APPENDIX C

SOLUTION OR INVERSION OF LINEAR EQUATIONS

The Computer Time Required to Solve a Symmetric Set of "N " Linear Equations Is Approximately 1/6th the Computer Time Required to Multiply Two "N By N" Matrices

C.1 INTRODUCTION

{ XE "Solution of Equations" }The solution of a large set of equations by hand calculations is a tenuous and time-consuming process. Therefore, before 1960 the majority of structural analysis techniques were based on ***approximations and computational tricks***. Many of those methods, such as moment distribution, allowed the engineer to gain physical insight into the behavior of structures and were forgiving with respect to human computational errors. It was very common for an experienced structural engineering ***human-computer*** to predict the answer to within two significant figures before performing any calculations. At the present time, however, with the assistance of an inexpensive personal computer and efficient computational methods, the structural engineer can solve over 1,000 equations in a few seconds.

{ XE "Cholesky" }{ XE "Matrix Inversion" }The fundamental method currently used to directly solve sets of equilibrium equations is the Gauss elimination that was first used in 1826. Gauss also worked with approximate approaches that resulted in the Gauss-Seidel iterative method in 1860. Most of the methods presented in the last 150 years, such as Cholesky (1916) and Grout (1938), are numerically equivalent to the Gauss elimination method; however, they were easier to use for hand calculations. A modified form of the Gauss elimination method can also be used for matrix inversion.

{ XE "Cramer's Rule" }Cramer’s rule and the theory of determinates, which are presented by many mathematicians as fundamental to matrix analysis, are abstract theorems and are not necessary to understand matrix notation. It is easily shown that the use of Cramer’s rule to solve equations is very numerically inefficient (approximately *N*! numerical operations) and should never be used to solve practical problems in all fields of engineering.

The author’s “hobby” has been the writing of numerically efficient computer programs for the solution of equations. This “pastime” has resulted in the publication of several papers on this topic[1, 2, 3, 4]. Most of this development is summarized in this appendix; therefore, it is not necessary to read the references to fully understand the numerical algorithms presented in this section.

C.2 NUMERICAL EXAMPLE

To illustrate the detailed numerical operations required to solve a set of linear equations by the Gauss elimination method, consider the solution of the following three equations:

$$5.0x_1 + 4.0x_2 + 3.0x_3 = 2.0 \tag{C.1}$$

$$4.0x_1 + 7.0x_2 + 4.0x_3 = -1.0 \tag{C.2}$$

$$3.0x_1 + 4.0x_2 + 4.0x_3 = 3.0 \tag{C.3}$$

First, solve Equation (C.1) for x_1:

$$x_1 = 0.40 - 0.80x_2 - 0.60x_3 \tag{C.4}$$

Second, substitute Equation (C.4) into Equations (C.2) and (C.3) to eliminate x_1 and the following two equations are obtained:

$$3.80x_2 + 1.60x_3 = -2.60 \tag{C.5}$$

$$1.60x_2 + 2.20x_3 = 1.80 \tag{C.6}$$

Third, solve Equation (C.5) for x_2:

$$x_2 = -0.68421 - 0.42105x_3 \tag{C.7}$$

Fourth, substitute Equation (C.7) into Equation (C.6) to eliminate x_2, and the following equation is obtained:

$$x_3 = 1.8865 \tag{C.8a}$$

Back-substitute Equation (C.8a) into Equations (C.7) to obtain:

$$x_2 = -1.4829 \tag{C.8b}$$

Back-substitute Equations (C.8a) and (C8.b) into (C.4) to obtain:

$$x_1 = 0.44828 \tag{C.8c}$$

Therefore, matrix notation is not necessary to solve a set of linear equations. However, the Gauss elimination algorithm can be summarized in a general subscript notation that can be programmed for the computer for an arbitrary number of equations.

It is important to point out that the back-substitution Equations (C.4), (C.7) and (C.8) can be written as the following matrix equation:

$$\begin{bmatrix} 1.00 & 0.80 & 0.60000 \\ 0 & 1.00 & 0.42105 \\ 0 & 0 & 1.00000 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.40000 \\ -0.68421 \\ 1.8865 \end{bmatrix} = \mathbf{L}^T \mathbf{y} \tag{C.9}$$

It will be later shown that Equation (C.9) is identical to the equation used in the matrix factorization solution method.

C.3 THE GAUSS ELIMINATION ALGORITHM

To develop a computer program for the solution of equations, it is first necessary to uniquely define the numerical procedure, ***or algorithm***, by a finite number of clearly defined steps. For Gauss elimination, the initial set of N equations can be written as:

$$\sum_{j=1}^{N} a_{nj} x_j = b_n \qquad n = 1......N \tag{C.10}$$

Starting with the first equation, $n = 1$, we can solve for x_n by dividing all terms in equation n by a_{nn}. Or:

$$x_n = \frac{b_n}{a_{nn}} - \sum_{j=n+1}^{N} \frac{a_{nj}}{a_{nn}} x_j = \overline{b}_n - \sum_{j=n+1}^{N} \overline{a}_{nj} x_j \tag{C.11}$$

Substitution of Equation (C.11) into a typical remaining equation i yields:

$$\sum_{j=n+1}^{N} (a_{ij} - a_{in}\overline{a}_{nj}) x_j = b_i - \overline{a}_{nj} b_n \ \text{ or, } \sum_{j=n+1}^{N} \overline{a}_{ij} x_j = \overline{b}_i \quad i = n+1...Neq \tag{C.12}$$

