

**NASA CONTRACTOR
REPORT**



NASA CR-2173

NASA CR-2173

**CASE FILE
COPY**

**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

CASE FILE COPY

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

1. REPORT NO. NASA CR-2173	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.	
4. TITLE AND SUBTITLE Computer Programs for the Solution of Systems of Linear Algebraic Equations		5. REPORT DATE January 1973	
		6. PERFORMING ORGANIZATION CODE M380	
7. AUTHOR(S) William T. Sequi		8. PERFORMING ORGANIZATION REPORT #	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Memphis State University Memphis, Tenn. 38152		10. WORK UNIT NO.	
		11. CONTRACT OR GRANT NO. NCA8-44	
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D.C. 20546		13. TYPE OF REPORT & PERIOD COVERED Contractor Report	
		14. SPONSORING AGENCY CODE	
15. SUPPLEMENTARY NOTES			
16. ABSTRACT 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.			
17. KEY WORDS		18. DISTRIBUTION STATEMENT	
19. SECURITY CLASSIF. (of this report) Unclassified		20. SECURITY CLASSIF. (of this page) Unclassified	
21. NO. OF PAGES 250		22. PRICE \$3.00	

TABLE OF CONTENTS

	Page
SUMMARY	1
I. INTRODUCTION	3
1.1 General	3
1.2 Test Matrix and Procedures	3
II. SUBROUTINE DESCRIPTIONS AND LISTINGS	7
2.1 In-core, Non-banded Equation Solvers	9
SIMEQ	10
SIMEQ3	13
WUGEL	16
GAUSEL	19
SIMEQ2	22
FACTOR and LU	26
GAUSS	30
SOLVIT	34
SIMEQ1	41
DPSIM	45
SHORT	48
SOLVE1	50
WUIGJ	53
WUGJN	56
SLVE	59
GASSIM	62
GJR	65
SIMULT	69
SOLVE3	77
SOLVE2	82
GSEID	88
2.2 In-core, Non-banded Inversion Subroutines	91
AINVER	92
BINVER	94
MATIN1	96
MATIN4	98
MIV	100
INVT	103
INVERT	107

	Page
INVTIT	111
MATIN3	117
INV1	119
INV1A	125
INV2	133
INV3	139
DPMNVT	147
DPINV	151
DPINVT	155
SYMVRT	159
2.3 In-core, Banded Equation Solvers	162
SOLEQ	163
SYMSOL	166
TRIMSS	170
BANSLV	175
DECOM and SOLV	177
CHOLES	181
SOLBAN	186
2.4 Out-of-core Equation Solvers	190
DISPL	191
BANSOL	197
III. DISCUSSION	203
3.1 Non-banded Methods	203
3.2 Banded Solvers	206
3.3 Out-of-core Solvers	208
IV. RECOMMENDATIONS	210
APPENDIX A. METHODS OF SOLUTION	212
A.1 General	212
A.2 Gauss Elimination	214
A.3 Gauss-Jordan Elimination	221
A.4 Decomposition	223
A.5 Decomposition of Symmetric Matrices	230
A.6 Choleski Decomposition	234
A.7 Gauss-Seidel Iteration	236
APPENDIX B. TEST COMPUTER PROGRAM	238
REFERENCES	245

PREFACE

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; CHOLAS, 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:

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

The solutions are $x_i = i$ for $i = 1, \dots, n$ 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 subprograms 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

Source: John M. McCormick and Mario G. Salvadori, Numerical Methods in FORTRAN, (c) 1964, p. 205. Used with permission of Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Method: Gauss elimination (See appendix A).

Usage:

```
DIMENSION A(n,n),Y(n),X(n)
```

```
.
```

```
.
```

```
.
```

```
CALL SIMEQ (A,Y,X,N)
```

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:

```
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 SIMEQ (A, Y, X, N)
  DIMENSION A(100,100),Y(100),X(100)
  DOUBLE PRECISION A,X,Y
  M = N = 1
  DO 10 I=1,M
    L = I + 1
    DO 10 J = L,N
      IF (A(J,I)) 6,10,6
6    DO 8 K=L,N
      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 = I
        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=N
DO 5 J=1,KA
5 B(I,J)=0.0
K2=2
K3=1
DO 10 I=1,N
DO 10 J=1,KA
C 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)40,40,300
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.
DO 220 I=K1,N
220 D=D+B(K3,I)*C(I)
C(K3)=B(K3,KA)-D
K3=K3-1
IF(K3)350,350,210
300 TOL=TOL/10.
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//)

```

350 RETURN
END

WUGEL

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

Method: Gauss elimination (see Appendix A)

Usage:

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

```
.
```

```
.
```

```
.
```

```
CALL WUGEL (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 = N plus 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:

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

```

SUBROUTINE WUGEL(A,N,M)
DIMENSION A(100,101)
NM=N-1
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) 110,100,100
110 DO 100 I=K1,N
DO 100 J=K1,M
A(I,J)=A(I,J)-A(I,K)*A(K,J)
100 CONTINUE
DO 115 L=1,NM
K=M-L
K2=N-L
DO 115 I=1,K2
115 A(I,M)=A(I,M)-A(I,K)*A(K,M)
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

$$\begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & a_{1N} & -b_1 \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & a_{2N} & -b_2 \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ a_{N1} & a_{N2} & \cdot & \cdot & \cdot & a_{NN} & -b_N \end{bmatrix}$$

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
DO1 I=1,ND
IF(I-1)2,2,3
3 M=I-1
DO4 K=1,M
R=A(I,K)
DO4 J=K,N1
4 A(I,J)=R*A(K,J) + A(I,J)
2 D=A(I,I)
DO1J=I,N1
1 A(I,J)=A(I,J)/(-D)
N2=ND-1
DO5 I=1,N2
L=ND-I
M=L+1
DO5 J=M,ND
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:

```
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)+1
TEMP=A(L)
A(L)=A(I)
1 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)
NMNM=N+MM
CALL SIM (A,N,IE,MM)
L=0
DO 2 I=NN,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=1,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
3  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-J
    IF (KK)13,13,10
10 DO 11 I=J,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

Source: Numerical Algorithms: Origins and Applications,
by B. W. Arden and K. N. Astill, Copyright 1970
by Addison-Wesley Publishing Company. Used with
permission of the publishers.

Method: Decomposition (see Appendix A).

Usage:

DIMENSION A(n,n),B(n,r),L(n)

.

.

.

DETER = FACTOR(N,A,L)

CALL LU(A,L,N,B,M)

where

A = coefficient matrix

B = matrix of right-hand side constants (consisting
of one or more constant vectors)

L = working storage

n = 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: DIMENSION A(n,n),P(n)

In LU: DIMENSION A(n,n),P(n),B(n,r),T(n)

```

FUNCTION FACTOR(N,A,P)
DIMENSION A(100,100),P(100)
REAL MAX
INTEGER P,PK,PI
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
14 A(PI,J)=A(PI,J)-A(PI,K)*A(PK,J)
11 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(1)=B(P(1),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=IP1,N
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

Source: Applied Numerical Methods for Digital Computation
with FORTRAN, by M. L. James, G. M. Smith, and J. C.
Wolford. Copyright 1967, by International Text-
book Company. Used with permission of Intext
Educational Publishers.

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 statment 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)
C
C   REFERENCE = APPLIED NUMERICAL METHODS FOR DIGITAL
C               COMPUTATION, BY JAMES, SMITH, AND
C               WOFFORD.
C
C   SOLUTION OF SIMULTANEOUS EQUATIONS BY
C               GAUSSIAN ELIMINATION
C
      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
C
C   SEARCH FOR LARGEST POSSIBLE PIVOT ELEMENT
C
        DO 7 I=KP1,N
          AB=ABS(A(I,K))
          IF(BIG-AB)6,7,7
        6 BIG=AB
          JJ=I
        7 CONTINUE
C
C   DECISION ON NECESSITY OF ROW INTERCHANGE
C
        IF(JJ=K)8,10,8
C
C   ROW INTERCHANGE
C
        8 DO 9 J=K,M
          TEMP=A(JJ,J)
          A(JJ,J)=A(K,J)
        9 A(K,J)=TEMP
C
C   CALCULATION OF ELEMENTS OF NEW MATRIX
C
        10 DO 11 I=KP1,N
          QUOT=A(I,K)/A(K,K)
          DO 11 J=KP1,M
        11 A(I,J)=A(I,J)-QUOT*A(K,J)
          DO 12 I=KP1,N
        12 A(I,K)=0.
C
C   BACK-SUBSTITUTION
C
      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

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

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 SIMEQA(A,N,M,ACC,MAXIT,IT,IN,W(1,1),
I      W(1,N+1),W(1,N+2),W(1,N+3),W(1,N+4),W(1,N+5))
      CALL SIMEQB(B,X)
      RETURN
      END

```

```

SUBROUTINE SIMEQA(AA,NN,MM,EPS,MAXIT,IT,IN,A,B,FACT,ITOP,Y,Z)
DOUBLE PRECISION SUM
DIMENSION A(NN,NN),B(1),Z(1),Y(1),AA(MM,MM),FACT(1),ITOP(1)
GO TO 400
ENTRY SIMEQB(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.0.) K=0
   IF(A(I,1).LE.0.) GO TO 1000
   K=ALOG(A(I,1))
1000 DO 130 J=2,N
   IF(A(I,J).LE.0.) GO TO 130
   K=MAX0(K,ALOG(A(I,J)))
130  CONTINUE
   FACT(I)=2.**(-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.) 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)

```

```

      AA(M,I)=AA(IMAX,I)
      AA(IMAX,I)=TEMP
180    CONTINUE
190    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
      DO 200 J=MP,N
      A(I,J)=A(I,J)-A(M,J)*EM
200    CONTINUE
210    CONTINUE
220    CONTINUE
      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
      DO 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 330 K=1,N
      I=NP-K
      SUM=0.
      IF (K .EQ. 1) GO TO 320
      IP=I+1

```

```

      DO 310 J=1,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(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(1))
      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

```

SMEQ1

Source: Synthesis of Dynamic Systems Using FORMA (FORTRAN Matrix Analysis), a report prepared for NASA under MSFC Contract NAS8-25922 by Martin Marietta Corporation, May 1971.

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

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 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 SMEQ1 (A,B,Z,N,KR,NERROR)
DIMENSION A(KR,1),B(1),Z(1)
DATA TOL/1.E-15/

```

```

C
C SOLUTION OF LINEAR SIMULTANEOUS ALGEBRAIC EQUATIONS,  $A \cdot Z = B$ .
C GAUSS ELIMINATION METHOD. FORWARD SOLUTION TRANSFORMS ORIGINAL SYSTEM
C INTO TRIANGULAR FORM. BACK SOLUTION THEN GIVES RESULT.
C LARGEST PIVOTAL DIVISOR IS USED TO AVOID DIVISION BY SMALL NUMBERS.
C THE ROWS ARE INTERCHANGED WHEN NECESSARY TO ACCOMPLISH THIS.
C IF NO PIVOT CAN BE FOUND EXCEEDING  $1.E-15$ , THE MATRIX IS CONSIDERED
C SINGULAR AND THE PROGRAM STOPPED.
C CODED BY CARL BODLEY. AUGUST 1968.
C MODIFIED FOR CONTRACT NAS8-25922, MAY 1971.

```

SUBROUTINE ARGUMENTS

```

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

```

```

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

```

C FORWARD SOLUTION.

```

      5 DO 25 L=1,N
        AMAX = TOL
        DO 10 I=L,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(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
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

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

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: $A_{N1} = X_N$,

$A_{N2} = X_{N-1}, \dots, A_{NN} = X_1$

The array size in the DOUBLE PRECISION statement in
DPSIM must conform to the following:

```
DOUBLE PRECISION A(n,n + 1),DET,SIGN,AMAX,  
ABPIV,DIV,SS
```

```

SUBROUTINE DPSIM(A,N,DET)
DOUBLE PRECISION A(100,101),DET,SIGN,AMAX,ABPIV,DIV,SS
MM=N+1
SS=1.000
DET=1.000
NN=N-1
DO 8 K=1,NN
  AMAX=0.000
  DO 2 I=K,N
    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=0.000
   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,N
    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,N
9  DET=DET*A(I,I)
   DET=DET*SS
   JJ=N
   DO 12 J=1,N
     A(N,J)=A(JJ,MM)/A(JJ,JJ)
     KK=N-J
     IF (KK)13,13,10
10  DO 11 I=J,NN
     A(KK,MM)=A(KK,MM)-A(N,J)*A(KK,JJ)
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(1,J+1)/A(1,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

Source: Applied Numerical Methods for Digital Computation with FORTRAN, by M. L. James, G. M. Smith, and J. C. Welford. Copyright 1967, by International Textbook Company. Used with permission of Intext Educational Publishers.

