Appendix D

Solution of Simultaneous Equations

Application of some numerical methods to EM problems often results in a set of simultaneous equations

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

or

$$[A][X] = [B] \tag{D.1b}$$

where $[A]$ is the coefficient matrix, $[X]$ is the column matrix of the unknowns to be determined, and $[B]$ is the column matrix of constants. Familiarity with the various techniques for solving Eq. (D.1) is therefore vital. In this appendix, we provide a brief coverage of direct and iterative procedures for solving Eq. (D.1); direct methods are more versatile for linear problems, while iterative methods are suitable for nonlinear problems. We also consider various techniques for solving eigenvalue systems $[A][X] = \lambda[X]$.

D.1 Elimination Methods

Elimination methods constitute the simplest direct approach to the solution of a set of simultaneous equations. They usually involve successive elimination of the unknowns by combining equations. Such methods include Gauss's method, Gauss-Jordan, Cholesky's or Crout's method, and the square-root method. Only Gauss's and Cholesky's methods will be discussed. The reader should consult [1]–[4] for the treatment of other methods.

D.1.1 Gauss's Method

This simple method involves eliminating one unknown at a time and proceeding with the remaining equations. This leads to a set of simultaneous equations in triangular form from which each unknown is determined by back-substitution. To describe this method, consider Eq. (D.1b), i.e.,

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \tag{D.2a}$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \tag{D.2b}$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \tag{D.2c}$$

We divide Eq. (D.2a) by a_{11} to give

$$x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n = b'_1 \tag{D.3}$$

where the primes denote that the coefficients are new. We multiply Eq. (D.3) by $-a_{i1}$ for $i = 2, 3, \ldots, n$ and add Eq. (D.3) to the ith equation in (D.2) to eliminate x_1 from other equations so that Eq. (D.2) becomes

$$x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n = b'_1 \tag{D.4a}$$
$$a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2 \tag{D.4b}$$
$$\vdots$$
$$a'_{n2}x_2 + \cdots + a'_{nn}x_n = b'_n \tag{D.4c}$$

Equation (D.2a) used to eliminate x_1 from other equations is called the *pivot equation* and a_{11} is called the *pivot coefficient*. We now use Eq. (D.4b) as the pivot equation and we take similar steps to eliminate x_2 from all equations following the pivot equation. Continuing this reduction procedure eventually leads to an equivalent triangular set of equations:

$$\begin{aligned} x_1 + u_{12}x_2 + u_{13}x_3 + \cdots + u_{1n}x_n &= c_1 \\ x_2 + u_{23}x_2 + \cdots + u_{2n}x_n &= c_2 \\ x_3 + \cdots + u_{3n}x_n &= c_3 \\ &\vdots \\ x_n &= c_n \end{aligned} \tag{D.5}$$

This completes the first phase known as *forward elimination* in the Gauss algorithm, and the system in Eq. (D.5) is said to be in *upper triangular* form. The second phase known as *back substitution* involves solving for the unknowns in Eq. (D.5) by starting

at the bottom. That is,

$$\begin{aligned} x_n &= c_n \\ x_{n-1} &= c_{n-1} - u_{n-1,n}x_n \\ &\vdots \\ x_1 &= c_1 - u_{12}x_2 - \cdots - u_{1n}x_n \end{aligned} \tag{D.6}$$

In summary, this algorithm can be stated as:

Forward elimination

$$\begin{aligned} a'_{kj} &= a_{kj}/a_{kk}, \quad b'_k = b_k/a_{kk}, \quad j = k, k+1, \ldots, n \\ a'_{ij} &= a_{ij} - a_{ik}a'_{kj}, \quad i = k+1, \ldots, n \\ b'_i &= b_i - a_{ik}b'_k, \quad i = k+1, \ldots, n \end{aligned} \tag{D.7a}$$

Backward substitution

$$\begin{aligned} x_n &= b_n, \qquad \text{for the last row} \\ x_i &= b_i - \sum_{j=i+1}^{n} a_{ij}x_j, \quad i = n-1, \ldots, 1 \end{aligned} \tag{D.7b}$$

Based on the idea outlined above, a general FORTRAN subroutine for solving a set of simultaneous equations by Gaussian elimination is shown in Fig. D.1.