{ XE "Algorithms for:Gauss Elimination" }This simple Gauss elimination algorithm is summarized in a FORTRAN subroutine shown in Table C.1. Note that within a computer subroutine, the modified terms $\overline{a}_{ij}$ and $\overline{\mathrm{b}}_{\mathrm{i}}$ can be stored in the same locations as the original terms a_{ij} and b_{i}. Therefore, after Equations (C.11) and (C.12) have been applied N times, the unknown x_N is evaluated and stored in the same location as b_N. All other unknowns are evaluated using the back-substitution Equation (C.11). The FORTRAN subroutine allows for an arbitrary number of load vectors. Therefore, for large systems, additional load vectors do not increase the number of numerical operations significantly.

An examination of the subroutine clearly indicates the approximate number of numerical operations for L load conditions is given by :

$$Nop = \frac{1}{3} N^3 + N\,L \tag{C.13}$$

Table C.1 FORTRAN Subroutine to Solve Equations by Gauss Elimination

```
      SUBROUTINE GAUSSEL(A,B,NEQ,LL)
      IMPLICIT REAL*8 (A-H,O-Z)
C---- POSITIVE DEFINITE EQUATION SOLVER ---
      DIMENSION A(NEQ,NEQ),B(NEQ,LL)
C---- FORWARD REDUCTION  ------------------
      DO 500 N=1,NEQ
C---- CHECK FOR POSITIVE-DEFINITE MATRIX –
      IF (A(N,N).LE.0.0D0) THEN
      WRITE (*,*) ‘MATRIX NOT POSSITIVE DEFINITE’
      STOP
      ENDIF
C---- DIVIDE B(N,L) BY A(N,N) -------------------------
      DO 100 L=1,LL
  100 B(N,L) = B(N,L)/A(N,N)
C---- DIVIDE A(N,J) BY A(N,N) -------------------------
      IF (N.EQ.NEQ) GO TO 500 ! CHECK FOR LAST EQUATION
      DO 200 J=N+1,NEQ
  200 A(N,J) = A(N,J)/A(N,N)
C---- MODIFY REMAINING EQUATIONS ----------------------
      DO 500 I=N+1,NEQ
      DO 300 J=N+1,NEQ
  300 A(I,J) = A(I,J) - A(I,N)*A(N,J)
      DO 400 L=1,LL
  400 B(I,N) = B(I,L) - A(I,N)*B(N,L)
C
  500 CONTINUE                ! ELIMINATE NEXT UNKNOWN
C---- BACK-SUBSTITUTIONS ------------------------------
  600 N = N – 1
      IF (N.EQ.0) RETURN
      DO 700 L=1,LL
      DO 700 J=N+1,NEQ
  700 B(N,L) = B(N,L) – A(N,J)*B(N,L)
      GO TO 600
      END
```

Note that the FORTRAN program statements very closely resemble the equations given by the Gauss elimination algorithm. As one notes, the major restriction on this subroutine is that it cannot solve systems that have zero terms on the diagonal of the matrix. However, it can be proven that non-singular stiffness and flexibility matrices will not have zero terms on the diagonal if the displacement u_n and associated force R_n have the same sign convention. Therefore, the subroutine as presented can be used to solve many small structural systems.

C.4 SOLUTION OF A GENERAL SET OF LINEAR EQUATIONS

{ XE "Algorithms for:Solution of General Set of Equations" }It is very easy to modify the subroutine presented in Table C.1 to solve any non-singular sets of linear equations that have zero terms on the diagonal of the **A** matrix during the elimination process. The same Gauss elimination algorithm is used to solve the general set of equations with a very minor modification. The FORTRAN subroutine for this general Gauss elimination algorithm is given in Table C.2.

Before eliminating the next unknown, it is only necessary to search for the largest term that exists in the remaining equations. The largest term is then moved to the a_{nn} position by the interchange of the order of the equations (row interchange) and the interchange of the order of the unknowns (column interchange). The column interchange must be recorded to recover the unknowns in their original order.

If after r equations have been eliminated and all the remaining terms in the **A** matrix are zero (or near zero compared to their initial values), the matrix is singular and the equations cannot be solved. For this case, the matrix is said to have a rank of r. If the set of equations represents force-equilibrium, it simply means that the stiffness matrix has $N - r$ unstable modes or zero energy modes. This is an excellent physical illustration of a ***rank deficient matrix***.

C.5 ALTERNATIVE TO PIVOTING

{ XE "Pivoting" }An alternative method to pivoting can be used to solve a non-positive definite set of equations. Any set of equations can be made symmetrical and positive-definite by the multiplication of both sides of the equation by the transpose of the nonsymmetrical matrix. Or, Equation (C.10) can be written as

$$\overline{\mathbf{A}}x = \overline{\mathbf{B}} \tag{C.14}$$

where, $\overline{\mathbf{A}} = \mathbf{A}^{\mathsf{T}}\mathbf{A}$ is symmetric; and, the effective load is $\overline{\mathbf{B}} = \mathbf{A}^{\mathsf{T}}\mathbf{B}$. The additional numerical effort involved in the matrix multiplication is recovered by the reduction in numerical effort required to solve a symmetrical set of equations. In addition, the interchange of rows and columns, or pivoting, is eliminated.