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

Usage:

```
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)
C     SOLUTION OF SIMULTANEOUS EQUATIONS BY GAUSS = JORDAN ELIMINATION
C
C     ZERO DIAGONAL ELEMENTS ARE PERMITTED.
C
C     THE ORIGINAL ELEMENTS OF A, THE AUGMENTED MATRIX, ARE LOST.
C
C     THE SOLUTIONS ARE STORED IN THE FIRST COL OF THE AUGMENTED MATRIX.
C
      DIMENSION A(100,101),B(100,100)
      M = N + 1
      5 IF(A(1,1)) 11,6,11
      6 K = M-1
      DO 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(1,J)
      8 A(1,J) = TEMP
      GO TO 11
      9 CONTINUE
      WRITE(3,10)
      10 FORMAT(//// 19H NO UNIQUE SOLUTION)
      GO TO 18
      11 DO 12 J=2,M
      DO 12 I=2,N
      12 B(I-1,J-1) = A(I,J)-A(1,J)*A(I,1)/A(1,1)
      DO 13 J=2,M
      13 B(N,J-1) = A(1,J)/A(1,1)
      M = M-1
      DO 14 J=1,M
      DO 14 I=1,N
      14 A(I,J) = B(I,J)
      IF(M=1) 5,18,5
      18 RETURN
      END

```

WUIGJ

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:

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

```

SUBROUTINE WUIGJ(A,N,NSOL,DETA)
DIMENSION A(100,105)
DETA=1.
NCOL=N+NSOL
DO 10 K=1,N
IF(A(K,K))20,30,20
30 DETA=0.
RETURN
20 DETA=DETA*A(K,K)
DO 40 J=1,NCOL
IF(J=K)50,40,50
50 A(K,J) = A(K,J)/A(K,K)
40 CONTINUE
A(K,K) = 1./A(K,K)
DO 10 I=1,N
IF(I=K) 60,10,60
60 DO 70 J =1,NCOL
IF(J=K)80,70,80
80 A(I,J) = A(I,J)=A(I,K)*A(K,J)
70 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=1) 120,100,120
120  DO 130 J=K1,M
130  A(I,J)=A(I,J)-A(I,K)*A(K,J)
100  CONTINUE
      RETURN
      END

```

SLVE

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 (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)
C
C  REFERENCE = NUMERICAL AND MATRIX METHODS IN STRUCTURAL
C              MECHANICS, BY P. C. WANG
C
      DIMENSION A(100,100),U(100)
      DO 9015 I=1,NM
        K=I
        IF(I=NM)9021,9007,9021
C
C  TEST TO SEE WHETHER DIAGONAL ELEMENT IS ZERO
C
      9021 IF(ABS(A(I,I))-EPS)19006,9006,9007
C
C  ADD SUBSEQUENT EQUATIONS TO THE CURRENT ONE
C
      9006 K=K+1
        U(I)=U(I)+U(K)
        DO 9023 J=1,NM
      9023 A(I,J)=A(I,J)+A(K,J)
        GO TO 9021
      9007 DIV=A(I,I)
        U(I)=U(I)/DIV
C
C  DIVIDE ALL THE ELEMENTS OF I-TH EQ. BY A(I,I)
C
        DO 9009 J=1,NM
      9009 A(I,J)=A(I,J)/DIV
C
C  REDUCE THE I-TH ELEMENT OF THE OTHER EQUATIONS
C  TO ZERO.
C
        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
      9015 CONTINUE
          RETURN
        END

```

GASSIM

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

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 = coefficient matrix

B = matrix of right-hand side constants

n = maximum permissible number of equations

r = maximum permissible number of right-hand side
constant vectors

N = 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/ 2500000000/
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=K,N
TEMP=A(L,J)
A(L,J)=A(K,J)
5 A(K,J)=TEMP
DO 15 J=1,NN
TEMP=B(L,J)
B(L,J)=B(K,J)
15 B(K,J)=TEMP
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
16 B(K,J)=B(K,J)*A(K,K)
DO 7 I=1,N
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(IE)-IABS(ID).GT.IG) D=0.
A(I,J)=D
8 CONTINUE
DO 18 J=1,NN
18 B(I,J)=B(I,J)-FAC*B(K,J)
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 GJR(A,NC,NR,N,L,NN,JC)
DIMENSION A(NR,NC),JC(1)
C      -----
C      SEARCH FOR PIVOT ROW
C      -----
DO 91 I=1,N
M=I
IF(I.EQ.N) GO TO 60
X=1.
DO 30 J=I,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
C      INTERCHANGE ROW I AND ROW K
C
DO 50 J=M,L
X=A(I,J)
A(I,J)=A(K,J)
50 A(K,J)=X
C      TEST FOR SINGULARITY
C
60 IF(ABS(A(I,I)).GT.0.) GO TO 72
C      MATRIX IS SINGULAR
C
150 JC(1)=I-1
RETURN NN
72 X=A(I,I)
A(I,I)=1.
C      NORMALIZATION OF THE I=,H ROW
C
DO 80 J=M,L
A(I,J)=A(I,J)/X
C      TEST OVERFLOW SWITCH. IF ON,
C      RETURN NEGATIVE VALUE OF I IN JC(1)
C
CALL OVERFL(IFL)
IF(IFL.EQ.1) GO TO 150
80 CONTINUE
C      REDUCTION OF ALL REMAINING ROWS
C      -----
DO 91 K=1,N
IF(K.EQ.I) GO TO 91

```

```

X=A(K,I)
A(K,I)=0.
DO 90 J=M,L
A(K,J)=A(K,J)+X*A(I,J)
C
C      TEST OVERFLOW SWITCH. IF ON,
C      RETURN NEGATIVE VALUE OF I IN JC(1)
C
CALL OVERFL(IFL)
IF(IFL.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,1) = 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,NLNOD,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
IC=ICOL(I,J)
IF(IC.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).GE.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).GE.IK.OR.IRC(II).LE.0) GO TO 104

```

```

    PIVCOL = II
    IK=IRC(II)
    IY=I
104 CONTINUE
C   NORMALIZE PIVROW
    X=A(PIVROW,IY)
    IC=ICC(PIVROW)
    DO 5 J=1,IC
    5  A(PIVROW,J)=A(PIVROW,J)/X
    A(PIVROW,IY)=1.0
    PLOAD(PIVROW)=PLOAD(PIVROW)/X
C   SELECT ROWS THAT CAN BE OPERATED ON
    DO 106 I=1,NU
    IF(I.EQ.PIVROW.AND.IRC(PIVCOL).EQ.1)GO TO 107
    IF(I.EQ.PIVROW) GO TO 106
    IC=ABS(ICC(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(PIVROW,JKPI).EQ.0) GO TO 106
    IF(ICOL(OPROW,JKOP).EQ.0) GO TO 80
    IF(ICOL(PIVROW,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
    II=ABS(ICC(I))
    IF(II.GT.JCOL) JCOL=II
    IF(II.LE.NAA) GO TO 85
    WRITE(OUT,400)II
    STOP
85  JKL=JKOP+1
90  CONTINUE
    IX=II-1
    A(OPROW,II)=A(OPROW,IX)
    ICOL(OPROW,II)=ICOL(OPROW,IX)
    II=IX
    IF(II.GE.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

```

```

C   PIVROW AND OPROW CONTAIN THIS ELEMENT. SHIFT BOTH AND
C   OPERATE ON OPROW.
81  IX=ICOL(OPROW,JKOP)
    IF (IX.EQ.PIVCOL) GO TO 11
    X=A(PIVROW,JKPI)*C+A(OPROW,JKOP)
    A(OPROW,JKOP)=X
C   TEST OPROW TO SEE WHAT WAS ELIMINATED
    ATEST=ABS(X)-ZTEST
    IF (ATEST.GT.0.) GO TO 83
11  IRC(IX)=IRC(IX)-1
    JELIM=JELIM+1
    ICC(OPROW)=ICC(OPROW)-1
    IF (ICC(OPROW)) 140,141,142
141 CONTINUE
    IB=1
    DO 10 M=1,NU
    II=ICC(M)
    IF (II.GT.IB) IB=II
10  CONTINUE
    IB=IB+1
    WRITE(OUT,999) LKJ,PIVROW,PIVCOL,OPROW
999 FORMAT('1 MATRIX SINGULAR',//,' NO. CYCLES COMPLETED='
1,I5,' PIVROW=',I5,' PIVCOL=',I5,' OPROW=',I5)
    WRITE(OUT,801) (M,IRC(M),ICC(M),M=1,NU)
801 FORMAT('0 COL./ROW NO.,NO. COL. ENTRIES,NO. ROW ENTRIES'
1/(6(I7,2I5)))
    WRITE(OUT,6)
    CALL RITE(1,NU,IB, NAA,A,ICOL)
    WRITE(OUT,200)
    CALL RITE(2,NU,IB, NAA,A,ICOL)
6   FORMAT('1 COEFFICIENT MATRIX')
200 FORMAT('1 COLUMN IDENTIFICATION')
    STOP
140 CONTINUE
    ICC(OPROW)=ICC(OPROW)+2
142 IX=ABS(ICC(OPROW))
    DO 131 NK=JKOP,IX
    A(I,NK)=A(I,NK+1)
131 ICOL(I,NK)=ICOL(I,NK+1)
    IX=IX+1
    A(I,IX)=0.
    ICOL(I,IX)=0
    JKPI=JKPI+1
    GO TO 79
83  JKPI=JKPI+1
C   PIVROW DOES NOT CONTAIN THIS ELEMENT. SHIFT OPROW AND CONTINUE.
82  JKOP=JKOP+1
    GO TO 79
105 CONTINUE
106 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/108/
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('1')
104 FORMAT(' ',I5,30I4)
105 FORMAT(' ',I5,12G10.3)
END

```

SOLVE3

Source: The Finite Element Method in Engineering Science,
by O. C. Zienkiewicz. Copyright 1971 McGraw-Hill
Publishing Company Limited. Used with permission
of McGraw-Hill Book Company.

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

```
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
```


IF(IN)365,480,365

C
C
C

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)

C
C
C

LOOP ON COLUMN TO BE ELIMINATED

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

C
C
C

MODIFY TERM OF MATRIX

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

C
C
C

MODIFY LOAD VECTOR

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

C
C
C

RESET ROW FOR BACK-SUBSTITUTION

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

C
C
C

BACK-SUBSTITUTE

DO 560 IB=1,NEQM
I=NEG-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

Source: The Finite Element Method in Engineering Science,
by O. C. Zienkiewicz. Copyright 1971. McGraw-
Hill Publishing Company Limited. Used with
permission of McGraw-Hill Book Company.

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+1),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 $n \times n$ 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.

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),NFI(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

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

TOLER = .1E-6

RELAX = 1.8

accordingly.

The maximum number of iterations is given in the statement

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

$$\text{DIS}(N) = \text{DIS}(N) + \text{RLAX} * \text{DX}$$

was changed to

$$\text{DIS}(N) = \text{DIS}(N) + \text{RELAX} * \text{DX}$$

(this was a typographical error in the original version).

