<td></td>
</tr>
<tr>
<td>INV3</td>
<td>71</td>
<td>0.0176</td>
<td>0.254</td>
<td></td>
</tr>
<tr>
<td>DPMNVT</td>
<td>58</td>
<td>0.255x10^-11</td>
<td>0.356x10^-10</td>
<td>Double precision</td>
</tr>
<tr>
<td>DPinv</td>
<td>47</td>
<td>0.687x10^-13</td>
<td>0.101x10^-11</td>
<td>Double precision</td>
</tr>
<tr>
<td>DPinvt</td>
<td>79</td>
<td>0.270x10^-13</td>
<td>0.423x10^-12</td>
<td>Double precision</td>
</tr>
<tr>
<td>SYMVRT</td>
<td>24</td>
<td>0.0177</td>
<td>0.254</td>
<td></td>
</tr>
</tbody>
</table>
precision arithmetic appears to be the easiest way to gain accuracy without excessive execution times, but computer core storage requirements are roughly doubled. Two inversion subroutines which use Choleski decomposition, INV3 and SYMVRT, yielded results with somewhat less than average accuracy.

As expected, the Gauss elimination schemes are somewhat faster than the Gauss-Jordan elimination solvers. Subroutine DPINV, a double precision matrix inversion subroutine, is surprisingly fast in view of the fact that it uses Gauss-Jordan elimination with full pivoting.

The execution times for subroutines SIMEQ and SOLVIT are misleading and need some elaboration. In each of these subroutines, a provision is made for the possibility of a zero multiplier when rows of the coefficient matrix are modified. When such a situation exists, the modification of that row is omitted, since there is no change in those coefficients. In a large matrix with a narrow band, such as the test matrix, this amounts to a large reduction in the computations. This is discussed in more detail in Appendix A.

Subroutine SIMULT and SOLVE 3, designed specifically for sparse matrices proved to be very efficient for the test matrix, which contained relatively few non-zero terms.

Some of the inversion subroutines, as well as the equation solver SIMEQ3, retain the original value of the coefficient matrix. This feature is of dubious value, however, since it requires twice as much core storage. If required, the same results can be achieved by storing the coefficient matrix in an additional array before entry into the subroutine.
Subroutine SOLVE 2 and GSEID were the only Gauss–Seidel iteration procedure tested. In the test of GSEID, an initial trial solution vector containing all zeros was used. After 200 iterations, no usable solutions had been found, but the procedure did appear to be slowly converging. The performance of SOLVE 2, designed for sparse matrices, was much better, but still not comparable to direct equation solvers. In a large system with a narrow band of nonzero coefficients, the effects of changing the value of one unknown takes a long time to reach the other unknowns. It is felt that this, at least in part, accounts for the poor showing of these two subroutines.

3.2 Banded Solvers

The results of tests of those equation solvers which recognize the banded nature of the coefficient matrix are tabulated in Table 3. Unlike the non-banded subroutines, these were difficult to compare with regard to execution times. Some require in-core storage of the full band of coefficients, whereas some require only the half-band.

Where possible, the bandwidth of the test matrix was artificially increased by including zeros within the band. This was done in order to get execution times large enough for comparison, but some of the subroutines would not accept this type of coefficient matrix.

As can be observed from the error indicators tabulated, single precision accuracy for 200 or more equations is unacceptable. In all cases tested, satisfactory results were obtained when double precision arithmetic was used.
<table>
<thead>
<tr>
<th>Subprogram Name</th>
<th>Precision</th>
<th>Number of Equations</th>
<th>Half-band width</th>
<th>Execution Time, sec</th>
<th>Maximum Error</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLEQ</td>
<td>single</td>
<td>200</td>
<td>50</td>
<td>1</td>
<td>0.0549</td>
<td>full band required</td>
</tr>
<tr>
<td>SYMSS</td>
<td>double</td>
<td>200</td>
<td>50</td>
<td>13</td>
<td>0.148×10⁻¹⁰</td>
<td></td>
</tr>
<tr>
<td>TRMSS</td>
<td>double</td>
<td>200</td>
<td>3500</td>
<td>2</td>
<td>0.184×10⁻⁷</td>
<td></td>
</tr>
<tr>
<td>BANSVV</td>
<td>single</td>
<td>200</td>
<td>50</td>
<td>2</td>
<td>0.0549</td>
<td></td>
</tr>
<tr>
<td>DECOM, SOLV</td>
<td>double</td>
<td>3500</td>
<td>2</td>
<td>9</td>
<td>0.184×10⁻⁷</td>
<td></td>
</tr>
<tr>
<td>CHOLES</td>
<td>single</td>
<td>3500</td>
<td>2</td>
<td>1</td>
<td>0.127×10⁻¹⁰</td>
<td></td>
</tr>
<tr>
<td>SOLBAN</td>
<td>double</td>
<td>200</td>
<td>2</td>
<td>14</td>
<td>0.0713</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.36×10⁻⁹</td>
</tr>
</tbody>
</table>
3.3 Out-of-core Solvers

Only two subroutines utilizing auxiliary data storage devices were tested, and the results are given below in Table 4. Programs which effectively utilize temporary tape or disk storage files are relatively scarce. Most efficient large scale structural analysis programs use techniques which mesh the solution procedure with the generation of the coefficients, and it is difficult to extract a stand-alone equation solver from them.

Both subroutines DISPL and BANSOL bring the coefficients of the half-band into core in blocks, or segments. DISPL uses the lower half-band and BANSOL the upper. The term "block size" in Table 4 refers to the number of rows of coefficients in the block. In these particular cases, it also coincides with the half-band width. The minimum block size in each case is \( h \times h \), where \( h \) is the half-band width.

<table>
<thead>
<tr>
<th>Subprogram Name</th>
<th>Execution Time, sec</th>
<th>Error Measure</th>
<th>Maximum Error</th>
<th>Block Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPL</td>
<td>95</td>
<td>( 0.177 \times 10^{-10} )</td>
<td>( 0.360 \times 10^{-9} )</td>
<td>2</td>
</tr>
<tr>
<td>BANSOL</td>
<td>29</td>
<td>( 0.177 \times 10^{-10} )</td>
<td>( 0.360 \times 10^{-9} )</td>
<td>2</td>
</tr>
</tbody>
</table>
The accuracy achieved was identical in both cases, but BANSOL was considerably faster than DISPL. The primary reason for this is that DISPL requires only one block of coefficients in core at one time, whereas BANSOL requires two. As a consequence, there is considerably less shifting of data in and out of core with BANSOL. With either subroutine, the block size is limited only by the core capacity of the computer. Using the maximum possible block size, of course, will result in maximum efficiency.
IV. RECOMMENDATIONS

Although the primary purpose of this report is to document and catalogue subroutines, some recommendations can be made on the basis of their performance with the test system given in Section 1.