Table C.2 FORTRAN Subroutine for Solution of a General Set of Equations

```
      SUBROUTINE SOLVE(A,B,IEQ,NEQ,NLV)
C---- SOLUTION OF GENERAL SET OF LINEAR EQUATIONS
C     WHERE
C     A = NEQ x NEQ NON-SYMMETRIX, NON-POSITIVE
C         DEFINITE MATRIX
C     B = NEQ x NLV LOAD MATRIX TO BE REPLACED BY
C         SOLUTION
C     IEQ = TEMPORARY STORAGE ARRAY OF NEQ
C           INTERGERS
C-----------------------------------------------
      REAL*8     A(NEQ,NEQ),B(NEQ,NLV),D,BIG
      INTEGER*4  IEQ(NEQ),NEQ,NLV,II,JJ,I,J,L,N
C---- SET INITIAL UNKNOWN NUMBERS ----------------
      DO 100 N=1,NEQ
 100  IEQ(N) = N
C---- ELIMINATE UNKNOWNS N=1,2....NEQ  -----------
      DO 1000 N=1,NEQ
C---- (1) LOCATE LARGEST TERM REMAINING ----------
      IF (N.NE.NEQ) THEN
      BIG = ABS(A(N,N))
      II = N
      JJ = N
      DO 200 I=N,NEQ
      DO 200 J=N,NEQ
       IF (ABS(A(I,J)).GT.BIG) THEN
       BIG = ABS(A(I,J))
       II = I
       JJ = J
       ENDIF
 200  CONTINUE
C---- (2) CHECK FOR SINGULAR MATRIX --------------
      IF (BIG.EQ.0.0) THEN
      WRITE (*,*) ' MATRIX IS SINGULAR '
      PAUSE 'CORRECT DATA AND RERUN'
      STOP
      ENDIF
C---- (3) INTERCHANGE COLUMNS --------------------
      DO 300 I=1,NEQ
      D = A(I,JJ)
      A(I,JJ) = A(I,N)
 300  A(I,N)  = D
C---- (4) KEEP TRACK OF EQUATION NUMBERS ---------
      J =  IEQ(N)
      IEQ(N) = IEQ(JJ)
      IEQ(JJ)= J
C---- (5) INTERCHANGE ROW "N" AND ROW "II" -------
      DO 400 J=N,NEQ
      D = A(N,J)
      A(N,J) = A(II,J)
 400  A(II,J)= D
C---- (6)INTERCHANGE LOADS ----------------------
      DO 500 L=1,NLV
      D = B(N,L)
      B(N,L) = B(II,L)
 500  B(II,L)= D
      ENDIF
C---- (6)INTERCHANGE LOADS -----------
      DO 500 L=1,NLV
      D = B(N,L)
      B(N,L) = B(II,L)
 500  B(II,L)= D
      ENDIF
C---- (7) DIVIDE LOADS BY DIAGONAL TERM
 550  DO 600 L=1,NLV
 600  B(N,L) =B(N,L)/A(N,N)
C---- (8) DIVIDE ROW BY DIAGONAL TERM -
      IF (N.NE.NEQ) THEN
      DO 700 J=N+1,NEQ
 700  A(N,J) = A(N,J)/A(N,N)
C---- (9) SUBSTITUTE IN REMAINING Eq.--
      DO 900 I=N+1,NEQ
      DO 800 J=N+1,NEQ
 800  A(I,J) = A(I,J) - A(I,N)*A(N,J)
      DO 900 L=1,NLV
 900  B(I,L) = B(I,L) - A(I,N)*B(N,L)
      ENDIFC
 1000 CONTINUE
C---- BACK-SUBSTITUTION ---------------
      IF (NEQ.EQ.1) GO TO 1700
      DO 1300 N=NEQ-1,1,-1
       DO 1200 L=1,NLV
        IF (N.NE.NEQ) THEN
        DO 1100 J=N+1,NEQ
 1100   B(N,L) = B(N,L) - A(N,J)*B(J,L)
        ENDIF
 1200  CONTINUE
 1300 CONTINUE
C---- RETURN UNKNOWNS IN ORIGINAL ORDER
      DO 1600 N=1,NEQ
       DO 1500 I=N,NEQ
        II = IEQ(I)
         IF(II.EQ.N) THEN
          DO 1400 L=1,NLV
          D = B(N,L)
          B(N,L) = B(I,L)
 1400     B(I,L)= D
          IEQ(I) = IEQ(N)
         GO TO 1600 !CHECK NEXT UNKNOWN
         ENDIF
 1500  CONTINUE
 1600 CONTINUE
C---- RETURN TO CALLING PROGRAM -------
 1700 RETURN
      END
```

Mathematicians do not recommend this approach because it increases the "condition number" and the theoretical error. However, for small, well-conditioned systems, it has been the author’s experience that this approach works very well. It also can be proven that this approach will minimize the sum of the square of the error terms.