4. The statement

$$\text{TOLER} = .1\text{E}-3$$

was changed to

$$\text{TOLER} = .1\text{E}-6$$

The user may wish to streamline this subroutine somewhat by deleting the extraneous entries from the COMMON statements.

```

C      SUBROUTINE SOLVE2
C      SUBROUTINE FOR ITERATIVE SOLUTION OF EQUATIONS
C      INTEGER OUT/108/
C      COMMON/CONTR/TITLE(20),NP,NE,NB,NDF,NCN,NLD,NMAT,NEG,LI,NT4
C      COMMON CORD(100,2),NOP(200,4),IMAT(200),ORT(25,2),NBC(25),NFIX(25)
C      1,R(200),A(200,20),ITEM(200,20),DIS(200)
C      NT=20
C
C      BUILT IN RELAXATION AND TOLERANCE
C
C      TOLER=.1E-6
C      RELAX=1.8
C
C      NEGATIVE NEG SKIPS SETTING UP
C      OF INITIAL DATA
C
C      IF(NEG.LT.0) GO TO 310
C      DO 300 N=1,NEG
C      DO 250 M=1,NT
C      IF(ITEM(N,M).NE.0) GO TO 250
C
C      ITEM(N,1) CONTAINS COUNT OF
C      NUMBER OF OFF DIAGONAL TERMS
C
C      ITEM(N,1)=M-1
C      GO TO 260
C      250 CONTINUE
C      260 CONTINUE
C      300 CONTINUE
C      310 NEG=IABS(NEG)
C
C      SET MAX NO OF CYCLES
C
C      NCYC=15000
C      IF(NCYC.LT.25) NCYC=25
C
C      UNKNOWNNS ARE SET TO ZERO
C
C      DO 320 N=1,NEG
C      IF(A(N,1).NE.0.)A(N,1)=1./A(N,1)
C      320 DIS(N)=0
C
C      LOOP ON CYCLES
C
C      DO 500 NC=1,NCYC
C      SUM=0.
C      SUMD=0.
C
C      THEN ON EQUATIONS
C
C      DO 450 N=1,NEG
C      FX=R(N)
C      NUM=ITEM(N,1)

```

```

      DO 330 M=2,NUM
      L=ITEM(N,M)
330  FX=FX-A(N,M)*DIS(L)
C
C          FX IS TOTAL UNBALANCE OF RHS
C          DX IS THE CHANGE
C
      DX=A(N,1)*FX-DIS(N)
      DIS(N)=DIS(N)+RELAX*DX
C
C          SUM AND SUMD ARE CONVERGENCE
C          PARAMETERS
C
      SUM=SUM+ABS(DX)
      SUMD=SUMD+ABS(DIS(N))
450  CONTINUE
C
C          SKIP OUT OF LOOP IF CONVERGED
C
      ND=NC
      IF(SUM.LT.SUMD*TOLER) GO TO 550
500  CONTINUE
C
C          MOVE FINAL RESULTS TO R
C
550  DO 600 N=1,NEQ
600  R(N)=DIS(N)
C
C          PRINT LAST VALUE OF SUM ETC
C
      WRITE(OUT,10)ND,SUM,SUMD
10  FORMAT(18H0  LAST CYCLE NO.=,I10
1, / 18H (SN=SN-1)/SN = ,E10.3
2, / 18H          SN = ,E10.3)
      RETURN
      END

```

GSEID

Source: Applied Numerical Analysis, by C. F. Gerald,
copyright 1970, by Addison-Wesley Publishing Company.
Used with permission of the publishers.

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

Usage:

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

```

C
C
C
C
C
C
C
C
C
C
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=0.
      DO 20 J=1,N
      SAVE=X(J)
      X(J)=A(J,NP1)
      DO 21 K=1,N
      IF(J=K)22,21,22
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))23,20,20
23  DIFF=ABS(X(J)-SAVE)
20  CONTINUE
      IF(TOL=DIFF)30,40,40
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 AINVER(A,N,N1)
  DIMENSION A(N1,N1)
  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 + 1),B(n + 1,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
15 A(I,J) = C(I,J)
K = N + 1
A(1,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(1,JC+1)/A(1,1)
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(A(IZ,IZ).EQ.0.) GO TO 6
IF(ABS(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 MATIN4(ARRAY,N)
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=1)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:

```
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.

C MATRIX INVERSION BY GAUSS-JORDAN METHOD, BY P. C. WANG
C

```
      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=I
      IF(I=NM)9021,9007,9021
9021  IF(A(I,I)=EPS)9005,9006,9007
9005  IF(-A(I,I)=EPS)9006,9006,9007
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)=EPS)9015,9015,9016
9016  IF(MM=I)9010,9015,9010
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

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

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:

```
DIMENSION A(n,n),IPIV(n),INDEX(n,2)
```

```

      SUBROUTINE INVT(A,N,M,DETER)
C     MATRIX INVERSION AND SIMULTANEOUS EQUATIONS SOLVER
C     A=INPUT MATRIX FOR INVERISON OR AUGUMENTED MATRIX FOR SIME. EQS.
C     N=ORDER OF COEFFICIENT MATRIX
C     M=0 FOR INVERISON ONLY
C     M=NUMBER OF CONSTANT VECTORS
C     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.E-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,N
     35 AMAX=0.0
     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)55,75,250
     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

```

```

175 DO 180 I=1,NN
176 A(L,I)=A(L,I)-A(IC,I)*AMAX
177 IF (I=N)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)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
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+I
16 TEMP(K)=I
  DO 9 IJK=1,N
    M=M+1
    AMAX=ABS(A(1,1))
    IROW=1
    IF(M.EQ.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=I
2    CONTINUE
3    IF(AMAX) 14,14,4
4    T=1.0/A(IROW,1)
    DO 5 I=2,N
      TEMP(I-1)=A(IROW,I)*T
    DO 6 J=1,N
6    A(IROW,J)=A(1,J)
    DO 7 I=2,N
      DO 15 J=2,N
        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=TOL) 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
      J=N+I
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
      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 (//20X,42H MATRIX IS SINGULAR, NO INVERSE OBTAINABLE//)
      GO TO 1000
      END

```

INVTIT

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

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 INVTIT (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)
C  A      INPUT MATRIX, SQUARE
C  N      NUMBER OF ROWS IN MATRICES A AND B. (B IS THE INVERSE MATRIX)
C  M      NUMBER OF ROWS OF THE ARRAYS WHERE MATRICES A AND B ARE STORED
C  B      ARRAY WHERE THE INVERSE OF MATRIX A IS TO BE STORED
C  MAXIT  MAXIMUM NUMBER OF ITERATIONS TO BE ATTEMPTED.
C          PROGRAM SETS MAXIT=-1 IF MATRIX IS SINGULAR
C  W      WORKING STORAGE. N*(N+6) WORDS
      DIMENSION W(N,M),B(M,M)
      CALL SIMEQA(A,N,M,1.E-8,MAXIT,IT,IN,W(1,2),W(1,N+2),
C          W(1,N+3),W(1,N+4),W(1,N+5),W(1,N+6))
      IN=1
      DO 110 J=1,N
      DO 100 I=1,N
      W(I,1)=0.
100  CONTINUE
      W(J,1)=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,A,B,FACT,ITOP,Y,Z)
DOUBLE PRECISION SUM
DIMENSION A(NN,NN),B(1),Z(1),Y(1),AA(MM,MM),FACT(1),ITOP(1)
GO TO 400
ENTRY SIMEGB(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.0.) K=0
IF(A(I,1).LE.0.) GO TO 1000
K=ALOG(A(I,1))
1000 DO 130 J=2,N
IF(A(I,J).LE.0.) GO TO 130
K=MAX0(K,ALOG(A(I,J)))
130 CONTINUE
FACT(I)=2.**(-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.) 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)

```

```

      AA(M,I)=AA(IMAX,I)
      AA(IMAX,I)=TEMP
180  CONTINUE
190  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
      DO 200 J=MP,N
      A(I,J)=A(I,J)-A(M,J)*EM
200  CONTINUE
210  CONTINUE
220  CONTINUE
      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
      DO 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 330 K=1,N
      I=NP+K
      SUM=0.
      IF (K .EQ. 1) GO TO 320
      IP=I+1

```

```

      DO 310 J=1,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(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(1))
      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 380 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 MATIN3(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
5    B3(I,J) = C(I)*A(N,J)
      DO 6 I=1,M
      DO 6 J=1,M
6    A(I,J) = A(I,J) - B3(I,J)
      RETURN
      END

```

INV1

Source: Synthesis of Dynamic Systems Using FORMA (FORTRAN Matrix Analysis), a report prepared for NASA under MSFC Contract NAS8-25922 by Martin Marietta Corporation, May 1971.

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 INVL 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 INVL.

These changes were made for the purpose of obtaining a "stand-alone" subroutine not requiring any supporting subroutines.

```

SUBROUTINE INV1 (A,Z,N,KR,NERROR)
DIMENSION A(1), Z(1), IX(150), B(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.

```

```

2000 FORMAT (// 10X,10(7X,1H(,I2,1H)))
2001 FORMAT (// 10X,45H SUBROUTINE INV1 HAS CALCULATED THE DATA BELOW
*          ///10X,44H THE DETERMINANT RATIOS DET(I+1) / DET(I) ARE
*          // (13X,10E11.3))
2002 FORMAT (///10X,37H THE (A**=1)*(A) INVERSION CHECK GIVES
*          ///10X,25H THE DIAGONAL ELEMENTS ARE // (13X,10F11.8))
2003 FORMAT (// 10X,35H THE MAXIMUM OFF-DIAGONAL ELEMENT IS
*          E11.3, 2X, 4H AT ( I3, 1H, I3, 1H) )

```

```

C
C                                     NERROR=1
IF (N .GT. 150) RETURN

```

```

C
DO 160 I=2,N
160 IX(I) = 1
C INVERT FIRST NON-ZERO ELEMENT IN FIRST COLUMN.
DO 190 I=1,N
190 IF (A(I) .NE. 0.) GO TO 220.

```

```

C                                     NERROR=2

```

```

RETURN

```

```

C
C START INVERSION WITH ROW I.
220 DETR(1) = 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

```

```

      LI = L - 1
250 S = 0.
      MIXL = KR * (IX(L) - 1)
      LL = IX(L) + MIXL
      DO 450 I=1,LI
          MIXI = KR * (IX(I) - 1)
          LI = IX(L) + MIXI
          B(I) = 0.
          G(I) = 0.
          DO 440 J=1,LI
              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
440 G(I) = G(I) + A(LJ)* Z(JI)
450 S = S + A(LI)* B(I)
      AL = A(LL)+ S
      IF (A(LL) .EQ. 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 L = IX(L)
      IX(L) = IX(K)
      IX(K) = IX L
      GO TO 250
540 IF (ALBAR .GE. .1E-8) GO TO 550

```

NERROR=3

RETURN

C

```

550 Z(LL)= 1. / AL
      DETR(L) = AL
      DO 570 I=1,LI
          IL = IX(I) + MIXL
          LI = IX(L) + KR * (IX(I) - 1)
          Z(IL) = B(I) * Z(LL)
          Z(LI) = G(I) * Z(LL)
          DO 570 J=1,LI
              IJ = IX(I) + KR * (IX(J) - 1)
570 Z(IJ)= Z(IJ)+ G(J) * Z(IL)
580 CONTINUE

```

C

C COMPUTE INVERSION CHECK Z*A.
XOFF = 0.0

```

      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
C  PRINT THE DETERMINANT RATIO AND INVERSION CHECK.
      WRITE (NOT,2000) (JC, JC=1,10)
      WRITE (NOT,2001) (DETR(I), I=1,N)
      WRITE (NOT,2002) ( G  (I), I=1,N)
      WRITE (NOT,2003) XOFF,IOFF,JOFF
      NERROR=0
      RETURN
C
      END

```

INV1A

Source: Synthesis of Dynamic Systems Using FORMA (FORTRAN Matrix Analysis), a report prepared for NASA under MSFC Contract NAS8-25922 by Martin Marietta Corporation, May 1971.

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:

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

If n is anything other than 120, the statement

```
IF(N.GT.120)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 INVLA.

These changes were made for the purpose of obtaining a "stand-alone" subroutine not requiring any supporting subroutines.