On the basis of the test results obtained in this study, Gauss-Seidel iteration has little to recommend it. The extremely slow convergence exhibited renders it impractical for large systems of equations. A reevaluation may be necessary, however, if systems different from the test system are to be solved.

In general, inversion methods are not recommended unless the inverse matrix is needed for its own sake. Multiple constant vectors can be treated more efficiently by solving them simultaneously. If a decomposition method is used, they can be efficiently handled separately. For this reason, and based upon their performance in the tests, subprograms SOLVIT (decomposition) with iterative improvement, FACTOR and LU (decomposition) and WUGEL (Gauss elimination with multiple constant vectors permitted) are recommended in-core, non-banded procedures. In addition, function subprogram FACTOR computes the determinant of the coefficient matrix. If an inverse matrix is desired, subroutines DPINV and MIV are recommended.
For symmetric banded systems, those subroutines which require only the half-band of coefficients should be used. Subroutine CHOLES requires less core storage than any of the others of this type, but the arrangement of the coefficients may not be the most convenient one. Unless the core storage requirements make CHOLES attractive, the more convenient subroutines BANSLV and the combination of DECOM and SOLV are suggested. Both solvers use decomposition and provide for solution of more than one constant vector.

For very large systems, either DISPL or BANSOL may be used to advantage. DISPL requires less core, but BANSOL is faster, so the selection should be made on that basis.

In all cases, whether banded, non-banded, in-core, or out-of-core, double precision arithmetic should be employed if the core storage capacity permits it.
A.1 General

Although there exist many methods for the solution of systems of linear algebraic equations, there are only a few basic methods with many variations. Among the direct methods, the method of Gauss is probably the most frequently encountered. As will be pointed out later, this technique is a derivative of a more fundamental method, that of decomposition. Gauss-Seidel iteration is the most common iterative method in use, and consequently, it will be the only one considered in this appendix.

Only the more commonly used methods will be discussed. Some of the lesser known or infrequently used procedures are not included, but can be found in the references indicated in the subroutine descriptions in Part 2 of this report. There is no attempt to provide rigorous mathematical formulations in what follows. The intent is to furnish a review of the basic techniques of equation solving, or in the case of unfamiliar methods, a concise explanation. More exhaustive treatments can be found in a number of excellent textbooks (2),(5),(7),(9),(10).
A system of $n$ linear algebraic equations

\[
\begin{align*}
    a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
    a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
    \vdots \\
    a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{align*}
\]  
(A.1)

where the $x$'s are unknowns, and the $a$'s and $b$'s are constants, may be written in matrix form as

\[
\begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{bmatrix}
= 
\begin{bmatrix}
    b_1 \\
    b_2 \\
    \vdots \\
    b_n
\end{bmatrix}
\]  
(A.2)

or

\[
A X = B
\]  
(A.3)

where $A$ is called the coefficient matrix, $X$ the solution vector, and $B$ the right-hand side or constant vector. For a given coefficient matrix, the problem consists of determining the solution vector corresponding to a given constant vector.
Equation (A.3) may also be written in the form

$$\mathbf{X} = \mathbf{A}^{-1} \mathbf{B}$$  \hspace{1cm} (A.4)

where $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$. Some techniques depend upon determining the inverse matrix and premultiplying the constant vector by it to obtain a solution vector, whereas other methods seek the solution directly, bypassing the computation of the inverse.

A.2 Gauss Elimination

The objective in Gauss elimination is to transform the equations from their form in Eq. (A.1) to an equivalent system which has the form

$$
\begin{align*}
    a_{11}' x_1 + a_{12}' x_2 + \cdots + a_{1n}' x_n &= b_1' \\
    0 + a_{22}' x_2 + \cdots + a_{2n}' x_n &= b_2' \\
    \vdots \\
    0 + 0 + \cdots + a_{n1}' x_1 &= b_n'
\end{align*}
$$

The matrix representation is

$$
\begin{bmatrix}
    a_{11}' & a_{12}' & \cdots & a_{1n}' \\
    0 & a_{22}' & \cdots & a_{2n}' \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & a_{nn}'
\end{bmatrix}
\begin{bmatrix}
    x_1 \\
    x_2 \\
    \vdots \\
    x_n
\end{bmatrix}
= 
\begin{bmatrix}
    b_1' \\
    b_2' \\
    \vdots \\
    b_n'
\end{bmatrix}  \hspace{1cm} (A.5)
$$
Once this transformation has been achieved, the last equation can be solved directly for $X_n$, since it is the only unknown in that equation. Equation (n-1), consisting of zero terms to the left of the next-to-last unknown, can then be solved for this unknown. The entire solution vector can thus be obtained in a backward sweep through this upper triangular system. This back-substitution algorithm can be expressed by the equation

$$x_i = \frac{1}{a_{ii}} \left( b_i - \sum_{k=i+1}^{n} a_{ik} x_k \right) \quad (A.6)$$

if the summation is taken as zero when $i = n$. This is $b_n/a_{nn}$. The convention used throughout this appendix with regard to summation is as follows: if the upper limit is not equal to or greater than the lower limit, the sum is taken as zero.

It will always be possible to reduce the system to the form of Eq. (A.5) provided $A$ is nonsingular (i.e., its determinant is not zero).