C.6 MATRIX INVERSION

{ XE "Algorithms for:Matrix Inversion" }{ XE "Matrix Inversion" }The inverse of a matrix can be obtained by setting the matrix **B** to a unit matrix, **I,** and then solving the following equation for the N by N ***x*** matrix (the inverse of **A**):

$$\mathbf{A\,x} = \mathbf{B} \text{ or } \mathbf{A\,A}^{-1} = \mathbf{I} \tag{C.15}$$

The major problem with this approach is that it requires more numerical operations and computer storage than the direct application of the modified Gauss algorithm. It is only necessary to write an algorithm to interchange x_n with b_n and then apply it with $n = 1.....N$. A typical equation is:

$$\sum_{j=1}^{Neq} a_{ij} x_j = b_i \qquad i = 1......N \tag{C.16}$$

By dividing the *n th* equation by a_{nn} , it can be written as:

$$-\sum_{j=1}^{n-1} \overline{a}_{nj} x_j + \frac{b_n}{a_{nn}} - \sum_{j=n+1}^{N} \overline{a}_{nj} x_j = x_n \tag{C.17}$$

Now, x_n can be eliminated from all equations before and after equation *n.* It is then moved to the right-hand side of the equation, and b_n is moved to the left-hand side of the equation. Or:

$$\sum_{j=1}^{n-1} (a_{ij} - a_{in}\overline{a}_{nj})x_j - \frac{a_{jn}}{a_{nn}} b_n + \sum_{j=n+1}^{N} (a_{ij} - a_{in}\overline{a}_{nj})x_j = b_i \tag{C.18}$$
$$\text{for } i = 1..n,\ n+1..N$$

Hence, the new set of Equations can be written, after n transformations, in matrix form as:

$$\mathbf{A}^{(n)}\mathbf{x}^{(n)} = \mathbf{b}^{(n)} \tag{C.19}$$

After *N* transformations:

$$\mathbf{A}^{(N)} = \mathbf{A}^{-1}, \quad \mathbf{x}^{(N)} = -\mathbf{b} \quad \text{and} \quad \mathbf{b}^{(N)} = -\mathbf{x} \tag{C.20}$$

Using this modified Gauss inversion algorithm, it can easily be shown that a closed form solution for a 2 by 2 system is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{C.21}$$

A FORTRAN subroutine that summarizes the matrix inversion algorithm is given in Table C.3. Note that the inverse can be stored in the same locations as the original matrix and no new computer storage is required.

Table C.3 Subroutine to Invert a Matrix by Modified Gauss Elimination

```
      SUBROUTINE INVERT(A,NMAX)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NMAX,NMAX)
C---- MATRIX INVERSION BY MODIFIED GAUSS ELIMINATION
      DO 200 N=1,NMAX
      D = A(N,N)      ! SAVE DIAGONAL TERM
C---- DIVIDE ROW BY DIAGONAL TERM ------------------
      DO 100 J=1,NMAX
  100 A(N,J) = -A(N,J)/D
C---- MODIFY OTHER EQUATIONS ----------------------
      DO 150 I=1,NMAX
      IF(N.EQ.I) GO TO 150
       DO 140 J=1,NMAX
       IF(N.EQ.J) GO TO 140
       A(I,J) = A(I,J) + A(I,N)*A(N,J)
  140  CONTINUE
C---- MODIFY COLUMN -------------------------------
  150 A(I,N) = A(I,N)/D
C---- INVERT DIAGONAL TERM ------------------------
      A(N,N) = 1.0/D
  200 CONTINUE       ! REDUCE NEXT EQUATION
      RETURN         ! INVERSION COMPLETE
      END
```

It should be emphasized that matrix inversion is almost never required in structural analysis. The only exception is the inversion of the 6 by 6 strain-stress matrix. Many textbooks imply that if a large number of load vectors exists, the additional numerical effort associated with matrix inversion is justifiable—not true.

An examination of the matrix inversion subroutine indicates that the approximate number of numerical operations, as previously defined, to invert an N by N matrix is approximately N^3. If there are L load vectors, the total number of numerical operations to invert the matrix and multiply by the load matrix will be:

$$n.o. = N^3 + N^2 L \tag{C.22}$$

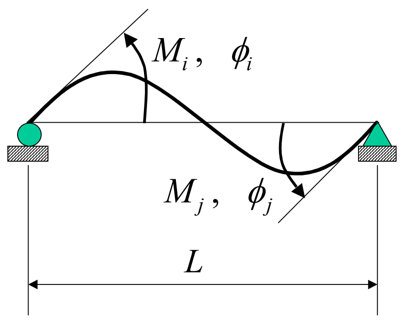
If the set of equations is solved directly by Gauss elimination, the total number of numerical operations is:

$$n.o. = \frac{1}{3} N^3 + N^2 L \tag{C.23}$$

Therefore, matrix inversion is always inefficient compared to the direct solution of equations by Gauss elimination. In addition, if a sparse or banded matrix is inverted, a full matrix may be produced that would require a significant increase in computer storage and execution time.