```

Z(I) = 1. / A(I)
IF (N .EQ. 1) RETURN
IX(I) = 1
IX(1) = I
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)
LI = 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(LJ)* Z(JI)
30 S = S + A(LI)* B(I)
AL = A(LL)+ S
IF (A(LL) .EQ. 0.) 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
DETR(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) + G(J) * Z(IL)
65 CONTINUE

```

NERROR=3

```

C
C PRINT THE DETERMINANT RATIOS.
  WRITE (NOT,2000) (JC, JC=1,10)
  WRITE (NOT,2001) (DETR(I), I=1,N)
C
C INVERSION IMPROVEMENT LOOP.
  E = 1.E+8
  DO 405 LIJ=1,10
C MULTIPLY (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) = 1
  DO 210 I=1,N
210 IF (WORK(I) .NE. 0.) GO TO 215

  RETURN
215 DETR(1) = WORK(1)
  WORK(1) = 1. / WORK(1)
  IX(1) = 1
  IX(1) = 1
  DO 265 L=2,N
    K = L
    L1 = L - 1
220 S = 0.
    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.
      G(I) = 0.
    DO 225 J=1,L1
      MIXJ = KR * (IX(J) - 1)
      IJ = IX(I) + MIXJ
      JL = IX(J) + MIXL

```

NERROR=4

```

      B(I) = B(I) + WORK(IJ)* WORK(JL)
      JI = IX(J) + MIXI
      LJ = IX(L) + MIXJ
225  G(I) = G(I) + WORK(LJ)* WORK(JI)
230  S = S + WORK(LI)* 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

```

NERROR=5

```

      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(IL)
265  CONTINUE
C  MULTIPLY ((Z*A)**-1)*(Z) AND TEST CONVERGENCE.
      DEL IJ = 0.
      MJ = 0
      DO 385 J=1,N
      DO 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 IU .GE. DEL I) GO TO 375
      DEL IU = DEL I
375  DO 380 I=1,N
      IU = I + MU
      Z(IU) = COL(I)
380  CONTINUE
      MU = MU + KR
385  CONTINUE
      D = DEL IU
      IF (D .LE. 1.E-7) GO TO 415
      IF (D .GE. E) GO TO 410
      E = D
405  CONTINUE
      GO TO 415
C
410  WRITE (NCT,2007) E,IU
      GO TO 420
415  WRITE (NCT,2008) D,IU
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

```

INV2

Source: Synthesis of Dynamic Systems Using FORMA (FORTRAN Matrix Analysis), a report prepared for NASA under MSFC Contract NAS8-25922 by Martin Marietta Corporation, May 1971.

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:

```
DIMENSION A(KR,1),Z(KR,1),W(n),U(n),IV(n),  
           IRE(n),BIN(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 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,NERROR)
DIMENSION A(KR,1),Z(KR,1),W(150),U(150),IV(150),
+ IRE(150),BIN(150)
DATA NIT,NOT/105,108/

C
C MATRIX INVERSION (A**=1 = Z). RANK ANNIHILATION METHOD.
C ALGORITHM FORMULATED BY CARL BODLEY.
C THE INVERSION CHECK Z*A IS CALCULATED AND PRINTED.
C THE MAXIMUM SIZE IS
C   N = 150
C CODED BY CARL BODLEY. JANUARY 1967.
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,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,10F11.8))
2003 FORMAT(/ / 10X,35HTHE MAXIMUM OFF-DIAGONAL ELEMENT IS
+ E11.3, 2X, 4HAT (13,1H,13,1H) )

C
NERROR=1
IF(N.GT.150) RETURN
NERROR=2
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)=I
5 IV(I)=I

C
CONDITION A FOR MAXIMUM DIAGONAL ELEMENTS.
NM1=N-1
DO 6 L=1,NM1
SMAX=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

```

```

      IF (ABS(A(K,I)) .LE. SMAX) GO TO 8
      JMAX=J
      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.
      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.
      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.1.0E-35) GO TO 60

```

NERROR=3

```

      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

```

```

      W(I)=0.0
      DO 25 J=1,N
25   W(I)=W(I)+A(LA,J)*Z(J,I)
      S=1.0+W(LA)
      DO 30 I=1,N
30   U(I)=Z(I,LA)
      DO 35 I=1,N
      DO 35 J=1,N
35   Z(I,J)=Z(I,J)-U(I)*W(J)/S

C
C   RESTORE A
65   DO 40 L=1,N
      LA=IRE(L)
      I=L
      K=LA
      IF(IT.EQ.2) I=LA
      IF(IT.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.
      XOFF=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.XOFF) GO TO 45
      XOFF=X
      IOFF=I
      JOFF=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) XOFF,IOFF,JOFF
      NERROR=0
      RETURN
98   Z(1,1)=1.0/A(1,1)
      NERROR=0
      RETURN
      END

```

INV3

Source: Synthesis of Dynamic Systems Using FORMA (FORTRAN Matrix Analysis), a report prepared for NASA under MSFC Contract NAS8-25922 by Martin Marietta Corporation, May 1971.

Method: Choleski decomposition (see Appendix A).

Usage:

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

      .
      .
      .

CALL INV3(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.

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,1),Z(KR,1),W1(150),W2(150),DETR(150)
  DATA NIT,NOT/105,108/

C
C  MATRIX INVERSION (A**=1 = Z). METHOD USES TRIANGULAR DECOMPOSITION
C  AND TRIANGULAR INVERSION. MATRIX A SHOULD BE SYMMETRIC, POSITIVE
C  DEFINITE. UPPER HALF OF MATRIX A IS USED TO CALCULATE Z. FULL
C  MATRICES A,Z ARE USED FOR INVERSION CHECK.
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  CALLS FORMA SUBROUTINES DCOM1,INV4.
C  THE MAXIMUM SIZE IS
C    N=150
C  CODED BY CARL BODLEY. MARCH 1969.
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,1H(,I2,1H)))
2001 FORMAT(// 10X,45H SUBROUTINE INV3 HAS CALCULATED THE DATA BELOW
+      ///10X,44H THE DETERMINANT RATIOS DET(I+1) / DET(I) ARE
+      /// (13X,10E11.3))
2002 FORMAT(///10X,37H THE (A**=1)*(A) INVERSION CHECK GIVES
+      ///10X,25H THE DIAGONAL ELEMENTS ARE // (13X,10F11.8))
2003 FORMAT(// 10X,35H THE MAXIMUM OFF-DIAGONAL ELEMENT IS
+      E11.3,2X,4H AT ( 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  W1(I)=Z(L,I)
  DO 30 I=1,N
  W2(I)=0.0
  DO 30 K=L,N
30  W2(I)=W2(I) + Z(I,K)*W1(K)
  DO 40 K=1,N
40  Z(K,L)=W2(K)

```

```

C
C   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
C
C   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 DCOM1(A,Z,N,KR,NERROR)
      DIMENSION A(KR,1),Z(KR,1)
      DATA EPS/1.E-35/
      DATA NIT,NOT/105,108/
C
C   DECOMPOSE MATRIX (A) TO FORM UPPER TRIANGULAR MATRIX (Z) SUCH THAT
C    $A = Z(\text{TRANS}) * Z$ . 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 WOHLER. OCTOBER 1970.
C   MODIFIED FOR CONTRACT NAS8-25922, MAY 1971.
C
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 A,Z IN CALLING PROGRAM.
C
      3001 FORMAT(5H1I = 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,I)=A(I,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,I)=SQRT(Z(I,I))
      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,I)
C
      40 DO 50 I=2,N
      IM1=I-1
      DO 50 J=1,IM1
50     Z(I,J)=0.0
      RETURN

```

C
998 WRITE(PNOT,3001) I
END

```

SUBROUTINE INV4(A,Z,N,KR,NERROR)
  DIMENSION A(KR,1),Z(KR,1)
  DATA EPS/1.E-35/

C
C   MATRIX INVERSION (A**1 = Z). MATRIX A IS ASSUMED TO BE
C   UPPER TRIANGULAR.
C   MATRICES A,Z, MAY SHARE SAME CORE LOCATIONS.
C   CODED BY RL KOHLEN, JANUARY 1971.
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.
C   KR = INPUT ROW DIMENSION OF A,Z IN CALLING PROGRAM.
C
      NERROR = 1
      DO 10 I=1,N
        IF (ABS(A(I,1)).LT.EPS) RETURN
10    Z(I,1)=1./A(I,1)
      NERROR=0
      IF (N.EQ.1) RETURN
C
      NM1=N-1
      DO 25 I=1,NM1
        IP1=I+1
        DO 25 J=IP1,N
          Z(I,J)=Z(I,I)*A(I,J)
          IF (J.EQ.IP1) GO TO 23
          JM1=J-1
          DO 20 K=IP1,JM1
20    Z(I,J)=Z(I,J) + Z(I,K)*A(K,J)
23    Z(I,J)=-Z(I,J)*Z(J,J)
25    CONTINUE
C
      DO 30 I=2,N
        IM1=I-1
        DO 30 J=1,IM1
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 DPMXVT (A,N,ERROR,TEMP)
DOUBLE PRECISION A,TEMP,AMAX,T
DIMENSION A(N,N),TEMP(1)
INTEGER OUT/108/
INTEGER IROW,ERROR
ERROR=0
L=N+1
N2=N+N
M=N+1
DO 16 I=1,N
K=N+I
16 TEMP(K)=I
DO 9 IJK=1,N
M=M+1
AMAX=ABS(A(1,1))
IROW=1
IF(M.EQ.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=I
2 CONTINUE
3 IF(AMAX) +,1+,4
4 T=1.000/A(IROW,1)
DO 5 I=2,N
5 TEMP(I-1)=A(IROW,I)*T
DO 6 J=1,N
6 A(IROW,J)=A(1,J)
DO 7 I=2,N
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
J=N+I
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.1) GO TO 13
KORD=K+N
TEMP(IROW)=TEMP(KORD)
TEMP(KORD)=K

```

```

      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
      GO TO 1000
100  FORMAT (//20X,42H MATRIX IS SINGULAR, NO INVERSE OBTAINABLE//)
      END

```

DPINV

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:

```
DOUBLE PRECISION A(n,n),B(n),C(n),PIVOT,Z
```

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

```

C      SUBROUTINE DPINV(A,ORDER,SING)
C      THE ARGUMENTS ARE AS FOLLOWS
C
C      A          IS THE MATRIX TO BE INVERTED. A IS DOUBLE
C                  PRECISION. THE INVERSE OF A IS RETURNED IN
C                  A.
C
C      ORDER      IS AN INTEGER VARIABLE. THE ORDER OF THE
C                  MATRIX TO BE INVERTED.
C
C      SING       IS A LOGICAL VARIABLE. WHEN THE ROUTINE
C                  RETURNS SING IS .FALSE. IF THE MATRIX WAS
C                  NON-SINGULAR, AND .TRUE. IF THE MATRIX WAS
C                  SINGULAR. IF SING IS .TRUE. THE MATRIX A
C                  CONTAINS GARBAGE.