We begin by solving for $x_1$ in the first equation and substituting the solution into the remaining equations. From the first equation,

$$x_1 = \left[ b_1 - \frac{(a_{i2} x_2 + a_{i3} x_3 + \cdots + a_{in} x_n)}{a_{11}} \right] \quad (A.7)$$

The second equation becomes
\[
a_{2l} \left[ b_1 - \left( a_{i2} x_2 + a_{i3} x_3 + \cdots + a_{in} x_n \right) \right]
\]
\[
\frac{a_{i1}}{a_{ii}}
\]
\[
+ a_{22} x_2 + a_{23} x_3 + \cdots + a_{2n} x_n = b_2
\]
or
\[
(a_{22} - \frac{a_{21}}{a_{ii}} a_{i2}) x_2 + (a_{23} - \frac{a_{21}}{a_{ii}} a_{i3}) x_3
\]
\[
+ \cdots + (a_{2n} - \frac{a_{21}}{a_{ii}} a_{in}) x_n = b_2 - \frac{a_{21}}{a_{ii}} b_1
\]
or
\[
a'_{22} x_2 + a'_{23} x_3 + \cdots + a'_{2n} x_n = b_2
\]

In other words, subtracting \( \frac{a_{21}}{a_{ii}} \) times the first equation from the second will eliminate the unknown \( x_1 \) from the second equation. Similarly, when Eq. (A.7) is substituted into any of the remaining equations, the effect is to subtract \( \frac{a_{ii}}{a_{ii}} \) times the first equation from equation \( i \), eliminating \( x_1 \) from that equation. In this manner, the entire system can be reduced to the form
where
\[ a'_{ij} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{ij} \]
and
\[ b'_{i} = b_{i} - \frac{a_{i1}}{a_{11}} b_{1} \]

The next step is to eliminate \( x_{2} \) from each equation after the second to obtain the equivalent system

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\
  0 & a'_{22} & a'_{23} & \ldots & a'_{2n} \\
  0 & 0 & a''_{33} & \ldots & a''_{3n} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & a'_{n3} & \ldots & a'_{nn} \\
\end{bmatrix}
\begin{bmatrix}
  x_{1} \\
  x_{2} \\
  x_{3} \\
  \vdots \\
  x_{n} \\
\end{bmatrix}
= 
\begin{bmatrix}
  b_{1} \\
  b'_{2} \\
  b'_{3} \\
  \vdots \\
  b'_{n} \\
\end{bmatrix}
\]
where

\[ a'_{ij} = a_{ij} - \frac{a'_{ik}}{a'_{kk}} \]

and

\[ b'_i = b_i - \frac{a'_{ik}}{a'_{kk}} b_k \]

Next, the third unknown is eliminated from each equation after the third, the fourth unknown is eliminated from each equation after the fourth, etc., until the system is reduced to the upper triangular form of Eq. (A.5).

In general, at the kth stage of elimination the kth unknown is eliminated from all equations that follow the kth. There will be a total of n-1 stages, since the nth unknown will not be eliminated. Dispensing with the primes and letting the equals sign be a replacement or substitution operator, we can express the forward elimination procedure as

\[ a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} \]  \hspace{1cm} (A.8)

and

\[ b_i = b_i - \frac{a_{ik}}{a_{kk}} b_k \]  \hspace{1cm} (A.9)
These relationships give the new values of the coefficients of each of the equations following the kth equation, as obtained during the kth stage of elimination. In practice, division of the kth row of A and B by \( a_{kk} \) is usually done at the beginning of each stage, so that the final form of the system is

\[
\begin{bmatrix}
1 & a_{12}' & a_{13}' & \cdots & a_{1n}' \\
0 & 1 & a_{23}' & \cdots & a_{2n}' \\
0 & 0 & 1 & \cdots & a_{3n}' \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_n
\end{bmatrix}
=
\begin{bmatrix}
b_1' \\
b_2' \\
b_3' \\
\vdots \\
b_n'
\end{bmatrix}
\]

With this modification, the complete forward elimination algorithm is given by

\[
a_{kj}' = a_{kj} / a_{kk} \quad \text{for } j = k, \ldots, n
\]

\[
a_{ij}' = a_{ij} - a_{ik} a_{kj}' \quad \text{for } j = k+1, \ldots, n
\]

\[
b_i' = b_i - a_{ik} b_k
\]

and the backward sweep by

\[
x_i = b_i - \sum_{k=i+1}^{n} a_{ik} x_k
\]
If the coefficient matrix $A$ is banded -- that is, if the nonzero coefficients are all within a narrow band about the principle diagonal -- then a substantial savings in computational effort can be effected by a simple modification of the above algorithm. If $A$ is banded, then beginning with some equation after the $k$th, the coefficient of $x_k$ will be zero for all of the remaining equations. The coefficients in those equations will be unaltered, and therefore the elimination for that stage can be terminated. Even if the coefficient matrix is not banded, a zero $a_{ik}$ will allow the $i$th equation to be skipped, and a check for this possibility is worth incorporating into a computer program.

If several sets of equations differing only in their constant vectors are to be solved, they can be solved at the same time. The forward elimination will be the same with respect to $A$ regardless of whether it is performed once or many times. In Eq. (A.9), the $a_{ik}/a_{kk}$ multiplier will be the same each time the system is solved for a different $B$, so all of the $B$ vectors can be handled at once. The back-substitution procedure of Eq. (A.6) can then be applied to all of the $B$ vectors simultaneously to produce the corresponding solution vectors.

Examination of Eqs. (A.8) and (A.9) reveals that if, at any stage $k$ of the elimination, one of the diagonal coefficients $a_{kk}$ is zero, an indeterminant situation will exist. The difficulty can be circumvented by searching the
equations following the kth for one with a nonzero coefficient in the kth column of A. This equation is then interchanged with the kth equation so that the $a_{kk}$ term is now nonzero. The $a_{kk}$ term that is used is called the pivot element, or simply the pivot, and the row containing this element is the pivot row. If a nonzero pivot element cannot be found, then the matrix A is singular, and a solution is unattainable by any means.

If the diagonal coefficient is not zero but very small, computational difficulties may arise. The division may produce a number whose magnitude exceeds the capacity of the computer, or roundoff errors may be introduced. One way to minimize these possibilities is to search the column of A below the term being eliminated and choose the row with the largest element in that column as the pivot row. This strategy is known as partial pivoting, or partial positioning. If the search for the largest possible pivot extends to the entire submatrix below the diagonal position of the kth unknown, the process is called full pivoting. In this procedure both columns and rows must be interchanged. This will change the order of the unknowns, necessitating a rearrangement of the solution vector.

A.3 Gauss-Jordan Elimination

In this variation of Gauss elimination, whenever an unknown is eliminated, it is eliminated from all equations
other than the pivot equation, not just those following the pivot equation. After the second stage of elimination, for example, the system will have the form

\[
\begin{bmatrix}
1 & 0 & \cdots & a'_{1n} \\
0 & 1 & \cdots & a'_{2n} \\
0 & 0 & \cdots & a'_{3n} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_n
\end{bmatrix} =
\begin{bmatrix}
b'_1 \\
b'_2 \\
b'_3 \\
\vdots \\
b'_n
\end{bmatrix}
\]

After the forward sweep is finished, the system of equations is reduced to the form

\[
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
0 & 0 & \cdots & 1 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_n
\end{bmatrix} =
\begin{bmatrix}
b'_1 \\
b'_2 \\
b'_3 \\
\vdots \\
b'_n
\end{bmatrix}
\]

and no backward substitution is required, since the transformed B vector is now equal to the solution vector X.