D.1.2 Cholesky's Method

This method, also known as Crout's method or the method of matrix decomposition, involves determining a lower triangular matrix that will reduce the original system in Eq. (D.1) to a unit upper triangular matrix. If the original system

$$[A][X] = [B] \tag{D.1a}$$

or

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

can be redefined in the upper unit triangular matrix $[T]$ such that

$$[T][X] = [C] \tag{D.8a}$$

```
0001    C*********************************************************
0002    C  THIS SUBROUTINE EMPLOYS GAUSSIAN ELIMINATION METHOD
0003    C  TO SOLVE A SET OF SIMULTANEOUS EQUATIONS [A][X] = [B]
0004    C  IDM = DIMENSION OF [A] IN THE CALLING PROGRAM
0005    C  N   = ACTUAL SIZE OF [A] IN THE CALLING PROGRAM
0006    C*********************************************************
0007
0008            SUBROUTINE GAUSS(A,B,X,N,IDM)
0009            DIMENSION A(IDM,IDM), B(IDM), X(IDM)
0010    C
0011    C  FORWARD ELIMINATION
0012    C
0013           DO 40 K=1,N-1
0014           DO 30 I=K+1,N
0015           FACTOR = A(I,K)/A(K,K)
0016    C
0017    C  CALCULATE ELEMENTS OF THE NEW MATRIX
0018    C
0019           DO 20 J=K+1,N
0020           A(I,J) =A(I,J) - FACTOR*A(K,J)
0021    20   CONTINUE
0022           A(I,K) = FACTOR
0023           B(I) = B(I) - FACTOR*B(K)
0024    30   CONTINUE
0025    40   CONTINUE
0026    C
0027    C  BACK SUBSTITUTION PROCESS BEGINS
0028    C
0029           X(N) = B(N)/A(N,N) !1ST STEP
0030           DO 60 I=N-1,1,-1
0031           SUM = 0.0
0032           DO 50 J=I+1,N
0033           SUM = SUM + A(I,J)*X(J)
0034    50   CONTINUE
0035           X(I) = ( B(I) - SUM )/A(I,I)
0036    60   CONTINUE
0037           RETURN
0038           END
```

Figure D.1
Gauss elimination method of solving $[A][X] = [B]$**.**

or

$$\begin{bmatrix} 1 & T_{12} & \dots & T_{1n} \\ 0 & 1 & \dots & T_{2n} \\ \vdots & & & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{D.8b}$$

the unknown x_i can be obtained by back substitution. Let $[A]$ be a product of an upper unit triangular matrix $[T]$ and a lower triangular matrix $[L]$, i.e.,

$$\big[L\big]\big[T\big] = \big[A\big] \tag{D.9}$$

Since

$$\big[L\big]\big[TX - C\big] = 0 = \big[AX - B\big], \tag{D.10}$$

it follows that

$$[L][C] = [B] \tag{D.11}$$

For computational reasons, it is convenient to work with the augmented form of the matrices. The augmented matrix is obtained by adding the column vector of constants to the square coefficient matrix. Equations (D.8) and (D.11) may be combined to give

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \vdots b_1 \\ a_{21} & a_{22} & \dots & a_{2n} \vdots b_2 \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \vdots b_n \end{bmatrix} =$$

$$\begin{bmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & L_{22} & \dots & 0 \\ \vdots & & & \\ L_{n1} & L_{n2} & \dots & L_{nn} \end{bmatrix} \begin{bmatrix} 1 & T_{12} & \dots & T_{1n} \vdots c_1 \\ 0 & 1 & \dots & T_{2n} \vdots c_2 \\ \vdots & & & \\ 0 & 0 & \dots & 1 \vdots c_n \end{bmatrix} \tag{D.12a}$$

or

$$[A \vdots B] = [L][T \vdots C] \tag{D.12b}$$

The elements of $[L]$, $[T]$, and $[C]$ can be defined in terms of $[A]$ and $[B]$ as follows [1, 2, 5]:

$$\begin{aligned} L_{ij} &= a_{ij} - \sum_{k=1}^{j-1} L_{ik} T_{kj}, \quad i \geq j,\ i = 1, 2, \dots, n \\ L_{ij} &= a_{i1}, \quad j = 1 \\ T_{ij} &= \frac{1}{L_{ii}} \left(a_{ij} - \sum_{k=1}^{i-1} L_{ik} T_{kj} \right), \quad i < j,\ j = 2, 3, \dots, n \\ T_{ij} &= a_{ij}/a_{11}, \quad i = 1 \\ c_i &= \frac{1}{L_{ii}} \left(b_i - \sum_{k=1}^{i-1} L_{ik} c_k \right), \quad i = 2, 3, \dots, n \\ c_1 &= b_1/L_{11} \end{aligned} \tag{D.13}$$