```

```

C      INTEGER ORDER
C      DOUBLE PRECISION A(100,100),B(100),C(100),PIVOT,Z
C      N=ORDER
C      LOGICAL SING
C      INTEGER IP(100),IQ(100)
C      DO 1 K=1,N
C      DETERMINATION OF PIVOT ELEMENT
C      PIVOT=C.
C      DO 100 I=K,N
C      DO 100 J=K,N
C      IF(DABS(A(I,J))-DABS(PIVOT)) 100,100,101
C 101  PIVOT=A(I,J)
C      IP(K)=I
C      IQ(K)=J
C 100  CONTINUE
C      IF(PIVOT)102,900,102
C 102  IPK=IP(K)
C      IQK=IQ(K)
C      EXCHANGE OF THE PIVOTAL ROW WITH THE KTH ROW
C      IF(IPK=K)200,299,200
C 200  DO 201 J=1,N
C      Z=A(IPK,J)
C      A(IPK,J)=A(K,J)
C 201  A(K,J)=Z
C 299  CONTINUE
C      EXCHANGE OF THE PIVOTAL COLUMN WITH THE KTH COLUMN
C      IF(IQK=K)300,399,300
C 300  DO 301 I=1,N
C      Z=A(I,IQK)
C      A(I,IQK)=A(I,K)
C 301  A(I,K)=Z
C 399  CONTINUE
C      JORDAN ELIMINATION STEP
C      DO 400 J=1,N

```

```

      IF(J=K)403,402,403
402  B(J)=1./PIVOT
      C(J)=1.
      GO TO 404
403  B(J)=A(K,J)/PIVOT
      C(J)=A(J,K)
404  A(K,J)=0.
400  A(J,K)=0.
      DO 405 I=1,N
      DO 405 J=1,N
405  A(I,J)=A(I,J)+C(I)*B(J)
1    CONTINUE
C    REORDERING THE MATRIX
      K=N
      DO 500 KDUM=1,N
      IPK=IP(K)
      IQK=IQ(K)
      IF(IPK=K)501,502,501
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)504,500,504
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

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

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 DPINVT(A,N,M,DETER)
C   MATRIX INVERSION AND SIMULTANEOUS EQUATIONS SOLVER
C   A=INPUT MATRIX FOR INVERISON OR AUGMENTED MATRIX FOR SIME. EGS.
C   N=ORDER OF COEFFICIENT MATRIX
C   M=0 FOR INVERISON 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 DETER=1.000
      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=0.000
      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)55,75,250
      55 IF (AMAX=DABS(A(I,J)))60,75,75
      60 IR=I
      65 IC=J
      70 AMAX=DABS(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)=0.000
      175 DO 180 I=1,NN
      176 A(L,I)=A(L,I)-A(IC,I)*AMAX
      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)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 N=N-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)
```

```

C      SUBROUTINE SYMVRT(A,N,ISING)
      SYMETRIC INVERSION SUBROUTINE
      DIMENSION A(100,100),B(100),T(100)
      IF(N.EQ.1) GO TO 100
      IX=N
      ISING=0
      DO 18 ID=1,IX
      T(ID)=1.
      IDM1=ID-1
      IDP1=ID+1
      IF(IDM1)10,14,10
10 DO 11 J=ID,IX
      DO 11 K=1,IDM1
11 A(ID,J)=A(ID,J)-A(K,ID)*A(K,J)*T(K)
14 IF(A(ID,ID))13,12,15
12 ISING=ID
      GO TO 30
13 T(ID)=-1.
      A(ID,ID) = ABS(A(ID,ID))
15 A(ID,ID)=SQRT (A(ID,ID))*T(ID)
      IF(IDP1=IX)16,16,19
16 DO 17 J=IDP1,IX
17 A(ID,J)=A(ID,J)/A(ID,ID)
      A(ID,ID)=A(ID,ID)*T(ID)
18 CONTINUE
19 A(IX,IX)=A(IX,IX)*T(IX)
      IXM1=IX-1
      DO 24 I=1,IXM1
      II=I+1
      DO 20 J=11,IX
20 A(I,J)=A(I,J)/A(I,I)
24 A(I,I)=T(I)/A(I,I)
      A(IX,IX)=T(IX)/A(IX,IX)
      DO 23 I=2,IX
      K=I+1
      IM1=I-1
      DO 23 L=1,IM1
      IF(K=IX)21,21,23
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
      B(I)=0.
      DO 29 K=1,IX
29 B(I)=B(I)+A(I,K)**2*T(K)
      M=I+1
      IF(M=IX)26,26,32
26 DO 27 J=M,IX
      A(J,I)=0.
      DO 27 K=J,IX

```

```

27 A(J,I)=A(J,I)+A(I,K)*A(J,K)*T(K)
32 DO 28 I=1,IX
28 A(I,1)=B(I)
   DO 35 J = 1,N
   DO 35 I = 1,N
35 A(J,I) = A(I,J)
   GO TO 101
101 CONTINUE
100 A(1,1) = 1./A(1,1)
30 RETURN
END

```

SOLEQ

Source: The Finite Element Method in Engineering Science,
by O. C. Zienkiewicz. Copyright 1971. McGraw-Hill
Publishing Company Limited. Used with permission
of McGraw-Hill Book Company.

Method: Gauss elimination (see Appendix A).

Usage:

```
DIMENSION SK(n,h), R1(n)
```

```
.
```

```
.
```

```
.
```

```
CALL SOLEQ (SK,R1,NSZF)
```

where

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

R1 = constant vector on entry

R1 = 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:

DIMENSION SK(n,h), R1(n)

If n is anything other than 50, the statement

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 SOLEG(SK,R1,NSZF)

SPECIFICATION STATEMENTS

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

REDUCE MATRIX

DO 300 N=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

N=NSZF
350 N=N-1
IF(N) 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

Source: International Journal for Numerical Methods in Engineering, "An Equation Solver of Very Large Capacity" by Gilles Cantin. Copyright (C) 1971 by John Wiley & Sons, Ltd. Reprinted by permission.

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 SYMSOL(A,B,C,NN,MM)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(NN,MM),B(NN),C(MM)

C
C      A      IS THE COEFFICIENT MATRIX
C      B      IS THE RIGHT HAND MEMBER
C      C      IS A WORKING SPACE NEEDED IN THE SUBROUTINE
C      NN     IS THE TOTAL NUMBER OF EQUATIONS
C      MM     IS THE HALF-BANDWIDTH OF THE SYSTEM
C
      N=0
100  N=N+1

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

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

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

C
C      4. REDUCE REMAINING EQUATIONS
C
      DO 260 L=2,MM
      I=N+L-1
      IF(NN-I) 260,240,240
240  J=0
      DO 250 K=L,MM
      J=J+1
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

C
C      BACK SUBSTITUTION
C
300  N=N-1

C
C      1. CHECK FOR FIRST EQUATION
C
      IF(N) 350,500,350

```

C 2. CALCULATE UNKNOWNNS B(N)

C

350 DO 400 K=2,MM

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

CATEGORY

MATHEMATICAL

PURPOSE

SOLVE BAND MATRIX SIMULTANEOUS EQUATIONS.

DESCRIPTION

THIS SUBROUTINE WILL SOLVE AN $N \times N$ 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 ROUND OFF 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 $N \times N + 1$ FULL BLOWN SYSTEM WOULD BE REDUCED BY THE ABOVE METHOD TO AN $N \times M + 1$ 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.

```

C      2  D      CONTAINS DETERMINANT OF A.
C      3  R      CONTAINS RANK OF A.
C      4  E      E=0., SOLUTION O.K. E=1., A SINGULAR.
C              E=2., SOLUTION ATTEMPTED, BUT A ILL CONDITIONED
C              OR SINGULAR. IN THIS CASE SOLUTIONS SHOULD BE
C              CHECKED TO ASSURE VALIDITY.
C
C  REMARKS
C      ONE OF THE AREAS IN WHICH THIS ROUTINE SHOULD BE VALUABLE
C      IS THAT OF SOLUTION OF ELLIPTIC BOUNDARY VALUE PROBLEMS.
C
C  REFERENCE
C      ANY GOOD TEXT ON NUMERICAL ANALYSIS
C      SOME TEXTS ON ELLIPTIC BOUNDARY VALUE PROBLEMS.
C
C  CONTACT
C      CHIEF, THEORETICAL PROBLEMS SECTION
C      SCIENTIFIC DIGITAL PROGRAMMING BRANCH
C      ARMY COMPUTATION CENTER
C
C
C      SUBROUTINE TRIMSS(A,N,NLD,NRD,NED,D,R,E)
C      DIMENSION A(3500,4)
C      DOUBLE PRECISION A,D,Y,W,S
C      DOUBLE PRECISION A,D,Y,W,S
C      X1 = 1.
C      L1 = 1
C      E=0.
C      R = 0.
C      D=1.
C      ND1=NED+1
C      M=NLD
C      NM1=N-1
C      DO 1 I=1,NM1
C      IF(I.GT.(N-NLD))M=M+1
C      NN=I+M-1
C      DO 2 II=1,NN
C      IF(ABS(A(I,1)).GE.ABS(A(II+1,1)))GO TO 2
C      D=-D
C      DO 3 J=1,ND1
C      Y=A(I,J)
C      A(I,J)=A(II+1,J)
C      3 A(II+1,J)=Y
C      2 CONTINUE
C      D=D+A(I,1)
C      IF(A(I,1).EQ.0.)GO TO 10
C      GO TO (5,13),L1
C      13 IF (ABS(ABS((X1=A(I,1))/X1)-1.) .LT. 1.E-7) E = 2.
C      X1 = A(I,1)
C      5 R = R + 1.
C      L1 = 2
C      DO 4 J=2,ND1
C      4 A(I,J)=A(I,J) / A(I,1)
C      K=I+1

```

```

NN=I+M
DO 1 II=K,NN
W=A(II,1)
DO 6 J=1,NED
6 A(II,J)=A(II,J+1)+A(I,J+1)*W
A(II,ND1)=A(II,NED)
1 A(II,NED)=0.
D=D+A(N,1)
IF(A(N,1).EQ.0.)GO TO 10
IF(ABS(ABS((X1=A(N,1))/X1)-1.)*LT.1.E-7)E=2.
5 R = R + 1.
A(N,1)=A(N,ND1)/A(N,1)
K=NM1
NN=2
6 IF(NN.GT.NED)NN=NED
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,ND1)+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:

```
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 traingular 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 DECOM(N,NB)
DOUBLE PRECISION A,Q
COMMON/A/ A(200,50),Q(200)
DOUBLE PRECISION SUM,C1,C2,C3
DO 40 I=1,N
KF=I+1
JMAX=I+NB-1
IF(JMAX .GT. N) JMAX=N
DO 40 J=1,JMAX
JJ=J-I+1
SUM=A(I,JJ)
KS=J-NB+1
IF(KS .LT. 1) KS=1
IF(KS .GT. KF) GO TO 15
DO 10 K=KS,KF
IK=I-K+1
JK=J-K+1
C1=A(K,IK)
C2=A(K,JK)
10 SUM=SUM+C1+C2
15 IF(I-J)30,20,30
20 CONTINUE
C3=DSQRT(SUM)
A(I,JJ)=C3
GO TO 40
30 A(I,JJ)=SUM/C3
40 CONTINUE
RETURN
END

```

```

SUBROUTINE SOLV(N,NB)
DOUBLE PRECISION A,G
DOUBLE PRECISION SUM,C1,C2,C3
COMMON/A/ A(200,50),G(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=G(K)
10 SUM=SUM-C1*C2
20 C3=A(I,1)
21 Q(I)=SUM/C3
I=N+1
DO 41 II=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
JI=J-1+1
C1=A(I,JI)
C2=G(J)
30 SUM=SUM-C1*C2
40 C3=A(I,1)
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 CHOLAS(A,N,MM,IB,NT)
```

```
THIS SUBROUTINE SOLVES A SYSTEM OF LINEAR EQUATIONS WHOSE COEFFIC-  
IENT 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
```

```
MM= HALF-BAND WIDTH + DIAGONAL ELEMENT
```

```
IB= NUMBER OF CONSTANT VECTORS = 1
```

```
NT= 1 IF DECOMPOSITION AND SUBSTITUTION DESIRED
```

```
GT 1 IF SUBSTITUTION ONLY
```

```
DIMENSION A(1)
```

```
MUD=MM-1
```

```
NS=MUD*MM/2
```

```
NN=N*MM-NS
```

```
IF(NT.GT.1) GO TO 31
```

```
30 DO 20 J=1,N
```

```
IF (J-MUD) 1,1,2
```

```
2 IN=J-MUD
```

```
L=IN+(J-MM)*MUD+NS
```

```
GO TO 7
```

```
1 IN=1
```

```
L=IN+(J-1)*J/2
```

```
7 IF (J-N+MUD) 103,103,105
```

```
105 MS=N
```

```
GO TO 104
```

```
103 MS=J+MUD
```

```
104 S1=0.
```

```
J1=J-1
```

```
J2=J+1
```

```
IF(J1) 4,4,3
```

```
3 DO 6 K=IN,J1
```

```
T1=A(L)
```

```
S1=S1+T1**2
```

```
6 L=L+1
```

```
4 T1=A(L)
```

```
T1=SQRT(T1-S1)
```

```
A(L)=T1
```

```
IF(J=N) 19,20,20
```

```
19 DO 18 I=J2,MS
```

```
SUM=0.
```

```
IF(I-MUD) 68,68,71
```

```
71 IN=I-MUD
```

```
LL=IN+(I-MM)*MUD+NS
```

```
GO TO 5
```

```
68 IN=1
```

```
LL=IN+(I-1)*I/2
```

```
5 IF(J1) 18,18,8
```

```

8 IF (IN=J1) 53,53,18
53 DO 17 K=1,N,J1
   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=IB+1
   NB=NM+1
   DO 65 K=1,NR
     A(NB)=A(NB)/A(1)
   DO 60 I=2,N
     IF (I=MOD) 21,21,22
22 IN=I-MM
   KS=IN*MOD+NS
   M5=MOD
   GO TO 27
21 IN=0
   M5=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)
65 NB=NB+N
   NM=NM+N
   DO 75 K=1,NR
     A(NB)=A(NB)/A(NM)
   DO 80 II=2,N
     I=N+II+1
     IF (I=MOD) 41,41,95
95 ID=I+(I-MM)*MOD+NS
   GO TO 42
41 ID=I+(I-1)*I/2
42 IF (I=N+MM) 43,43,45
45 M5=II-1
   GO TO 76
43 M5=MOD
76 SUM=0.
   DO 81 J=1,M5
     JR=I+J
     IF (JR=MOD) 98,98,99
99 L=I+(JR-MM)*MOD+NS
   GO TO 82
98 L=I+(JR-1)*JR/2
82 JR=NB=N+JR

```

```
81 SUM=SUM+A(L)*A(JR)
   JR=NB=N+1
80 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 = number of equations to be solved

NB = 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:

```
COMMON/A/ A(n,h), Q(n)
```

```

SUBROUTINE SOLBAN(NU,BW)
COMMON/A/ A(200,50),Q(200)
DOUBLE PRECISION A,Q,C1,SUM,SUB
INTEGER H, H1, BW ,0

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

```

```
SUM=SUM+A(I,JJ)*G(J)
14 CONTINUE
15 G(I)=G(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: DOUBLE PRECISION FORCT
 DIMENSION FORCT(n)
 INTEGER TA,TB
 .
 .
 .
 REWIND TB
 CALL DISPL(N,IHB,FORCT,TA,TB)

where

FORCT = solution vector

n = maximum permissible number of equations

TA = number assigned to a sequential tape or disk
file to be used as a scratch file

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 = number of equations to be solved

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

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

`XK(h,h),FORCT(n)`

where h is the maximum permissible half-band width.