222
The Gauss-Jordan elimination scheme is frequently used to compute the inverse of a matrix. It was previously pointed out that multiple constant vectors can be solved simultaneously to obtain several solution vectors. If $n$ constant vectors are selected such that they form the columns of an $n$th order unit matrix $I$, where

$$I = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}$$

there will also be $n$ solution vectors which will form an $n$th order matrix $C$, where each column of $C$ is a solution vector corresponding to the constant vector contained in the same column of $I$. If the system $AC = I$ is solved for $C$, then $C$, by definition, will be the inverse of $A$. If no row or column interchanges have been made (i.e., the diagonal elements are used as pivots), the matrix of solution vectors $C$ is the inverse with no row or column rearrangements necessary.

A.4 Decomposition

If the coefficient matrix $A$ and all of its upper left principle submatrices are nonsingular, $A$ may be expressed as the product of a lower triangular and an upper triangular matrix:
If \( A \) is decomposed into lower and upper triangular matrices \( L \) and \( U \), the system \( AX = B \) may be written as \( LUX = B \) or

\[
\begin{bmatrix}
1_{11} & 0 & \cdots & 0 \\
1_{21} & 1_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
1_{n1} & 1_{n2} & \cdots & 1_{nn}
\end{bmatrix}
\begin{bmatrix}
u_{11} & v_{12} & \cdots & v_{1n} \\
v_{21} & v_{22} & \cdots & v_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
v_{n1} & v_{n2} & \cdots & v_{nn}
\end{bmatrix}
= \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\]

(A.10)

where

\[
L C = B
\]

(A.11)

and

\[
UX = C
\]

(A.12)

Since \( L \) is triangular, the system (A.11) can be easily solved for \( C \) by beginning with the first unknown and proceeding sequentially to the last. The procedure is exactly the reverse of that for the solution of an upper triangular system as given in Eq. (A.6) and can be expressed as

\[
c_i = \frac{1}{l_{ii}} \left( b_i - \sum_{k=1}^{i-1} l_{ik} c_k \right)
\]

if the summation term is taken as zero when \( i = 1 \).

Once the vector \( C \) has been computed, the solution vector \( X \) can be found from Eq. (A.12). Since \( U \) is an upper triangular matrix, the solution algorithm is

\[
x_i = \frac{1}{u_{ii}} \left( c_i - \sum_{k = i+1}^{n} u_{ik} x_k \right)
\]

224
An examination of the system (A.10) shows that there are a total of \( n^2 + n \) undetermined \( \lambda \) and \( u \) coefficients and only \( n^2 \) equations. In order that the elements of \( L \) and \( U \) may be obtained, \( n \) of them must be specified in advance.

The usual technique is to put 1's on the principal diagonal of either \( L \) or \( U \). If we let the \( u_{ii} = 1 \), the system becomes

\[
\begin{bmatrix}
\ell_{11} & 0 & \cdots & 0 \\
\ell_{21} & \ell_{22} & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
\ell_{n1} & \ell_{n2} & \cdots & \ell_{nn}
\end{bmatrix}
\begin{bmatrix}
u_{12} & \cdots & \nu_{1n} \\
0 & 1 & \cdots & \nu_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
= 
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\]

Expanding the above, we obtain the following equations:

\[
\begin{align*}
\ell_{11} \cdot 1 & = a_{11} \\
\ell_{11} \cdot u_{12} & = a_{12} \\
\vdots & \\
\ell_{11} \cdot u_{1n} & = a_{1n} \\
\ell_{21} \cdot 1 & = a_{21}
\end{align*}
\]
Matrices $L$ and $U$ cannot be computed independently; if the first column of $L$ is computed first, the first row of $U$ can then be obtained. Next the second column of $L$ is found, then the second row of $U$, and so forth. The expressions for elements of these two matrices are

\[
\begin{align*}
\ell_{21} u_{12} + \ell_{22} \cdot 1 &= a_{22} \\
&\vdots \\
\ell_{n1} u_{1n} + \ell_{n2} u_{2n} &= a_{nn} \\
\ell_{n1} u_{12} + \ell_{n2} \cdot 1 &= a_{n2} \\
&\vdots \\
\ell_{n1} u_{1n} + \ell_{n2} u_{2n} + \cdots + \ell_{nn} \cdot 1 &= a_{nn}
\end{align*}
\]

Matrices $L$ and $U$ cannot be computed independently; if the first column of $L$ is computed first, the first row of $U$ can then be obtained. Next the second column of $L$ is found, then the second row of $U$, and so forth. The expressions for elements of these two matrices are

\[
\begin{align*}
u_{kk} &= 1 \\
\ell_{ik} &= a_{ik} - \sum_{m=1}^{k-1} \ell_{im} u_{mk} \quad \text{for } i = k, \ldots, n
\end{align*}
\]
When the diagonal elements of \( U \) are specified as unity as has been done here, the decomposition process is called the Crout reduction. When the diagonal elements of \( L \) are assigned a value of one, the decomposition is known as the Doolittle method. The algorithm for terms of \( L \) and \( U \) in this case is

\[
\lambda_{kj} = \frac{1}{\lambda_{jj}} \left( \alpha_{kj} - \sum_{m=1}^{j-1} \lambda_{km} u_{mj} \right) \quad \text{for } j = 1, \ldots, k-1
\]

\[
\lambda_{ij} = 0 \quad \text{for } i < k
\]

\[
\lambda_{kj} = 0 \quad \text{for } j < k
\]