The unknown x_i are obtained by back substitution as follows:

$$\begin{aligned} x_n &= c_n \\ x_i &= c_i - \sum_{j=i+1}^{n} T_{ij} x_j, \quad i = 1, 2, \dots, n-1 \end{aligned} \tag{D.14}$$

Cholesky's method can easily be applied in calculating the determinant of $[A]$. Since

$$\det\left[A\right] = \det\left[L\right]\det\left[T\right] \tag{D.15}$$

and $\det[T] = 1$ due to the fact that $T_{ii} = 1$, it follows that

$$\det\left[A\right] = \det\left[L\right] = L_{11}L_{22}\ldots L_{nn}$$

or

$$\det\left[A\right] = \prod_{i=1}^{n} L_{ii} \tag{D.16}$$

Figure D.2 shows a subroutine based on Cholesky's method of solving a set of simultaneous equations.

D.2 Iterative Methods

The direct or elimination method for solving a system of simultaneous equations can be used for $n = 25$ to 60. This number can be greater if the system is well conditioned or the matrix is sparse. For very large systems, say $n = 100$ or even 1000, elimination methods become time-consuming and prove inadequate due to roundoff error. For these types of problems, indirect or iterative methods provide an alternative.

D.2.1 Jacobi's Method

This is the simplest iterative method. If the system in Eq. (D.1) is rearranged so that the ith equation is explicit in x_i, we obtain

$$x_1 = \frac{1}{a_{11}}\left[b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n\right] \tag{D.16a}$$

$$x_2 = \frac{1}{a_{22}}\left[b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n\right] \tag{D.16b}$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}\left[b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}\right] \tag{D.16c}$$

assuming that the diagonal elements are all nonzero. We start the solution process by using guesses for the x's, say $x_1 = x_2 = \cdots = x_n = 0$. The first equation can be solved for x_1, the second for x_2, and so on. If we denote the estimates after the

```
0001    C*******************************************************
0002    C    THIS SUBROUTINE APPLIES CHOLESKY ELIMINATION METHOD
0003    C    TO SOLVE THE SYSTEM [AA][X] = [B]
0004    C        [A]  = AUGMENTED MATRIX = [AA : B]
0005    C        MAX  = MAXIMUM ROW DIMENSION OF [AA] AND [X] MATRIX
0006    C        MAX1 = MAXIMUM COLUMN DIMENSION OF [AA] = MAX + 1
0007    C        N    = ACTUAL ROW SIZE OF [AA]
0008    C        M    = ACTUAL COLUMN SIZE OF [AA] = N+1
0009    C        [X]  = STORES THE RESULTS
0010    C
0011    C REF: [2, PP. 184, 185]
0012    C*******************************************************
0013          SUBROUTINE CHOLESKY(A,MAX,MAX1,N,M,X)
0014    C
0015    C  NOTE:  THIS PROGRAM CAN BE USED TO SOLVE COMPLEX EQUATIONS
0016    C         BY REPLACING EVERY SUM = 0.0 WITH SUM = (0.0,0.0)
0017    C         USING THE NEXT LINE INSTEAD OF THE ONE FOLLOWING IT
0018    C     COMPLEX A,X,SUM
0019          REAL  A,X,SUM
0020          DIMENSION A(MAX,MAX1), X(MAX)
0021
0022    C
0023    C  CALCULATE THE FIRST ROW OF THE [T] MATRIX
0024    C
0025          DO 10 J = 2, M
0026             A(1,J)=A(1,J)/A(1,1)  !CALCULATES T(1,J)
0027    10     CONTINUE
0028    C
0029    C CALCULATE OTHER ELEMENTS OF [T] AND [L] MATRICES
0030    C
0031          DO 60 I=2,N
0032             DO 30 II=I,N
0033             SUM = 0.0
0034             DO 20 K=1,I-1
0035               SUM = SUM + A(II,K)*A(K,I)
0036    20       CONTINUE
0037               A(II,I) = A(II,I) - SUM  !CALCULATES [L]
0038    30  CONTINUE
0039                DO 50 J = I+1,M
0040                   SUM = 0.0
0041                   DO 40 K=1,I-1
0042                      SUM= SUM + A(I,K)*A(K,J)
0043    40  CONTINUE
0044                A(I,J) = ( A(I,J)-SUM )/A(I,I) !CALCULATES [T]
0045    50  CONTINUE
0046    60  CONTINUE
0047    C
0048    C SOLVE FOR [X] BY BACK SUBSTITUTION
0049    C
0050          X(N) = A(N,M)
0051          DO 80 NN = 1,N-1
0052             SUM = 0.0
0053             I= N - NN
0054             DO 70 J = I+1,N
0055               SUM = SUM + A(I,J)*X(J)
0056    70  CONTINUE
0057          X(I) = A(I,M) - SUM
0058    80    CONTINUE
0059          RETURN
0060          END
```