C.7 PHYSICAL INTERPRETATION OF MATRIX INVERSION

To illustrate the physical interpretation of the matrix inversion algorithm, consider the force-deformation relationship for the simple beam shown in Figure C.1.



$$\frac{4EI}{L}\phi_i + \frac{2EI}{L}\phi_j = M_i$$
$$\frac{2EI}{L}\phi_i + \frac{4EI}{L}\phi_j = M_j$$

Figure C.1 Force-Deformation Behavior of Simple Supported Beam

The force-deformation equations written in matrix form are:

$$\begin{bmatrix} \frac{4EI}{L} & \frac{2EI}{L} \\ \frac{2EI}{L} & \frac{4EI}{L} \end{bmatrix} \begin{bmatrix} \phi_i \\ \phi_j \end{bmatrix} = \begin{bmatrix} M_i \\ M_j \end{bmatrix} \tag{C.24}$$

Note the first column of the stiffness matrix represents the moments developed at the ends as a result of a unit rotation at *i*. The second column of the stiffness matrix represents the moments developed at the ends as a result of a unit rotation at *j*. By applying the inversion algorithm for *n=1,* the following equation is obtained:

$$\begin{bmatrix} \frac{L}{4EI} & -\frac{1}{2} \\ \frac{1}{2} & \frac{3EI}{L} \end{bmatrix} \begin{bmatrix} M_i \\ \phi_j \end{bmatrix} = \begin{bmatrix} \phi_i \\ M_j \end{bmatrix} \tag{C.25}$$

Each term in the modified matrix has a physical meaning. The first column, with $\phi_j = 0$, a unit moment applied at *i* produces a rotation of $L/\ 4EI$ at *i* and a moment of $1/2$ at *j*. The second column, with $M_j = 0$, a unit rotation applied at *j* produces a rotation of $-1/2$ at *i* and a moment of $3EI\,/\,L$ at *j*.

After application of the inversion algorithm for *n=2,* the following flexibility equation is obtained:

$$\frac{L}{12EI} \begin{bmatrix} 4 & -2 \\ -2 & 4 \end{bmatrix} \begin{bmatrix} M_i \\ M_j \end{bmatrix} = \begin{bmatrix} \phi_i \\ \phi_j \end{bmatrix} \tag{C.26}$$

Therefore, the abstract mathematical procedure of matrix inversions has a very physical interpretation. Each term in the matrix, after an interchange of x_n and b_n, represents a displacement or force per unit of displacement or forces. It also indicates, using the displacement method of structural analysis for the solution of joint equilibrium equations, that the diagonal term has the units of stiffness and cannot be negative or zero for a stable structural system; therefore, there is no need to pivot during the solution algorithm.

C.8 PARTIAL GAUSS ELIMINATION, STATIC CONDENSATION AND SUBSTRUCTURE ANALYSIS

{ XE "Algorithms for:Partial Gauss Elimination" }{ XE "Algorithms for:Static Condensation" }{ XE "Partial Gauss Elimination" }{ XE "Static Condensation" }{ XE "Substructure Analysis" }In the displacement method of structural analysis the stiffness matrix times the joint displacements are equal to the external joint loads. The application of the Gauss elimination algorithm to the solution of these equilibrium equations has a very important physical interpretation. The initial terms on the diagonal of the stiffness matrix are in the units of force per unit of deformation with all other degrees of freedom in the structure fixed. The elimination of an unknown displacement is equivalent to releasing the displacement, and the loads are carried over to the other degrees of freedom in the structure. The stiffness terms at the adjacent degrees of freedom are modified to reflect that movement is allowed at the degrees of freedom eliminated. Therefore, the solutions of the equilibrium equations by applying the Gauss elimination algorithm to all degrees of freedom can be interpreted, *by a structural engineer over the age of fifty*, as one giant cycle of moment distribution in which iteration is not required.

What is of greater significance, however, is if the algorithm is stopped at any point, the remaining equations represent the stiffness matrix with respect to the degrees of freedom not eliminated. This substructure stiffness can be extracted and used as a super element in another structural model. Also, the loads associated with the eliminated displacements are carried over to the substructure joints and must be applied to the new structural model. After the displacements associated with the substructure joints have been found, the eliminated displacements can be calculated by back-substitution.

This partial Gauss elimination algorithm is also called the static condensation method. The algorithm and a FORTRAN subroutine are summarized in Table C.4. Note that the stiffness matrix is still stored in square form; however, the number of numerical operations is reduced by recognition of the symmetry of the stiffness matrix, and some of the operations on zero terms are skipped.