If h is anything other than 50, the statements

`DO 80 I = 1,50`

`DO 80 J = 1,50`

must be modified accordingly.

SUBROUTINE DISPL(N,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,SUB,SUM
REWIND TA

DETERMINE THE NUMBER OF BLOCKS OF COEFFICIENTS REQUIRED IN CORE

ORDER=N
AIHB=IHB
AORDER=ORDER
BLOCK=AORDER/AIHB
NBLOCK=ORDER/IHB
ABLOCK=NBLOCK
IF(BLOCK*.5T.ABLOCK) NBLOCK=NBLOCK+1

FACTOR AND FORWARD SUBSTITUTION OF STIFFNESS MATRIX ASSEMBLED IN A HALF BAND SQUARE BLOCK AT A TIME.

DO 80 I=1,50
DO 80 J=1,50

```

80 XK(I,J)=0.
   M = IMB
   DO 5 NN=1,NBLOCK
     MM=M
     IF(NN.EQ.NBLOCK)MM=N-(NBLOCK-1)*IMB
     IF(NN.LE.2)GO TO 9
C
C   CHANGE OF XK SUBSCRIPTS FOR THE THIRD AND/OR LATER BLOCKS.
C
     DO 2 I=2,M
       DO 2 J=2,I
         2 XK(I,J)=XK(I,M-I+J)
         9 DO 4 I=1,MM
           IF(NN.NE.1) GO TO 101
           READ(TB)(XK(I,J),J=1,I),FORCT(I)
           GO TO 102
101 CONTINUE
C
C   READ FROM DISK (XK(I,J),J=1,M) AND FORCT(I)
C
     READ(TB)(XK(I,J),J=1,M),FORCT(I)
102 CONTINUE
     DO 6 J=1,M
       SUB=0.
       SUM=0.
       IF(NN.EQ.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
C
C   OFF-DIAGONAL FACTOR FOR THE FIRST BLOCK
C
       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,K)
         21 XK(I,J) = (XK(I,J)-SUM)/XK(J,J)
       GO TO 6
C
C   DIAGONAL FACTOR FOR THE FIRST BLOCK.
C
       20 I1=I-1
       IF(I1.EQ.0)GO TO 23
       DO 24 K=1,I1
         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) - SUM
         FORCT(I) = FORCT(I)-SUB
       GO TO 6
C

```

C OFF-DIAGONAL FACTOR FOR BLOCKS OTHER THAN THE FIRST.

C

```
8 IF(J.EQ.M) GO TO 10
  II=I+J
  IU=I+J
  IF(IU-M.GT.0) IU=I+J-M
  IF(II-M.GT.0) II=M
  J11=J-1
  IF(J11.EQ.0)GO TO 12
  DO 11 K=1,J11
    IK=I+K
    KI=IK
    MUK=M-J+K
    IF(MUK.GT.I+1.AND.I+J.LE.M) MUK=I+K
    IF(I+K-M.GT.0)IK=M
    IF(I+K-M.GT.0)KI=I+K-M
11 SUM = SUM + XK(I,K)*XK(IU,MUK)*XK(KI,IK)
12 XK(I,J) = (XK(I,J)-SUM)/XK(IU,II)
  GO TO 6
```

C

C DIAGONAL FACTOR FOR BLOCKS OTHER THAN THE FIRST.

C

```
10 M1=M-1
  DO 13 K=1,M1
    IK=I+K
    KI=IK
    IF(IK.GT.M) IK=I+K-M
    IF(KI.GT.M) KI=M
    SUM = SUM + XK(I,K)*XK(I,K)*XK(IK,KI)
13 SUB = SUB + XK(I,K)*FORCT(IK)
  XK(I,J) = XK(I,J) - SUM
  FORCT(I) = FORCT(I) - SUB
6 CONTINUE
  WRITE(TA)(XK(I,J),J=1,M),FORCT(I)
4 CONTINUE
5 CONTINUE
```

C

C

BACK SUBSTITUTION STARTS

C

```
  REWIND TB
  BACKSPACE TA
  MM = MM
  DO 52 NN=1,NBLOCK
    IF(NN.GT.1)MM=M
    DO 52 I=1,MM
30 MI=MM-I+1
    SUM=0
    READ(TA)(XK(MI,J),J=1,M),FORCT(MI)
    IF(NN.EQ.NBLOCK.AND.I.EQ.M) GO TO 39
```

C

```

C   TO READ FROM THE BOTTOM UP.
C
      BACKSPACE TA
      BACKSPACE TA
39  CONTINUE
      IF(NN.LT.NBLOCK)GO TO 33
C
C   FIRST BLOCK (TRIANGULAR) REARRANGED.
C
      I1=I-1
      IF(I1.EQ.0) GO TO 33
      DO 34 J=1,MI
        JM1=M-J+1-I1
34  XK(MI,K=J+1) = XK(MI,JM1)
33  IF(NN.GT.1) GO TO 36
C
C   DISPLACEMENTS FOR THE LAST BLOCK.
C
      I1=I-1
      IF(I1.EQ.0)GO TO 50
      DO 51 K=1,I1
51  SUM = SUM + XK(MI+K,M-K)*FORCT(MI+K)
50  FORCT(MI) = FORCT(MI)/XK(MI,I1) = SUM
      GO TO 52
C
C   DISPLACEMENTS FOR BLOCKS OTHER THAN THE LAST.
C
36  I1 = NNN + I - 1
      IF(I1.GT.M-1.OR.NN.GT.2) I1=M-1
31  DO 55 K=1,I1
      MK=MI+K
      IF(MK.GT.M)MK=MK-M
55  SUM = SUM + XK(MK,M-K)*FORCT(MK)
      FORCT(MI)=(FORCT(MI)/XK(MI,I1))-SUM
52  WRITE(TB)FORCT(MI)
C
C   NOW THE DISPLACEMENTS ARE STORED ON DISK TB IN REVERSE
C   ORDER WHICH WILL BE READ AND STORED IN CORE WITH THE
C   ORDER PROPERLY RESTORED.
C
      REWIND TB
      DO 56 I=1,N
        J=N-I+1
56  READ(TB)FORCT(J)
      RETURN
      END

```

BANSOL

Source: Analysis of Axisymmetric Solids, by E. L. Wilson,
Computer Programming Series, University of
California, 1967.

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 = the product of NUMBLK and NN (or larger)

m = twice the block depth NN (or larger)

p = the larger of MM and NUMBLK

IT = number assigned to a sequential tape or disk
file to be used as a scratch file

JT = number assigned to the sequential tape or
disk file in which the upper half-band and
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
```

```

C
C      SUBPROGRAM SUBROUTINE BANSOL
C
      SUBROUTINE BANSOL
      COMMON /BANARG/ MM,NUMBLK,B(200),A(4,100),NN,IT,UT
      DOUBLE PRECISION A,B,C
      NL=NN+1
      NH=NN+NN
      REWIND IT
      REWIND UT
      NB=0
      GO TO 150
C
C      REDUCE EQUATIONS BY BLOCKS
C      SHIFT BLOCK OF EQUATIONS
C
100  NB=NB+1
      DO 125 N=1,NN
      NH=NN+N
      B(N)=B(NH)
      B(NH)=0.0
      DO 125 M=1,MM
      A(N,M)=A(NH,M)
125  A(NH,M)=0.0
C
C      READ NEXT BLOCK OF EQUATIONS INTO CORE
C
      IF (NUMBLK-NB) 150,200,150
150  READ (UT)(B(N),(A(N,M),M=1,MM),N=NL,NH)
      IF (NB) 200,100,200
C
C      REDUCE BLOCK OF EQUATIONS
C
200  DO 300 N=1,NN
      IF (A(N,1)) 225,300,225
225  B(N)=B(N)/A(N,1)
      DO 275 L=2,MM
      IF (A(N,L)) 230,275,230
230  C=A(N,L)/A(N,1)
      I=N+L-1
      J=0
      DO 250 K=L,MM
      J=J+1
250  A(I,J)=A(I,J)-C*A(N,K)
      B(I)=B(I)-A(N,L)*B(N)
      A(N,L)=C
275  CONTINUE
300  CONTINUE
C
C      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,MM),N=1,NN)
      GO TO 100

C
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
C      ORDER UNKNOWNNS 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)

Subprogram Name	Execution Time, sec.	Error Measure	Maximum Error	Comments
SIMEQ	1	0.00968	0.137	
SIMEQ3	14	0.00968	0.137	
WUGEL	16	0.00968	0.137	
GAUSEL	11	0.00968	0.137	
SIMEQ2	17	0.00968	0.137	
FACTOR, LU	14	0.00968	0.137	
GAUSS	12	0.00968	0.137	
SOLVIT	3	0.00970	0.137	No iterative improvement, ACC = 0.001
	3	0.0000170	0.000229	1 iteration
	5	0.0	0.0	2 iterations
SMEQ1	20	0.00968	0.137	
DPSIM	19	0.255×10^{-11}	0.356×10^{-10}	Double precision
SHORT	19	0.00968	0.137	
SOLVE1	31	0.00969	0.137	
WUIGJ	48	0.00968	0.137	
WUGJN	25	0.00968	0.137	
SLVE	34	0.00968	0.137	
GASSIM	31	0.00968	0.137	
GJR	42	0.00968	0.137	
SIMULT	1	0.159×10^{-13}	0.174×10^{-12}	Double precision
SOLV 3	1	0.00968	0.137	
SOLV 2	25	0.00588	0.00583	2199 Gauss-Seidel iterations
GSEID	103	5.51	84.1	200 Gauss-Seidel iterations

Table 2.

In-core, Non-banded Matrix Inversion Subroutines
(Single Precision Except as Noted)

Subprogram Name	Execution Time, sec.	Error Measure	Maximum Error	Comments
AINVER	57	0.00968	0.137	
BINVER	103	0.00968	0.137	
MATIN1	69	0.00968	0.137	
MATIN4	45	0.00968	0.137	
MIV	25	0.00968	0.137	
INVT	197	0.0000549	0.000900	
INVERT	85	---	---	50 equations
	177	---	---	70 equations
	300+	---	---	100 equations, no solution
INVTIT	63	0.00970	0.137	No iterative improvement
	117	0.0000170	0.000229	1 iteration
	262	0.0	0.0	2 iterations
MATIN3	38	0.00968	0.137	
INV1	110	0.00968	0.137	
INV1A	48	---	---	50 equations
	154	---	---	70 equations
	300+	---	---	100 equations, no solution
INV2	121	0.000334	0.00455	
INV3	71	0.0176	0.254	
DPMNVT	58	0.255×10^{-11}	0.356×10^{-10}	Double precision
DPINV	47	0.687×10^{-13}	0.101×10^{-11}	Double precision
DPINVT	79	0.270×10^{-13}	0.423×10^{-12}	Double precision
SYMVRT	24	0.0177	0.254	

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 3.