Figure D.2

Cholesky's elimination method of solving $[A][X] = [B]$.

kth iteration as $x_1^k, x_2^k, \ldots, x_n^k$, the estimates after $(k+1)$th iteration can be obtained from Eq. (D.16) as

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j\neq 1}^{n} a_{ij} x_j^k \right], \qquad i = 1, 2, \ldots, n \tag{D.17}$$

The iteration process is continued until values of x_i at two successive iterations are within an allowable prescribed deviation.

Convergence is measured in terms of the change in x_i from the kth iteration to the next. If we compute

$$d_i = \left| \frac{x_i^{k+1} - x_i^k}{x_i^{k+1}} \right| \cdot 100\% \tag{D.18}$$

for each x_i, convergence can be checked using the criterion

$$d_i < \epsilon_s \tag{D.19}$$

where ϵ_s is a specified small quantity. A better test would be to compute

$$d = \frac{\sum_{i=1}^{n} \left| x_i^{k+1} - x_i^k \right|}{\sum_{i=1}^{n} \left| x_i^{k+1} \right|} \cdot 100\% \tag{D.20}$$

and require that $d < \epsilon_s$.

D.2.2 Gauss-Seidel Method

This is the most commonly used iterative method. In Jacobi's method the entire set of x_i from the kth iteration is used in calculating the new set during the $(k+1)$th iteration, whereas the most recently calculated value of each variable is used at each step in the Gauss-Seidel method. This makes the Gauss-Seidel method converge more rapidly than (about twice as) Jacobi's method and is always used in preference to it. Instead of Eq. (D.17), we use

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \qquad i = 1, 2, \ldots, n \tag{D.21}$$

A computer program based on this method is displayed in Fig. D.3.

D.2.3 Relaxation Method

This is a slight modification of the Gauss-Seidel method and is designed to enhance convergence. If x_i^k is added to the right-hand side of Eq. (D.21) and $(a_{ii} x_i^k)/a_{ii}$ is

```
0001    C*********************************************************
0002    C  THIS SUBROUTINE EMPLOYS GAUSS-SEIDEL ITERATIVE METHOD
0003    C  TO SOLVE A SET OF SIMULTANEOUS EQUATIONS [A][X] = [B]
0004    C  IDM = DIMENSION OF [A] IN THE CALLING PROGRAM
0005    C    N = ACTUAL SIZE OF [A] IN THE CALLING PROGRAM
0006    C*********************************************************
0007
0008            SUBROUTINE GSEIDEL(A,B,X,N,IDM)
0009            DIMENSION A(IDM,IDM), B(IDM), X(IDM)
0010    C
0011    C  INITIALIZATION
0012    C
0013          DO 10 I=1,N
0014          X(I) = 0.0
0015    10    CONTINUE
0016          K = 0 !NO. OF ITERATIONS
0017          TOL = 1.0E-30  !TOLERANCE FOR ZERO
0018    20    RES = 0.0
0019          DO 40 I=1,N
0020          SUM = 0.0
0021          DO 30 J=1,N
0022          IF(J.EQ.I) GO TO 30
0023          SUM = SUM + A(I,J)*X(J)
0024    30    CONTINUE
0025          XNEW = ( B(I) - SUM )/A(I,I)
0026    C
0027    C  FIND THE LARGEST RESIDUE
0028    C
0029          DIFF = ABS( X(I) - XNEW )
0030          IF( DIFF.GT.RES ) RES = DIFF
0031    C
0032    C  REPLACE OLD VALUE WITH NEWLY COMPUTED VALUE
0033    C
0034          X(I) = XNEW
0035    40    CONTINUE
0036    C
0037    C  TEST FOR CONVERGENCE
0038    C
0039          K = K + 1
0040          PRINT *,K
0041          IF( RES.GE.TOL ) GO TO 20
0042           RETURN
0043           END
```