The pattern here is to compute the first row of \( L \), then the first row of \( U \), the second row of \( L \), the second row of \( U \), etc. Once \( A \) is decomposed into an upper triangular and a lower triangular matrix, regardless of the technique used, the solution vector is obtained by solving Eqs. (A.11) and (A.12).
As in the case of Gauss elimination, several constant vectors may be handled at the same time. If the columns of an nth order unit matrix are treated as constant vectors, the inverse of A may be found by decomposition. If we write the system as \( AX = I \), or as \( X = A^{-1} I \), then \( A^{-1} = (LU)^{-1} = L^{-1}U^{-1} \). If L and U are known, their inverses may be easily found. Using a 3 x 3 upper triangular matrix for illustration and denoting elements of the inverse matrix by \( g \), we can express the relationship \( UU^{-1} = I \) as

\[
\begin{bmatrix}
  u_{11} & u_{12} & u_{13} \\
  0 & u_{22} & u_{23} \\
  0 & 0 & u_{33}
\end{bmatrix}
\begin{bmatrix}
  g_{11} & g_{12} & g_{13} \\
  g_{21} & g_{22} & g_{23} \\
  g_{31} & g_{32} & g_{33}
\end{bmatrix}
= \begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\]

Expanding this, we obtain

\[ u_{11} g_{11} + u_{12} g_{21} + u_{13} g_{31} = 1 \]
\[ u_{11} g_{12} + u_{12} g_{22} + u_{13} g_{32} = 0 \]
\[ u_{11} g_{13} + u_{12} g_{23} + u_{13} g_{33} = 0 \]
\[ u_{22} g_{21} + u_{23} g_{31} = 0 \]
\[ u_{22} g_{22} + u_{23} g_{32} = 1 \]
It will be observed that the inverse of an upper triangular matrix is also an upper triangular matrix. The pattern that emerges is

\[
\begin{align*}
g_{ii} &= \frac{1}{u_{ii}} \quad \text{for } i = n, \ldots, 1 \\
g_{ij} &= \frac{1}{u_{ii}} \left( -\sum_{k=i+1}^{j} u_{ik} g_{kj} \right) 
\end{align*}
\] (A.13)

for \( i = n-1, \ldots, 1 \) and \( j = n, \ldots, i+1 \).

In a similar fashion, the algorithm for obtaining elements of the inverse of a lower triangular matrix (which is itself lower triangular) can be deduced as

\[
\begin{align*}
g_{ii} &= \frac{1}{\ell_{ii}} \quad \text{for } i = 1, \ldots, n \\
g_{ij} &= \frac{1}{\ell_{ii}} \left( -\sum_{k=j}^{i-1} \ell_{ik} g_{kj} \right)
\end{align*}
\] (A.14)

for \( j = 1, \ldots, i-1 \) and \( i = 2, \ldots, n \).
A.5 Decomposition of Symmetric Matrices

If $A$ satisfies the criteria for decomposition set forth in the previous discussion and in addition is symmetric, it may be decomposed into the product of a lower triangular matrix, a diagonal matrix, and an upper triangular matrix, where the upper triangular matrix is the transpose of the lower triangular matrix. In matrix notation, the equivalent system is $LDL^T = A$. Furthermore, both the lower and upper triangular matrices will have 1's on their principal diagonals:

$$
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
\ell_{21} & 1 & \cdots & 0 \\
\vdots & & \ddots & \vdots \\
\ell_{n1} & \ell_{n2} & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
d_{11} & 0 & \cdots & 0 \\
0 & d_{22} & \cdots & 0 \\
\vdots & & \ddots & \vdots \\
0 & 0 & \cdots & d_{nn}
\end{bmatrix}
\begin{bmatrix}
1 & \ell_{21} & \cdots & \ell_{n1} \\
0 & 1 & \cdots & \ell_{n2} \\
\vdots & & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
$$

$$
= 
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
$$

230
Incorporating $D$ into $L$ we have

\[
\begin{bmatrix}
d_{11} & 0 & \cdots & 0 \\
\lambda_{21} d_{11} & d_{22} & \cdots & 0 \\
\vdots & & \ddots & \vdots \\
\lambda_{n1} d_{11} & \lambda_{n2} d_{22} & \cdots & d_{nn}
\end{bmatrix}
\begin{bmatrix}
1 & \lambda_{21} & \cdots & \lambda_{n1} \\
0 & 1 & \cdots & \lambda_{n2} \\
\vdots & & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
= \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\]

Expanding,

\[
\begin{align*}
d_{11} &= \lambda_{11} \\
d_{11} \lambda_{21} &= \lambda_{12} \\
&\vdots \\
d_{11} \lambda_{n1} &= \lambda_{1n} \\
\lambda_{21} d_{11} &= \lambda_{21} \\
\lambda_{21} d_{11} \lambda_{22} + d_{22} &= \lambda_{22} \\
&\vdots \\
\lambda_{n1} d_{11} \lambda_{n2} + d_{22} \lambda_{n2} &= \lambda_{n1} \\
&\vdots \\
\lambda_{n1} d_{11} \lambda_{n1} + d_{22} \lambda_{n1} + \cdots + d_{nn} &= \lambda_{nn}
\end{align*}
\]
by accounting for the symmetry of \( A \), we can compute the elements of \( D \) and \( L \) from

\[
d_{ii} = a_{ii} - \sum_{r=1}^{i-1} \lambda_{ir}^2 d_{rr}
\]

\[
\lambda_{ij} = \frac{1}{d_{jj}} \left( a_{ij} - \sum_{r=1}^{j-1} \lambda_{ir} \lambda_{jr} d_{rr} \right) \quad \text{for } i > j
\]

and

\[
\lambda_{ij} = 0 \quad \text{for } i < j
\]

To obtain the solution vector \( X \), we first write the system \( AX = B \) in the form \( LDL^TX = B \) which becomes

\[
LC = B \quad \text{(A.15)}
\]

where

\[
DLTXT = C \quad \text{(A.16)}
\]

As before, the final solution is obtained in two stages: Eq. (A.15) is solved for \( C \), followed by the solution of Eq. (A.16) for \( X \).

Taking into account the unit \( l_{ii} \), the procedure for solution of the lower triangular system (A.15) is

\[
c_i = b_i - \sum_{k=1}^{i-1} \lambda_{ik} c_k
\]
Since $DL^T$ is upper triangular, the previously given algorithm for the solution of upper triangular systems can be used.