In-core, Banded Equation Solvers

Subprogram Name	Precision	Number of Equations	Half-band width	Execution Time, sec	Error Measure	Maximum Error	Comments
SOLEQ	single	200	50	1	0.0549	1.11	Full band required
SYMSOL	double	200	50	13	0.144×10^{-10}	0.289×10^{-9}	
TRIMSS	double	3500	2	2	0.183×10^{-7}	0.155×10^{-5}	
BANSLV	single	200	50	9	0.0549	1.11	
	double	3500	2	1	0.183×10^{-7}	0.155×10^{-5}	
DECOM, SOLV	double	200	50	14	$.127 \times 10^{-10}$	0.253×10^{-9}	
CHOLCS	single	200	50	14	0.0713	1.39	
SOLBAN	double	200	50	24	0.177×10^{-10}	0.360×10^{-9}	

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

Table 4
Out-of-core Equation Solvers

Subprogram Name	Execution Time, sec	Error Measure	Maximum Error	Block Size
DISPL	95	0.177×10^{-10}	0.360×10^{-9}	2
BANSOL	29	0.177×10^{-10}	0.360×10^{-9}	2

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.

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 CHOLDS 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 CHOLDS 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.

APPENDIX A

METHODS OF SOLUTION

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

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\
 &\vdots \\
 a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n
 \end{aligned} \tag{A.1}$$

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

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{A.2}$$

or

$$AX = B \tag{A.3}$$

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

$$X = A^{-1}B \quad (A.4)$$

where A^{-1} is the inverse of 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

$$\begin{array}{ccccccc} a'_{11}x_1 & + & a'_{12}x_2 & + & \cdots & + & a'_{1n}x_n & = & b'_1 \\ 0 & & a'_{22}x_2 & + & \cdots & + & a'_{2n}x_n & = & b'_2 \\ \vdots & & & & & & & & \vdots \\ 0 & + & 0 & + & \cdots & + & a'_{nn}x_n & = & b'_n \end{array}$$

The matrix representation is

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & & & \\ 0 & 0 & \cdots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{bmatrix} \quad (A.5)$$

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

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{k=i+1}^n a'_{ik} x_k \right) \quad (\text{A.6})$$

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,

$$x_1 = \frac{b_1 - (a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n)}{a_{11}} \quad (\text{A.7})$$

The second equation becomes

$$a_{21} \left[\frac{b_1 - (a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n)}{a_{11}} \right]$$

$$+ a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2$$

or

$$\left(a_{22} - \frac{a_{21}}{a_{11}} a_{12} \right) x_2 + \left(a_{23} - \frac{a_{21}}{a_{11}} a_{13} \right) x_3 \\ + \dots + \left(a_{2n} - \frac{a_{21}}{a_{11}} a_{1n} \right) x_n = b_2 - \frac{a_{21}}{a_{11}} b_1$$

or

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2$$

In other words, subtracting a_{21}/a_{11} 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 a_{i1}/a_{11} 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

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & a'_{32} & a'_{33} & \dots & a'_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a'_{n2} & a'_{n3} & \dots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}$$

where

$$a'_{ij} = a_{ij} - \frac{a_{i1} a_{1j}}{a_{11}}$$

and

$$b'_i = b_i - \frac{a_{i1} b_1}{a_{11}}$$

The next step is to eliminate x_2 from each equation after the second to obtain the equivalent system

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a''_{33} & \dots & a''_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a''_{n3} & \dots & a''_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b''_3 \\ \vdots \\ b''_n \end{bmatrix}$$

where

$$a''_{ij} = a'_{ij} - \frac{a'_{i2}}{a'_{22}} a'_{2j}$$

and

$$b''_i = b'_i - \frac{a'_{i2}}{a'_{22}} b'_2$$

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 k th stage of elimination the k th unknown is eliminated from all equations that follow the k th. There will be a total of $n-1$ stages, since the n th 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

$$a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj} \quad (\text{A.8})$$

and

$$b_i = b_i - \frac{a_{ik}}{a_{kk}} b_k \quad (\text{A.9})$$

These relationships give the new values of the coefficients of each of the equations following the k th equation, as obtained during the k th stage of elimination. In practice, division of the k th 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

$$\begin{bmatrix} 1 & a'_{12} & a'_{13} & \dots & a'_{1n} \\ 0 & 1 & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & 1 & \dots & a'_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}$$

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

$$a_{kj} = a_{kj} / a_{kk} \quad \text{for } j = k, \dots, n$$

$$a_{ij} = a_{ij} - a_{ik} a_{kj} \quad \text{for } j = k+1, \dots, n$$

$$b_i = b_i - a_{ik} b_k$$

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 indeterminate situation will exist. The difficulty can be circumvented by searching the

equations following the k th for one with a nonzero coefficient in the k th column of A . This equation is then interchanged with the k th 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 k th 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

$$\begin{bmatrix} 1 & 0 & a'_{13} & \dots & a'_{1n} \\ 0 & 1 & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a'_{33} & \dots & a'_{3n} \\ \vdots & & & & \\ 0 & 0 & a'_{n3} & \dots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}$$

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

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

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

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:

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad (\text{A.10})$$

If A is decomposed into lower and upper triangular matrices L and U, the system $AX = B$ may be written as $LUX = B$ or

$$LC = B \quad (\text{A.11})$$

where

$$UX = C \quad (\text{A.12})$$

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

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

An examination of the system (A.10) shows that there are a total of $n^2 + n$ undetermined l 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

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

Expanding the above, we obtain the following equations:

$$\begin{aligned} l_{11} \cdot 1 &= a_{11} \\ l_{11} u_{12} &= a_{12} \\ &\vdots \\ l_{11} u_{1n} &= a_{1n} \\ l_{21} \cdot 1 &= a_{21} \end{aligned}$$

$$l_{21} u_{12} + l_{22} \cdot 1 = a_{22}$$

.

.

.

$$l_{21} u_{1n} + l_{22} u_{2n} = a_{2n}$$

.

.

.

$$l_{n1} \cdot 1 = a_{n1}$$

$$l_{n1} u_{12} + l_{n2} \cdot 1 = a_{n2}$$

.

.

.

$$l_{n1} u_{1n} + l_{n2} u_{2n} + \dots + l_{nn} \cdot 1 = a_{nn}$$

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

$$u_{kk} = 1$$

$$l_{ik} = a_{ik} - \sum_{m=1}^{k-1} l_{im} u_{mk} \quad \text{for } i = k, \dots, n$$

$$u_{kj} = \frac{1}{l_{kk}} \left(a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj} \right) \quad \text{for } j = k+1, \dots, n$$

$$l_{ij} = 0 \quad \text{for } i < k$$

$$u_{kj} = 0 \quad \text{for } j < k$$

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

$$l_{kk} = 1 \quad \text{for } k = 1, \dots, n$$

$$u_{kj} = a_{kj} - \sum_{m=1}^{k-1} l_{km} u_{mj} \quad \text{for } j = k, \dots, n$$

$$l_{kj} = \frac{1}{u_{jj}} \left(a_{kj} - \sum_{m=1}^{j-1} l_{km} u_{mj} \right) \quad \text{for } j = 1, \dots, k-1$$

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 n th 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 3×3 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

$$u_{11} g_{11} + u_{12} g_{21} + u_{13} g_{31} = 1$$

$$u_{11} g_{12} + u_{12} g_{22} + u_{13} g_{32} = 0$$

$$u_{11} g_{13} + u_{12} g_{23} + u_{13} g_{33} = 0$$

$$u_{22} g_{21} + u_{23} g_{31} = 0$$

$$u_{22} g_{22} + u_{23} g_{32} = 1$$

$$u_{22} g_{23} + u_{23} g_{33} = 0$$

$$u_{33} g_{31} = 0$$

$$u_{33} g_{32} = 0$$

$$u_{33} g_{33} = 1$$

It will be observed that the inverse of an upper triangular matrix is also an upper triangular matrix. The pattern that emerges is

$$g_{ii} = \frac{1}{u_{ii}} \quad \text{for } i = n, \dots, 1 \quad (\text{A.13})$$

$$g_{ij} = \frac{1}{u_{ii}} \left(- \sum_{k=i+1}^j u_{ik} g_{kj} \right) \quad (\text{A.14})$$

for $i = n-1, \dots, 1$ and $j = n, \dots, i+1$.

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

$$g_{ii} = \frac{1}{l_{ii}} \quad \text{for } i = 1, \dots, n$$

$$g_{ij} = \frac{1}{l_{ii}} \left(- \sum_{k=j}^{i-1} l_{ik} g_{kj} \right)$$

for $j = 1, \dots, i-1$ and $i = 2, \dots, n$.

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:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} d_{11} & 0 & \dots & 0 \\ 0 & d_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_{nn} \end{bmatrix} \begin{bmatrix} 1 & l_{21} & \dots & l_{n1} \\ 0 & 1 & \dots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \\
 = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Incorporating D into L we have

$$\begin{bmatrix} d_{11} & 0 & \dots & 0 \\ l_{21}d_{11} & d_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1}d_{11} & l_{n2}d_{22} & \dots & d_{nn} \end{bmatrix} \begin{bmatrix} 1 & l_{21} & \dots & l_{n1} \\ 0 & 1 & \dots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Expanding,

$$d_{11} = a_{11}$$

$$d_{11}l_{21} = a_{12}$$

$$\vdots$$

$$d_{11}l_{n1} = a_{1n}$$

$$l_{21}d_{11} = a_{21}$$

$$l_{21}d_{11}l_{21} + d_{22} = a_{22}$$

$$\vdots$$

$$l_{21}d_{11}l_{n1} + d_{22}l_{n2} = a_{2n}$$

$$\vdots$$

$$l_{n1}d_{11} = a_{n1}$$

$$l_{n1}d_{11}l_{21} + l_{n2}d_{22} = a_{n2}$$

$$\vdots$$

$$l_{n1}d_{11}l_{n1} + l_{n2}d_{22}l_{n2} + \dots + d_{nn} = a_{nn}$$

by accounting for the symmetry of A, we can compute the elements of D and L from

$$d_{ii} = a_{ii} - \sum_{r=1}^{i-1} l_{ir}^2 d_{rr}$$

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

and

$$l_{ij} = 0 \quad \text{for } i < j$$

To obtain the solution vector X, we first write the system $AX = B$ in the form $LDL^T X = B$ which becomes

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

where

$$DL^T X = C \quad (\text{A.16})$$

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

$$c_i = b_i - \sum_{k=1}^{i-1} l_{ik} c_k$$

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

$$\begin{aligned} x_i &= \frac{1}{g_{ii}} \left(c_i - \sum_{k=i+1}^n g_{ik} x_k \right) \\ &= \frac{1}{d_{ii}} \left(c_i - \sum_{k=i+1}^n d_{ii} l_{ki} x_k \right) \end{aligned}$$

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 non-singular. 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 $A = L^{-1}U$. 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 $A = LL^T$ or U^TU . If we use the latter form, the decomposition is represented by

$$\begin{bmatrix} u_{11} & 0 & \cdots & 0 \\ u_{12} & u_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ u_{1n} & u_{2n} & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix}$$

Expanding,

$$\begin{aligned} u_{11}^2 &= a_{11} \\ u_{11}u_{12} &= a_{12} \\ &\vdots \end{aligned}$$

$$u_{11} u_{1n} = a_{1n}$$

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

$$u_{12}^2 + u_{22}^2 = a_{22}$$

⋮

⋮

$$u_{12} u_{1n} + u_{22} u_{2n} = a_{2n}$$

⋮

⋮

$$u_{1n} u_{11} = a_{1n}$$

$$u_{1n} u_{12} + u_{2n} u_{22} = a_{2n}$$

⋮

⋮

$$u_{1n}^2 + u_{2n}^2 + \cdots + u_{nn}^2 = a_{nn}$$

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

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2} \quad \text{for } i = 1, \dots, n$$

$$u_{ij} = \frac{1}{u_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj} \right) \quad \begin{array}{l} \text{for } i = 1, \dots, n \\ \text{and } j = i+1, \dots, n \end{array}$$