Figure D.3
Gauss-Seidel iterative method of solving $[A][X] = [B]$**.**

subtracted from it, we obtain

$$x_i^{k+1} = x_i^k + \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right], \qquad i = 1, 2, \ldots, n \quad \text{(D.22)}$$

The second term on the right-hand side can be regarded as a correction term. The correction term tends to zero as convergence is approached. If this term is multiplied by ω, Eq. (D.22) becomes

$$x_i^{k+1} = x_i^k + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i}^{n} a_{ij} x_j^k \right], \qquad i = 1, 2, \ldots, n \quad \text{(D.23)}$$

The *relaxation factor* ω is selected such that $1 < \omega < 2$. The choice of a proper value of ω is problem dependent and is often determined by trial and error. The added weight of ω is intended to improve the estimate by pushing it closer to the exact value.

D.2.4 Gradient Methods

The iterative methods considered above may be broadly classified as *stationary* while the ones to be presented now are *gradient* (or nonstationary) methods. The two common gradient methods are the *steepest method* and *conjugate gradients method* [6]–[8]. A major advantage gradient methods have over stationary methods is that convergence is faster; hence gradient methods are particularly useful when the number of simultaneous equations is very large.

A set of n simultaneous equations may be solved by finding the position of the minimum of an error function defined over an n-dimensional space. In each step of a gradient method, a trial set of values for the variables is used to generate a new set corresponding to a lower value of the error function. If $\overline{X}$ is the trial vector, the vector residual is

$$R = B - A\overline{X} \tag{D.24}$$

where A is real, symmetric, and positive definite. If we define the error function as

$$e = R^t A^{-1} R \,, \tag{D.25}$$

then

$$e = \overline{X}^t A\overline{X} - 2B^t\overline{X} + B^t A^{-1} B \tag{D.26}$$

showing that e is quadratic in $\overline{X}$.

Starting from an arbitrary point X_o, we locate a sequence of points

$$X_{k+1} = X_k + \alpha_k D_k \tag{D.27}$$

which are successively closer to X, where X minimizes e in Eq. (D.26). The parameter α_k is proportional to the distance between X_i and X_{i+1} along the direction vector D_k. Substituting Eq. (D.27) into Eq. (D.26) and setting $\partial e/\partial\alpha_k$ equal to zero gives

$$\alpha_k = \frac{D_k^t R_k}{D_k^t A D_k} \tag{D.28}$$

Both the methods of steepest descent and conjugate gradients use Eq. (D.28) but differ in the choice of D_k.

In the method of descent, D_k is taken as the direction of maximum gradient of e at X_k. This direction is proportional to X_k so that the iterative algorithm has the form:

(i) select an arbitrary X_0

(ii) compute $R_0 = B - AX_0$

(iii) determine successively

$$\begin{aligned} U_k &= AR_k \\ \alpha_k &= \frac{R_k^t R_k}{R_k^t U_k} \\ X_{k+1} &= X_k + \alpha_k R_k \\ R_{k+1} &= R_k - \alpha_k U_k \end{aligned} \tag{D.29}$$

(iv) repeat step (iii) until residual vector $(R^T R)$ becomes sufficiently small.

In the method of conjugate gradients, D_k are selected as n vectors P_k which are mutually conjugate. The vectors P_k are conjugate or orthogonal to A, i.e.,

$$\begin{aligned} P_k^t A P_k &= 0, \qquad i \neq j \\ &\neq 0, \qquad i = j \end{aligned} \tag{D.30}$$

Thus the conjugate gradients algorithm is as follows:

(i) select an arbitrary X_0

(ii) set $P_0 = R_0 = B - AX_0$

(iii) determine successively

$$\begin{aligned} U_k &= AR_k \\ \alpha_k &= \frac{P_k^t R_k}{P_k^t U_k} \\ X_{k+1} &= X_k + \alpha_k R_k \\ R_{k+1} &= R_k - \alpha_k U_k \\ \beta_k &= -\frac{R_{k+1}^t U_k}{P_k^t U_k} \\ P_{k+1} &= R_k + \beta_k P_k \end{aligned} \tag{D.31}$$

(iv) repeat step (iii) until $k = n - 1$ or the residual vector $(R^T R)$ becomes sufficiently small.