Table C.4 Partial Gauss Elimination Algorithm and Subroutine

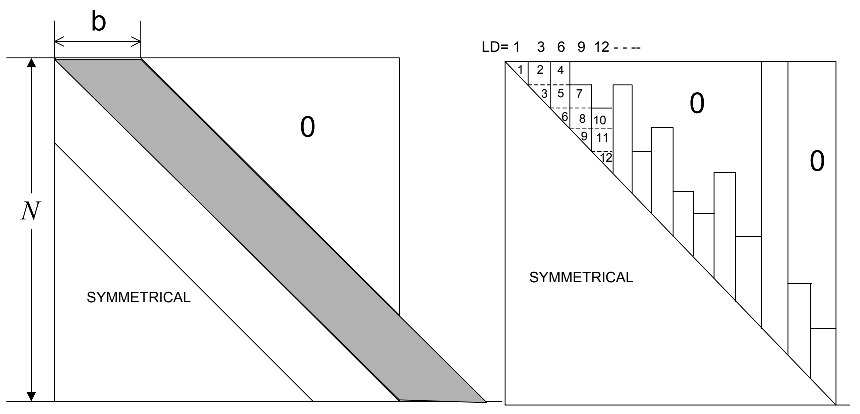
```
      SUBROUTINE SUBSOL(K,R,NEQ,LEQ,LL,MOP)
      REAL*8 K(NEQ,NEQ),R(NEQ,LL),T,ZERO
C---- SUBSTRUCTURE EQUATION SOLVER - WHERE -------------------
C     K   = STIFFNESS MATRIX TO BE REDUCED
C     R   = LOAD VECTORS - REPLACED BY DISPLACEMENTS
C     NEQ = TOTAL NUMBER OF EQUATIONS
C     LEQ = NUMBER OF MASSLESS D.O.F. TO BE ELIMINATED
C     LL  = NUMBER OF LOAD VECTORS
C     MOP = 0 TRIANGULARIZATION AND COMPLETE SOLUTION
C     MOP = 1 TRIANGULARIZATION ONLY
C     MOP = 2 LOAD REDUCTION ONLY
C     MOP = 3 DISPLACEMENT RECOVERY ONLY
      DATA ZERO /0.0D0/
C-------------------------------------------------------------
      IF(MOP.EQ.3) GO TO 800   ! DISPLACEMENT RECOVERY ONLY
      IF(MOP.EQ.2) GO TO 500   ! LOAD REDUCTION ONLY
C---- TRIANGULARIZATION --------------------------------------
      DO 400 N=1,LEQ
      IF(K(N,N).LE.ZERO) STOP ' STRUCTURE UNSTABLE '
      IF (N.EQ.NEQ) GO TO 400  ! CHECK FOR LAST EQUATION
      DO 300 J=N+1,NEQ
       IF(K(N,J).NE.ZERO) THEN ! OPERATE ONLY ON NONZERO TERMS
        T = K(N,J)/K(N,N)
        DO 200 I=J,NEQ         ! MODIFY OTHER EQUATIONS
  200   K(J,I) = K(J,I) - K(N,I)*T
        K(N,J) = T
       ENDIF
  300 CONTINUE                 ! END OF J LOOP
  400 CONTINUE                 ! END OF N LOOP
      IF(MOP.EQ.1) RETURN      ! TRIAGULARIZE  ONLY
C---- FORWARD REDUCTION OF LOAD VECTORS ----------------------
  500 DO 700 N=1,LEQ
       DO 650 L=1,LL           ! REDUCE ALL LOAD VECTORS
       IF (N.EQ.NEQ) GO TO 650
       DO 600 J=N+1,NEQ
  600  R(J,L) = R(J,L) - K(N,J)*R(N,L)
  650  R(N,L) = R(N,L)/K(N,N)
  700 CONTINUE                 ! END OF N LOOP
      IF(MOP.EQ.2) RETURN      ! RETURN TO CALLING PROGRAM
C---- RECOVERY OF DISPLACEMENTS ------------------------------
  800 DO 1000 NN=1,LEQ,1
      N = LEQ - NN + 1
      IF (N.EQ.NEQ) GO TO 1000 ! LAST EQUATION HAS BEEN SOLVED
       DO 900 L=1,LL           ! RECOVER ALL LOAD CONDITIONS
       DO 900 J=N+1,NEQ
  900  R(N,L) = R(N,L) - K(N,J)*R(J,L)
 1000 CONTINUE                 ! END OF N LOOP
      RETURN                   ! RETURN  TO CALLING PROGRAM
C-------------------------------------------------------------
      END
```

This subroutine can be used to solve a full set of equations. For this case, it is apparent that the number of numerical operations required for a solution of a complete set of equations is:

$$n.o. = \frac{1}{6} N^3 + N^2 L \tag{C.27}$$

C.9 EQUATIONS STORED IN BANDED OR PROFILE FORM

{ XE "Banded Equations" }{ XE "Profile Storage of Stiffness Matrix" }A careful examination of the Gauss elimination algorithm as applied to the global stiffness matrix indicates that new terms in the stiffness matrix are only generated below the first non-zero term in each column. Also, only the terms above the diagonal need to be stored during the solution procedure. Therefore, the symmetric stiffness matrix can be stored in banded or profile form, as indicated in Figure C.2.



A. Rectangular Banded Storage B. Profile or Envelope Type of Storage

Figure C.2 Methods of Storage for Symmetric Stiffness Matrices

The banded form of storage for the stiffness matrix was used in the early years of the development of structural analysis programs. For example, SAP-IV used a blocked-banded approach. However, the banded storage method initially required that the user number the nodes in an order that would minimize the bandwidth. Later, bandwidth minimization algorithms were developed; however, a large number of zero terms still existed within the band for most structural systems.

The profile method of storage reduces the computer storage requirements and reduces the operation on zero terms. For this method, the stiffness matrix is stored in one dimensional form, from the first non-zero term in a column to the diagonal term, as shown in Figure C.2.B. In addition, a one-dimensional integer array, LD, indicates the location of the diagonal term for each column. The profile storage method is used in most modern structural analysis programs. Many different algorithms have been developed to reduce the number of numerical operations and computer storage requirements for stiffness matrices. Within the SAP90 and SAP2000 programs, three different algorithms are tried, and the one that requires the minimum computer storage is used.

From the fundamental Gauss elimination equations, it is apparent that the banded storage method requires the following number of numerical operations:

$$Nop = \frac{1}{2} N\, b^2 - \frac{1}{3} b^2 + N\, b L \tag{C.28}$$

Note that for a small half-bandwidth *b*, the number of numerical operations to solve a set of equations can be very small, compared to the formation of element matrices and the calculation of member forces and stresses.