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 $AX = U^T UX = B$ 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

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

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

```

1.      COMMON/ A/ A(200,50),Y(200)
2.      DOUBLE PRECISION A,Y
3.      NC=200
4.      IHBK=50
5.      C
6.      C   GENERATE THE UPPER HALF-BAND (INCLUDING THE DIAGONAL TERM)
7.      C   MATRIX AND THE RIGHT HAND SIDE VECTOR.
8.      C
9.      DO 5 I=1,NC
10.     Y(I)=0.
11.     A(I,1)=2.
12.     5 A(I,2)=1.
13.     A(NC,1)=1.
14.     A(NC,2)=0.
15.     Y(NC)=1.
16.     DO 10 I=1,NC
17.     DO 10 J=3,IHBK
18.     10 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.     CALL SOLBAN(NC,IHBK)
27.     C
28.     C   COMPUTE EXECUTION TIME
29.     C
30.     CALL TESTTIME(NT1)
31.     TIME=FLOAT(NT1)/1000.
32.     WRITE(108,100) (I,Y(I),I=1,NC)
33.     100 FORMAT('1',//,35X,'SOLUTIONS',//,19X,'1',20X,'Y(1)',
34.     *          //,(I20,E27.8))
35.     WRITE(108,200) TIME
36.     200 FORMAT(///,' EXECUTION TIME =',F7.3,' SECONDS')
37.     C
38.     C   COMPUTE ERROR MEASURES
39.     C
40.     SUM=0.
41.     AMAX=0.
42.     DO 20 I=1,NC
43.     R=FLOAT(I)-Y(I)
44.     IF( ABS(R).GT.AMAX) AMAX= ABS(R)
45.     20 SUM=SUM+R**2
46.     ERROR= SQRT(SUM)/ FLOAT(NC)
47.     WRITE(108,300) ERROR
48.     300 FORMAT(///,' ERROR MEASURE =',E15.6)
49.     WRITE(108,400) AMAX
50.     400 FORMAT(///,' MAXIMUM ERROR =',E15.6)
51.     STOP
52.     END

```

```

1.      SUBROUTINE SOLBAN(NU,BW)
2.      COMMON/A/ A(200,50),G(200)
3.      DOUBLE PRECISION A,G,C1,SUM,SUB
4.      INTEGER M, H1, BW, IC
5.      C
6.      DO 4 I=1,NU
7.      DO 6 J=1,BW
8.      SUB=0.
9.      SUM=0.
10.     H1=I-1
11.     H=BW-1
12.     IF (H1.LE.H) H=H1
13.     IF (J.EQ.1) GO TO 10
14.     IF (H.LE.0) GO TO 17
15.     C COMPUTE OFF DIAGONAL TERMS
16.     DO 11 M=1,H
17.     K=I-M
18.     N=J+M
19.     C=M+1
20.     IF (C.GT.BW) GO TO 11
21.     IF (N.GT.BW) GO TO 11
22.     SUM=SUM+A(K,0)+A(K,N)*A(K,1)
23.     11 CONTINUE
24.     17 A(I,J)=(A(I,J)-SUM)/A(I,1)
25.     GO TO 6
26.     C COMPUTE DIAGONAL TERMS
27.     10 CONTINUE
28.     IF (H.LE.0) GO TO 16
29.     DO 13 M=1,H
30.     K=I-M
31.     N=J+M
32.     C1=A(K,N)
33.     SUM=SUM+C1+C1*A(K,1)
34.     C=I-K+1
35.     SUB=SUB2+G(K)*A(K,0)
36.     13 CONTINUE
37.     16 A(I,1)=A(I,1)-SUM
38.     G(I)=G(I)-SUB
39.     6 CONTINUE
40.     4 CONTINUE
41.     DO 12 III=1,NU
42.     I=NU-III+1
43.     SUM=0.
44.     H1=III-1
45.     H=BW-1
46.     IF (H1.LE.H) H=H1
47.     IF (H.LI.1) GO TO 15
48.     DO 14 K=1,H
49.     J=I+K
50.     JJ=J-I+1
51.     SUM=SUM+A(I,JJ)*G(JJ)
52.     14 CONTINUE
53.     15 G(I)=G(I)/A(I,1)-SUM
54.     12 CONTINUE
55.     RETURN
56.     END

```

SOLUTIONS

I	Y(I)
1	.10000000E 01
2	.20000000E 01
3	.30000000E 01
4	.40000000E 01
5	.50000000E 01
6	.60000000E 01
7	.70000000E 01
8	.80000000E 01
9	.90000000E 01
10	.10000000E 02
11	.11000000E 02
12	.12000000E 02
13	.13000000E 02
14	.14000000E 02
15	.15000000E 02
16	.16000000E 02
17	.17000000E 02
18	.18000000E 02
19	.19000000E 02
20	.20000000E 02
21	.21000000E 02
22	.22000000E 02
23	.23000000E 02
24	.24000000E 02
25	.25000000E 02
26	.26000000E 02
27	.27000000E 02
28	.28000000E 02
29	.29000000E 02
30	.30000000E 02
31	.31000000E 02
32	.32000000E 02
33	.33000000E 02
34	.34000000E 02
35	.35000000E 02
36	.36000000E 02
37	.37000000E 02
38	.38000000E 02
39	.39000000E 02
40	.40000000E 02
41	.41000000E 02
42	.42000000E 02
43	.43000000E 02
44	.44000000E 02
45	.45000000E 02
46	.46000000E 02
47	.47000000E 02
48	.48000000E 02
49	.49000000E 02
50	.50000000E 02
51	.51000000E 02
52	.52000000E 02
53	.53000000E 02
54	.54000000E 02
55	.55000000E 02

56	.56000000E 02
57	.57000000E 02
58	.58000000E 02
59	.59000000E 02
60	.60000000E 02
61	.61000000E 02
62	.62000000E 02
63	.63000000E 02
64	.64000000E 02
65	.65000000E 02
66	.66000000E 02
67	.67000000E 02
68	.68000000E 02
69	.69000000E 02
70	.70000000E 02
71	.71000000E 02
72	.72000000E 02
73	.73000000E 02
74	.74000000E 02
75	.75000000E 02
76	.76000000E 02
77	.77000000E 02
78	.78000000E 02
79	.79000000E 02
80	.80000000E 02
81	.81000000E 02
82	.82000000E 02
83	.83000000E 02
84	.84000000E 02
85	.85000000E 02
86	.86000000E 02
87	.87000000E 02
88	.88000000E 02
89	.89000000E 02
90	.90000000E 02
91	.91000000E 02
92	.92000000E 02
93	.93000000E 02
94	.94000000E 02
95	.95000000E 02
96	.96000000E 02
97	.97000000E 02
98	.98000000E 02
99	.99000000E 02
100	.10000000E 03
101	.10100000E 03
102	.10200000E 03
103	.10300000E 03
104	.10400000E 03
105	.10500000E 03
106	.10600000E 03
107	.10700000E 03
108	.10800000E 03
109	.10900000E 03
110	.11000000E 03
111	.11100000E 03
112	.11200000E 03
113	.11300000E 03
114	.11400000E 03

116	.11600000E 03
117	.11700000E 03
118	.11800000E 03
119	.11900000E 03
120	.12000000E 03
121	.12100000E 03
122	.12200000E 03
123	.12300000E 03
124	.12400000E 03
125	.12500000E 03
126	.12600000E 03
127	.12700000E 03
128	.12800000E 03
129	.12900000E 03
130	.13000000E 03
131	.13100000E 03
132	.13200000E 03
133	.13300000E 03
134	.13400000E 03
135	.13500000E 03
136	.13600000E 03
137	.13700000E 03
138	.13800000E 03
139	.13900000E 03
140	.14000000E 03
141	.14100000E 03
142	.14200000E 03
143	.14300000E 03
144	.14400000E 03
145	.14500000E 03
146	.14600000E 03
147	.14700000E 03
148	.14800000E 03
149	.14900000E 03
150	.15000000E 03
151	.15100000E 03
152	.15200000E 03
153	.15300000E 03
154	.15400000E 03
155	.15500000E 03
156	.15600000E 03
157	.15700000E 03
158	.15800000E 03
159	.15900000E 03
160	.16000000E 03
161	.16100000E 03
162	.16200000E 03
163	.16300000E 03
164	.16400000E 03
165	.16500000E 03
166	.16600000E 03
167	.16700000E 03
168	.16800000E 03
169	.16900000E 03
170	.17000000E 03
171	.17100000E 03
172	.17200000E 03
173	.17300000E 03

174	.17400000E 03
175	.17500000E 03
176	.17600000E 03
177	.17700000E 03
178	.17800000E 03
179	.17900000E 03
180	.18000000E 03
181	.18100000E 03
182	.18200000E 03
183	.18300000E 03
184	.18400000E 03
185	.18500000E 03
186	.18600000E 03
187	.18700000E 03
188	.18800000E 03
189	.18900000E 03
190	.19000000E 03
191	.19100000E 03
192	.19200000E 03
193	.19300000E 03
194	.19400000E 03
195	.19500000E 03
196	.19600000E 03
197	.19700000E 03
198	.19800000E 03
199	.19900000E 03
200	.20000000E 03

EXECUTION TIME = 20.266 SECONDS

ERROR MEASURE = .177302E+10

MAXIMUM ERROR = .359890E-09
STOP 0

REFERENCES

1. Gregory, Robert T. and Karney, David L.: A Collection of Matrices for Testing Computational Algorithms. Wiley-Interscience, 1969.
2. Westlake, J.: A Handbook of Numerical Matrix Inversion and Solution of Linear Equations. John Wiley and Sons, 1968.
3. Newman, Morris and Todd, John: The Evaluation of Matrix Inversion Programs. Journal of SIAM, Vol. 6, No. 4, December, 1958.
4. Desai, Chandrakant S. and Abel, John F.: Introduction to the Finite Element Method. Van Nostrand Reinhold Company, 1972.
5. Gastinel, Noel: Linear Numerical Analysis. Academic Press, Inc., New York, 1970.
6. Hall, A. S. and Woodhead, R. W.: Frame Analysis. John Wiley and Sons, 1961.
7. Faddeeva, V. N.: Computational Methods of Linear Algebra. Dover Publications, Inc., New York, 1959.
8. Helms, Ronald: An Experimental Comparison of Certain Matrix Inversion and Simultaneous Linear Equations Algorithms. Technical Report No. 6, University Computing Center, University of Tennessee, 1966.
9. Fox, L.: An Introduction to Numerical Linear Algebra. Oxford University Press, 1965.
10. Wendroff, B.: First Principles of Numerical Analysis: An Undergraduate Text. Addison-Wesley, 1969.

11. Segui, W. T.: Finite Element Analysis of Structures with Surfaces of Arbitrary Curvature. Ph.D. Dissertation, University of South Carolina, Columbia, South Carolina, 1971.
12. Arden, Bruce W. And Astill, Kenneth N.: Numerical Algorithms: Origins and Applications. Addison-Wesley, 1970.
13. Gerald, Curtis F.: Applied Numerical Analysis. Addison-Wesley, 1970.
14. James, M. L.; Smith, G. M. and Welford, J.C.: Applied Numerical Methods for Digital Computation with FORTRAN. International Textbook Company, 1968.
15. McCormick, John M. and Salvadori, Mario G.: Numerical Methods in FORTRAN. Prentice-Hall, Inc., 1965.
16. Ralston, Anthony: A First Course in Numerical Analysis. McGraw-Hill, 1965.
17. Wang, P.C.: Numerical and Matrix Methods in Structural Mechanics. John Wiley and Sons, 1966.
18. Watson, W. A.; Philipson, T. and Oates, P. J.: Numerical Analysis - The Mathematics of Computing, Vol. 1. American Elsevier Publishing Company, New York, 1969.
19. Fox, L. And Mayers, D. F.: Computing Methods for Scientists and Engineers. Oxford University Press, 1963.



POSTMASTER : If Undeliverable (Section 158
Postal Manual) Do Not Return

"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**