If the elements of $DL^T$ are termed $g_{ij}$, they may be determined from the relations

$$g_{ii} = d_{ii}, \text{ and } g_{ij} = d_{ii} l_{ji}, \text{ for } i < j$$

The solution of Eq. (A.16) for $X$ is then given by

$$x_i = \frac{1}{g_{ii}} \left( c_i - \sum_{k=1}^{n} g_{ik} x_k \right)$$

$$= \frac{1}{d_{ii}} \left( c_i - \sum_{k=1}^{n} d_{ik} \cdot l_{ki} x_k \right)$$

It is perhaps worth mentioning at this point that the most general form of the decomposition technique is embodied in the equation $LDU = A$. In this equation, $A$ is any square matrix whose upper left principal submatrices are nonsingular. The matrix $L$ is lower triangular, $D$ is diagonal, and $U$ is upper triangular. In the Crout variation, the diagonal matrix $D$ does not appear explicitly and is incorporated into $L$. The Doolittle method combines $D$ with $U$.
The Choleski square root method, to be discussed next, incorporates D into both the upper and the lower triangular matrices. Even Gauss elimination is a modification of the decomposition technique. Since AX = B reduces to UX = B', and the product of the matrix operators used to obtain U is a lower triangular matrix, the reduction to upper triangular form can be expressed as LA = U and A = L⁻¹U. This is equivalent to A = LDU with D included in U.

A.6 Choleski Decomposition

If, in addition to having the prerequisites for triangular decomposition, the coefficient matrix A is symmetric and positive definite, it will admit to a decomposition of the form A = LLᵀ or UᵀU. If we use the latter form, the decomposition is represented by

\[
\begin{bmatrix}
    u_{11} & 0 & \cdots & 0 \\
    u_{12} & u_{22} & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots \\
    u_{1n} & u_{2n} & \cdots & u_{nn}
\end{bmatrix}
\begin{bmatrix}
    u_{11} & u_{12} & \cdots & u_{1n} \\
    0 & u_{22} & \cdots & u_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & \cdots & u_{nn}
\end{bmatrix}
= 
\begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1n} \\
    a_{21} & a_{22} & \cdots & a_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\]

Expanding,

\[
u_{11}^2 = a_{11}
\]
\[
u_{11}u_{12} = a_{12}
\]
\[
\vdots
\]

234
If we omit the computations for the redundant symmetric elements, the elements of $U$ can be found as follows:

$$u_{ii} u_{in} = a_{in}$$

$$u_{i2} u_{11} = a_{12}$$

$$u_{22}^2 + u_{22}^2 = a_{22}$$

$$\vdots$$

$$u_{i2} u_{in} + u_{22} u_{2n} = a_{2n}$$

$$\vdots$$

$$u_{in} u_{11} = a_{in}$$

$$u_{i2} u_{12} + u_{n2} u_{22} = a_{22}$$

$$\vdots$$

$$\vdots$$

$$u_{in}^2 + u_{n2}^2 + \cdots + u_{nn}^2 = a_{nn}$$

If we omit the computations for the redundant symmetric elements, the elements of $U$ can be found as follows:

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{k}^2}$$

for $i = 1, \ldots, n$

$$u_{ij} = \frac{1}{u_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} u_{k}^2 u_{k j} \right)$$

for $i = 1, \ldots, n$ and $j = i+1, \ldots, n$

As before, if the summation limits are such that the upper limit is less than the lower, the sum is taken as zero.
Once A has been decomposed, the solution may be found from $AX = U^T U X = B$ and $X = (U^T U)^{-1} B = U^{-1} (U^T)^{-1} B = U^{-1} (U^{-1})^T B$. The inverse of $U$ can easily be found with the procedure of Eqs. (A.13) and (A.14). Alternately, the more general approach of solving Eqs. (A.11) and (A.12) may be taken.

A.7 Gauss-Seidel Iteration

The basic idea behind iterative methods in general is to guess a solution, try it out, then adjust it. In the basic iterative method for systems of linear algebraic equations, usually attributed to Jacobi, each equation is solved for one of the unknowns by using the trial values for all of the remaining unknowns. The determination of an improved value of $x_i$ may thus be obtained from the $i$th equation as

$$x_i = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{n} a_{ij} x_j \right) \quad (j \neq i) \quad (A.17)$$

During each cycle, $n$ new solutions are obtained by using the $n$ solutions from the previous cycle (the $x$'s on the right-hand side of Eq. (A.17)). If the most recently computed $x_j$, where $j < i$, is used in computing $x_i$, the procedure is termed Seidel or Gauss-Seidel iteration.
There are several convergence conditions which the coefficient matrix $A$ must satisfy. There is one useful all-encompassing criterion, however. If $A$ is symmetric and positive definite, the Gauss-Seidel method will converge for any initial value of $X$. In some cases, however, convergence may be so slow as to render the method impractical (11).
Appendix B

Test Computer Program
**COMMON/A/ (200,600,Y) 200yte DOUBLE PRECISION A/Y
3. NC=600
4. IMAX=60

6. C MATRIX AND THE RIGHT-HAND SIDE VECTOR.
7. C
8. C
9. DC 5 1=1, NC
10. Y(I)=C
11. A(I,1)=2.
12. 5 A(I,2)=1*
13. A(INC+1)=1*
14. A(INC+2)=0*
15. Y(INC+1)=1*
16. DC 10 I=1, NC
17. DC 10 J=5; IMAX
18. A(I,J)=0*
19. C
20. C BEGIN TIMING
21. C
22. CALL STARTIME
23. C
24. C SOLVE THE EQUATIONS
25. C
26. C
27. C CALL SOLBAN(NC, IMAX)
28. C COMPUTE EXECUTION TIME
29. C
30. C CALL TESTTIME(N11)
31. C TIME=FLOAT(N11)/100.
32. C WRITE(16,100) (I,Y(I),I=1,NC)
33. 100 FORMAT(11,15X, 'SOLUTIONS',/19X,'I',20X,'Y(I)'),
34. C /* (120,273).
35. C WRITE(16,200) TIME
37. C
38. C COMPUTE ERROR MEASURES
39. C
40. C SLM=0.
41. C MAX=0.
42. C 50 I=1, NC
43. C IF (ABS(R)>GT*MAX) MAX=ABS(R).
44. C 20 SUM=SUM+R**2
45. C ERROR=SQRT(SUM)/FLOAT(NC)
46. C WRITE(16,300) ERROR
47. C 300 FORMAT(/19X, 'ERROR MEASURE = ',F15.6)
48. C WRITE(16,300) ERROR
49. C 300 FORMAT(/19X, 'ERROR MEASURE = ',F15.6)
50. C 400 FORMAT(/19X, 'MAXIMUM ERROR = ',F15.6)
51. STOP
52. END
SUBROUTINE SOLVE(A,NU)

COMMON/A/ A(200,50)/U(200)