In the case of profile storage, the number of numerical operations to solve the set of equations can be estimated from:

$$Nop = \sum_{n=1}^{N} \frac{1}{2} h_n^2 + 2 h_n L \tag{C.29}$$

The column height is given by $h_n = LD(n) - LD(n-1)$. Note that both Equations (C.28) and (C.29) reduce to Equation (C.27) for a full stiffness matrix.

C.10 LDL FACTORIZATION

{ XE "Algorithms for:LDL Factorization" }{ XE "LDL Factorization" }In books on numerical analysis, the most common approach proposed to solve a set of symmetric equations is the $\mathbf{LDL}^\mathbf{T}$ factorization, or decomposition, method. This approach involves the identical number of numerical operations, computer storage and accuracy as the Gauss elimination method; however, it lacks the physical analogy that exists with the partial Gauss elimination method. On the other hand, the factorization approach has advantages in that the operations on the stiffness and load matrices are separated. Also, error estimations can be obtained from the method, and it can be directly extended to the solution of eigenvector or Ritz vector analysis. In any case, we can use the advantages of both approaches without being forced to use one or the other.

The set of linear equations to be solved is written in the following matrix form:

$$\mathbf{Ax} = \mathbf{b} \quad \text{or,} \quad \mathbf{LDL}^{\mathbf{T}}\mathbf{x} = \mathbf{b} \quad \text{or,} \quad \mathbf{LDy} = \mathbf{b} \quad \text{where,} \quad \mathbf{L}^{\mathbf{T}}\mathbf{x} = \mathbf{y} \tag{C.30}$$

where $\mathbf{A}$ is an N by N symmetric matrix that contains a large number of zero terms. The N by M $\mathbf{x}$ displacement and $\mathbf{b}$ load matrices indicate that more than one load condition can be solved at the same time. The solution of equations is divided into the following three steps:

C10.1 Triangularization or Factorization of the A Matrix

The first step in the solution of the set of linear equations is to factor the $\mathbf{A}$ matrix into the product of a lower triangular matrix $\mathbf{L}$**,** with all diagonal terms equal to 1.0, times an upper triangular matrix $\mathbf{U}$. Or, in the case of a symmetric matrix:

$$\mathbf{A} = \mathbf{LU} = \mathbf{LDL}^{\mathrm{T}} \tag{C.31}$$

From the basic definition of matrix multiplication, the following equation can be written:

$$A_{ij} = \sum_{k=1}^{N} L_{ik} U_{kj} = \sum_{k=1}^{i} L_{ik} U_{kj} \tag{C.32}$$

From Equation (C.32) a careful examination of the limits of the summation indicates that the *n th* column of the $\mathbf{U}$ matrix and the *n th* row of the $\mathbf{L}$ matrix can be calculated, in the order shown in Figure C.3, from the following equations:

$$U_{in} = A_{in} - \sum_{k=1}^{i-1} L_{ik} U_{kn} \tag{C.33}$$

$$L_{nj} = \frac{U_{nj}}{D_{jj}} \tag{C.34}$$

From Equation (C.34) the diagonal term is:

$$D_{nn} = U_{nn} = A_{nn} - \overline{A}_{nn} \quad \text{where} \quad \overline{A}_{nn} = \sum_{k=1}^{n-1} L_{nk} U_{kn} \tag{C.35}$$

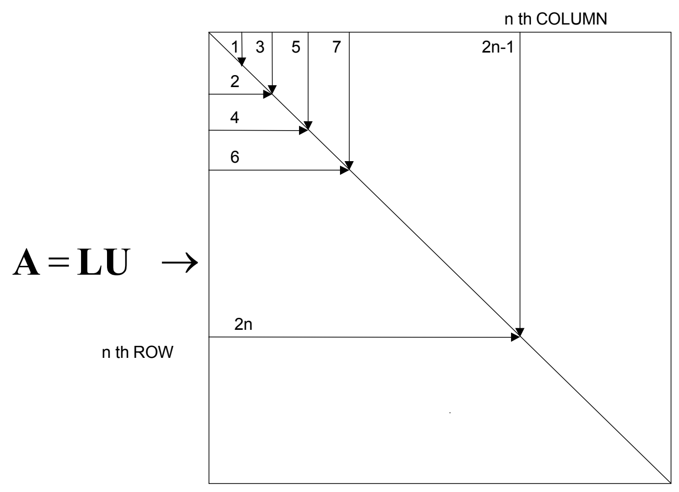


Figure C.3 Order of Calculation of the Rows and Columns in Factored Matrix

If these equations are evaluated in the appropriate order, it is possible to store the $\mathbf{L}^T$ matrix in the same locations as the original **A** matrix. Because the L_{nn} are always equal to one, the diagonal terms D_{nn} can be stored on the diagonal of the original matrix. Hence, it is possible to factor the matrix without additional storage requirements. Note that the lower limit of the "k" summation can be changed to the location of the first non-zero term in the column or row.