DOUBLE PRECISION A, N, S, SUM

INTEGER M, H, J, I, K

DO 1, I=1,NU

SUB=C

SUM=C

H=I-1

IF(H+L.E.H) H=H1

IF(U.E.GC) GO TO 16

C COMPUTE OFF DIAGONAL TERMS

DO 16, I=1,M

K=1-I

X=1

L=I+1

IF(L.GT.UM) GO TO 11

IF(N.GT.L) GO TO 11

SUM=SUM+A(K+1)*A(K+1)

C CONTINUE

11 CONTINUE

A(I,J)=A(I,J)-SUM/A(I,1)

GO TO 6

C COMPUTE DIAGONAL TERMS

C CONTINUE

IF(H.LE.C) GO TO 16

DO 16, I=1,M

K=I-1

X=1

L=I+1

C1=A(K+1)

SUM=SUM+C1*A(K+1)

C CONTINUE

SUB=SUM2+C1*K*A(K,C)

C CONTINUE

A(I,1)=A(I,1)-SUM

G(I)=G(I)-SUM

C CONTINUE

C CONTINUE

DO 12, I=1,NU

I=I+1

SUM=0

HI=I-1

H=HI+1

IF(H+L.E.H) H=H1

IF(H+L.E.H) GO TO 15

DO 15, K=1,H

J=K

JJ=J+1

SUM=SUM+A(I,J)*A(J)

C CONTINUE

13 G(I)=G(I)/A(I,1)*SUM

C CONTINUE

RETURN

END
<table>
<thead>
<tr>
<th></th>
<th>Y(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000000E+02</td>
</tr>
<tr>
<td>2</td>
<td>1.5000000E+02</td>
</tr>
<tr>
<td>3</td>
<td>2.0000000E+02</td>
</tr>
<tr>
<td>4</td>
<td>2.5000000E+02</td>
</tr>
<tr>
<td>5</td>
<td>3.0000000E+02</td>
</tr>
<tr>
<td>6</td>
<td>3.5000000E+02</td>
</tr>
<tr>
<td>7</td>
<td>4.0000000E+02</td>
</tr>
<tr>
<td>8</td>
<td>4.5000000E+02</td>
</tr>
<tr>
<td>9</td>
<td>5.0000000E+02</td>
</tr>
<tr>
<td>10</td>
<td>5.5000000E+02</td>
</tr>
<tr>
<td>11</td>
<td>6.0000000E+02</td>
</tr>
<tr>
<td>12</td>
<td>6.5000000E+02</td>
</tr>
<tr>
<td>13</td>
<td>7.0000000E+02</td>
</tr>
<tr>
<td>14</td>
<td>7.5000000E+02</td>
</tr>
<tr>
<td>15</td>
<td>8.0000000E+02</td>
</tr>
<tr>
<td>16</td>
<td>8.5000000E+02</td>
</tr>
<tr>
<td>17</td>
<td>9.0000000E+02</td>
</tr>
<tr>
<td>18</td>
<td>9.5000000E+02</td>
</tr>
<tr>
<td>19</td>
<td>1.0000000E+03</td>
</tr>
<tr>
<td>20</td>
<td>1.0500000E+03</td>
</tr>
<tr>
<td>21</td>
<td>1.1000000E+03</td>
</tr>
<tr>
<td>22</td>
<td>1.1500000E+03</td>
</tr>
<tr>
<td>23</td>
<td>1.2000000E+03</td>
</tr>
<tr>
<td>24</td>
<td>1.2500000E+03</td>
</tr>
<tr>
<td>25</td>
<td>1.3000000E+03</td>
</tr>
<tr>
<td>26</td>
<td>1.3500000E+03</td>
</tr>
<tr>
<td>27</td>
<td>1.4000000E+03</td>
</tr>
<tr>
<td>28</td>
<td>1.4500000E+03</td>
</tr>
<tr>
<td>29</td>
<td>1.5000000E+03</td>
</tr>
<tr>
<td>30</td>
<td>1.5500000E+03</td>
</tr>
<tr>
<td>31</td>
<td>1.6000000E+03</td>
</tr>
<tr>
<td>32</td>
<td>1.6500000E+03</td>
</tr>
<tr>
<td>33</td>
<td>1.7000000E+03</td>
</tr>
<tr>
<td>34</td>
<td>1.7500000E+03</td>
</tr>
<tr>
<td>35</td>
<td>1.8000000E+03</td>
</tr>
<tr>
<td>36</td>
<td>1.8500000E+03</td>
</tr>
<tr>
<td>37</td>
<td>1.9000000E+03</td>
</tr>
<tr>
<td>38</td>
<td>1.9500000E+03</td>
</tr>
<tr>
<td>39</td>
<td>2.0000000E+03</td>
</tr>
<tr>
<td>40</td>
<td>2.0500000E+03</td>
</tr>
<tr>
<td>41</td>
<td>2.1000000E+03</td>
</tr>
<tr>
<td>42</td>
<td>2.1500000E+03</td>
</tr>
<tr>
<td>43</td>
<td>2.2000000E+03</td>
</tr>
<tr>
<td>44</td>
<td>2.2500000E+03</td>
</tr>
<tr>
<td>45</td>
<td>2.3000000E+03</td>
</tr>
<tr>
<td>46</td>
<td>2.3500000E+03</td>
</tr>
<tr>
<td>47</td>
<td>2.4000000E+03</td>
</tr>
<tr>
<td>48</td>
<td>2.4500000E+03</td>
</tr>
<tr>
<td>49</td>
<td>2.5000000E+03</td>
</tr>
<tr>
<td>50</td>
<td>2.5500000E+03</td>
</tr>
<tr>
<td>51</td>
<td>2.6000000E+03</td>
</tr>
<tr>
<td>52</td>
<td>2.6500000E+03</td>
</tr>
<tr>
<td>53</td>
<td>2.7000000E+03</td>
</tr>
<tr>
<td>54</td>
<td>2.7500000E+03</td>
</tr>
<tr>
<td>55</td>
<td>2.8000000E+03</td>
</tr>
<tr>
<td>56</td>
<td>5600000000 02</td>
</tr>
<tr>
<td>----</td>
<td>---------------</td>
</tr>
<tr>
<td>57</td>
<td>5700000000 02</td>
</tr>
<tr>
<td>58</td>
<td>5800000000 02</td>
</tr>
<tr>
<td>59</td>
<td>5900000000 02</td>
</tr>
<tr>
<td>60</td>
<td>6000000000 02</td>
</tr>
<tr>
<td>61</td>
<td>6100000000 02</td>
</tr>
<tr>
<td>62</td>
<td>6200000000 02</td>
</tr>
<tr>
<td>63</td>
<td>6300000000 02</td>
</tr>
<tr>
<td>64</td>
<td>6400000000 02</td>
</tr>
<tr>
<td>65</td>
<td>6500000000 02</td>
</tr>
<tr>
<td>66</td>
<td>6600000000 02</td>
</tr>
<tr>
<td>67</td>
<td>6700000000 02</td>
</tr>
<tr>
<td>68</td>
<td>6800000000 02</td>
</tr>
<tr>
<td>69</td>
<td>6900000000 02</td>
</tr>
<tr>
<td>70</td>
<td>7000000000 02</td>
</tr>
<tr>
<td>71</td>
<td>7100000000 02</td>
</tr>
<tr>
<td>72</td>
<td>7200000000 02</td>
</tr>
<tr>
<td>73</td>
<td>7300000000 02</td>
</tr>
<tr>
<td>74</td>
<td>7400000000 02</td>
</tr>
<tr>
<td>75</td>
<td>7500000000 02</td>
</tr>
<tr>
<td>76</td>
<td>7600000000 02</td>
</tr>
<tr>
<td>77</td>
<td>7700000000 02</td>
</tr>
<tr>
<td>78</td>
<td>7800000000 02</td>
</tr>
<tr>
<td>79</td>
<td>7900000000 02</td>
</tr>
<tr>
<td>80</td>
<td>8000000000 02</td>
</tr>
<tr>
<td>81</td>
<td>8100000000 02</td>
</tr>
<tr>
<td>82</td>
<td>8200000000 02</td>
</tr>
<tr>
<td>83</td>
<td>8300000000 02</td>
</tr>
<tr>
<td>84</td>
<td>8400000000 02</td>
</tr>
<tr>
<td>85</td>
<td>8500000000 02</td>
</tr>
<tr>
<td>86</td>
<td>8600000000 02</td>
</tr>
<tr>
<td>87</td>
<td>8700000000 02</td>
</tr>
<tr>
<td>88</td>
<td>8800000000 02</td>
</tr>
<tr>
<td>89</td>
<td>8900000000 02</td>
</tr>
<tr>
<td>90</td>
<td>9000000000 02</td>
</tr>
<tr>
<td>91</td>
<td>9100000000 02</td>
</tr>
<tr>
<td>92</td>
<td>9200000000 02</td>
</tr>
<tr>
<td>93</td>
<td>9300000000 02</td>
</tr>
<tr>
<td>94</td>
<td>9400000000 02</td>
</tr>
<tr>
<td>95</td>
<td>9500000000 02</td>
</tr>
<tr>
<td>96</td>
<td>9600000000 02</td>
</tr>
<tr>
<td>97</td>
<td>9700000000 02</td>
</tr>
<tr>
<td>98</td>
<td>9800000000 02</td>
</tr>
<tr>
<td>99</td>
<td>9900000000 02</td>
</tr>
<tr>
<td>100</td>
<td>1000000000 03</td>
</tr>
<tr>
<td>101</td>
<td>1010000000 03</td>
</tr>
<tr>
<td>102</td>
<td>1020000000 03</td>
</tr>
<tr>
<td>103</td>
<td>1030000000 03</td>
</tr>
<tr>
<td>104</td>
<td>1040000000 03</td>
</tr>
<tr>
<td>105</td>
<td>1050000000 03</td>
</tr>
<tr>
<td>106</td>
<td>1060000000 03</td>
</tr>
<tr>
<td>107</td>
<td>1070000000 03</td>
</tr>
<tr>
<td>108</td>
<td>1080000000 03</td>
</tr>
<tr>
<td>109</td>
<td>1090000000 03</td>
</tr>
<tr>
<td>110</td>
<td>1100000000 03</td>
</tr>
<tr>
<td>111</td>
<td>1110000000 03</td>
</tr>
<tr>
<td>112</td>
<td>1120000000 03</td>
</tr>
<tr>
<td>113</td>
<td>1130000000 03</td>
</tr>
<tr>
<td>114</td>
<td>1140000000 03</td>
</tr>
<tr>
<td>112</td>
<td>116</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>117</td>
<td>118</td>
</tr>
<tr>
<td>119</td>
<td>120</td>
</tr>
<tr>
<td>121</td>
<td>122</td>
</tr>
<tr>
<td>123</td>
<td>124</td>
</tr>
<tr>
<td>125</td>
<td>126</td>
</tr>
<tr>
<td>127</td>
<td>128</td>
</tr>
<tr>
<td>129</td>
<td>130</td>
</tr>
<tr>
<td>131</td>
<td>132</td>
</tr>
<tr>
<td>133</td>
<td>134</td>
</tr>
<tr>
<td>135</td>
<td>136</td>
</tr>
<tr>
<td>137</td>
<td>138</td>
</tr>
<tr>
<td>139</td>
<td>140</td>
</tr>
<tr>
<td>141</td>
<td>142</td>
</tr>
<tr>
<td>143</td>
<td>144</td>
</tr>
<tr>
<td>145</td>
<td>146</td>
</tr>
<tr>
<td>147</td>
<td>148</td>
</tr>
<tr>
<td>149</td>
<td>150</td>
</tr>
<tr>
<td>151</td>
<td>152</td>
</tr>
<tr>
<td>153</td>
<td>154</td>
</tr>
<tr>
<td>155</td>
<td>156</td>
</tr>
<tr>
<td>157</td>
<td>158</td>
</tr>
<tr>
<td>159</td>
<td>160</td>
</tr>
<tr>
<td>161</td>
<td>162</td>
</tr>
<tr>
<td>163</td>
<td>164</td>
</tr>
<tr>
<td>165</td>
<td>166</td>
</tr>
<tr>
<td>167</td>
<td>168</td>
</tr>
<tr>
<td>169</td>
<td>170</td>
</tr>
<tr>
<td>171</td>
<td>172</td>
</tr>
<tr>
<td>173</td>
<td>243</td>
</tr>
</tbody>
</table>
EXECUTION TIME = 20.236 SECONDS

ERROR MEASURE = *1.77332E-10

MAXIMUM ERROR = *3.59890E-39

*STOP*
REFERENCES


"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—National Aeronautics and Space Act of 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons. Also includes conference proceedings with either limited or unlimited distribution.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include final reports of major projects, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546