C10.2 Forward Reduction of the b Matrix

The next step in the solution of linear equations is to conduct a forward reduction of the load vector by solving the following set of equations where $\mathbf{y}=\mathbf{L}^{\mathbf{T}}\mathbf{x}$:

$$\mathbf{LD}\,\mathbf{y}=\mathbf{b} \tag{C.36}$$

The solution is given by:

$$y_{nm}=\frac{b_{nm}}{D_{nn}}-\sum_{k=1}^{n-1} L_{nk}\,y_{km} \qquad n=1\ldots.\mathrm{N} \tag{C.37}$$

C10.3 Calculation of x by Backsubstitution

It is apparent that the unknowns **x** can now be calculated from:

$$x_{nm} = y_{nm} - \sum_{k=1}^{n-1} L_{kn} y_{km} \qquad n = N \ldots . 1 \tag{C.38}$$

The forward reduction and back substitution is conducted for all load vectors from m = 1 to the total number of load vectors. The fact that the factorization phase is completely separate from the solution phase allows the factorized matrix to be used for both the static and dynamic phase of the solution. FORTRAN subroutines, using profile storage, are given in reference [3].

The determinant of $\mathbf{LDL^T}$ is the product of the determinant of each matrix. Hence, the product of the diagonal terms of the **D** matrix is the determinant of the matrix. The determinant of a matrix is of little physical value. However, the mathematical properties of the sequence of diagonal terms D_{nn} are very significant.

The three equation given by Equations (C.1), (C.2) and (C.3) can be factored as:

$$\mathbf{L}^T\mathbf{DL} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.8 & 1.0 & 0.0 \\ 0.6 & .421 & 1.0 \end{bmatrix} \begin{bmatrix} 5.0 & 0 & 0 \\ 0 & 3.8 & 0 \\ 0 & 0 & 1.527 \end{bmatrix} \begin{bmatrix} 1.00 & 0.80 & 0.600 \\ 0 & 1.00 & 0.421 \\ 0 & 0 & 1.000 \end{bmatrix} \tag{C.39}$$

Note that the **L** matrix is identical to the Gauss elimination back-substitution matrix shown in Equation (C.9). Also,

$$\mathbf{y} = \begin{bmatrix} 0.40000 \\ -0.68421 \\ 1.8865 \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 0.44828 \\ -1.4829 \\ 1.8865 \end{bmatrix} \qquad \text{(C.40a and C.40b)}$$

Therefore, there is very little difference between the factorization approach and the Gauss elimination method.

C.11 DIAGONAL CANCELLATION AND NUMERICAL ACCURACY

{ XE "Diagonal Cancellation" }{ XE "Numerical Accuracy" }The numerical accuracy of the solution of a set of linear equations can be estimated by the examination of the expression for the diagonal terms, Equation (C.35). Or, in simplified form:

$$D_{nn} = A_{nn} - \overline{A}_{nn} \tag{C.41}$$

Where A_{nn} is the original unmodified term in the matrix and $\overline{A}_{nn}$ is the modification to the term to produce the new diagonal term D_{nn}. We know that if D_{nn} is zero, or very near zero, the matrix is singular and the solution algorithm must be terminated. Within modern computer systems, numbers have a range of approximately 10^{-300} to 10^{300}; therefore, an exact zero number is almost impossible to detect because of round off errors. What is really important, however, is the size of the original diagonal term compared to the reduced diagonal term. Therefore, the number of significant decimal ***figures lost*** can be estimated from:

$$f.l. = \log_{10}(A_{nn}) - \log_{10}(\overline{A}) \tag{C.42}$$

Because all normal engineering calculations are completed within the computer using approximately 15 significant figures, a loss of over 12 figures indicates that significant errors may exist; hence, the structural engineer should be warned, and the computer model of the structure examined. This problem exists if the model lacks appropriate boundary conditions, a collapse mechanism exists or if members with large relative stiffness are used.

C.12 SUMMARY

The most general approach for the solution, inversion and condensation of equilibrium equations is Gauss elimination. In programming this method for use in structural analysis programs, sparse storage and profile minimization [4] is required to minimize the numerical effort. Diagonal cancellation must be checked to detect numerical problems.

For the solution of structural equilibrium equations, pivoting should not be used. Before eliminating a degree of freedom, the diagonal term always represents the stiffness associated with the degree of freedom. Hence, a zero or near zero diagonal term indicates that the computational model of the structure is unstable.

Given the speed of a computer system, number of operations per second, it is possible to accurately predict the computer time to solve a set of equations. Whereas the computer time required by an iterative solver, which can be faster for certain large systems, cannot be accurately predicted. In addition, the triangularized stiffness matrix can be used directly to generate mode shapes required for a dynamic analysis.

C.13 REFERENCES

1 Wilson, E. 1974. "The Static Condensation Algorithm," *International Journal for Numerical Methods in Engineering*. Vol. 8. January. pp. 198-203.

2. { XE "Doherty, W. P." }Wilson, E.L., K. J. Bathe and W. P. Doherty. 1974. "Direct Solution of Large Systems of Linear Equations," *Computers and Structures.* Vol. 4. January. pp. 363-372.

3. { XE "Dovey, H. H." }Wilson E.L., and H. H. Dovey. 1979. "Solution or Reduction of Equilibrium Equations for Large Complex Structural Systems," *Advances in Engineering Software.* Vol. 1, No. 1. pp. 19-25.

4. { XE "Hoit, M." }Hoit M. and E. L. Wilson. 1983. "An Equation Numbering Algorithm Based on a Minimum Front Criterion," *J. Computers and Structures.* Vol. 16, No. 1-4. pp. 225-239.