This algorithm is guaranteed to yield the true solution in no more than n iterations—a condition known as *quadratic convergence*. Because of this, the conjugate gradients method has the advantage of an iterative scheme in that the roundoff error is limited only to the final step of the solution and also the advantage of a direct method in that it converges to the exact solution in a finite number of steps.

The subroutine in Fig. D.4 applies the conjugate gradients method to solve a given set of simultaneous equations. Typical areas where the conjugate gradient methods have been applied in EM can be found in [9]–[12].

```
0001    C*********************************************************
0002    C  THIS SUBROUTINE APPLIES THE CONJUGATE GRADIENTS METHOD
0003    C  TO SOLVE A SET OF SIMULTANEOUS EQUATIONS [A][X] = [B]
0004    C  IDM = DIMENSION OF [A] IN THE CALLING PROGRAM
0005    C    N = ACTUAL SIZE OF [A] IN THE CALLING PROGRAM
0006    C*********************************************************
0007
0008             SUBROUTINE CONJUGATE(A,B,N,IDM,X)
0009
0010            DIMENSION A(IDM,IDM), B(IDM), X(IDM)
0011            DIMENSION  U(100), R(100), P(100) !change 100
0012                                              !if N.GT.100
0013    C
0014    C   INITIALIZATION
0015    C
0016           K=0 !K=NO. OF ITERATIONS
0017           TOL = 1.0E-30 !TOLERANCE FOR ZERO
0018           DO 10 I=1,N
0019           X(I) = 0.0
0020           P(I) = B(I)
0021           R(I) = B(I)
0022    10     CONTINUE
0023    C
0024    C   ITERATION BEGINS HERE
0025    C
0026    20     K = K + 1
0027           DO 30 I=1,N
0028           U(I) = 0.0
0029           DO 30 J=1,N
0030           U(I) = U(I) + A(I,J)*P(J)
0031    30     CONTINUE
0032           SUM1 = 0.0
0033           SUM2 = 0.0
0034           DO 40 I=1,N
0035           SUM1 = SUM1 + P(I)*R(I)
0036           SUM2 = SUM2 + P(I)*U(I)
0037    40     CONTINUE
0038           ALPHA = SUM1/SUM2
0039           DO 50 I=1,N
0040           X(I) = X(I) + ALPHA*P(I)
0041    C      PRINT *,K,I,X(I)
0042           R(I) = R(I) - ALPHA*U(I)
0043    50     CONTINUE
0044    C
0045    C  CALCULATE RESIDUALS AND DIRECTION VECTORS
0046    C
0047           SUM3 = 0.0
0048           SUM4 = 0.0
0049           DO 60 I=1,N
0050           SUM3 = SUM3 + R(I)*R(I)
0051           SUM4 = SUM4 + R(I)*U(I)
0052    60     CONTINUE
0053    C
0054    C  CHECK IF RESIDUALS ARE ALREADY SMALL
0055    C  IF SO ALGORITHM HAS CONVERGED
0056    C
0057           IF( (K.EQ.N).OR.(SUM3.LT.TOL) ) GO TO 80
0058           BETA = - SUM4/SUM2
0059           DO 70 I=1,N
0060           P(I) = R(I) + BETA*P(I)
0061    70     CONTINUE
0062            GO TO 20
0063    80      RETURN
0064            END
```

Figure D.4

This subroutine applies the conjugate gradients method to solve $[A][X] = [B]$ *(Continued).*

D.3 Matrix Inversion

If $[A]$ is a square matrix, there is another matrix $[A]^{-1}$, called the *inverse* of $[A]$, such that

$$\big[A\big]\big[A\big]^{-1} = \big[A\big]^{-1}\big[A\big] = \big[I\big] \tag{D.32}$$

where I is the *identity* or *unit matrix*. Matrix inversion can be used to solve a set of simultaneous equations in Eq. (D.1) as

$$\big[X\big] = \big[A\big]^{-1}\big[B\big] \tag{D.33}$$

The solution of a system of simultaneous equations by matrix inversion and multiplication is most valuable when several systems are to be solved, all of which have the same coefficient matrix but different column matrices of constants. This situation requires calculating the inverse matrix only once and using it as a premultiplier of each of the column matrices of constants [2, 13].

The inversion of matrices is closely related to the solution of sets of simultaneous equations. The inverse of $[A]$ can be determined from Eq. (D.32). If we let $[C] = [A]^{-1}$, then

$$\big[A\big]\big[C\big] = \big[I\big] \tag{D.34a}$$

or

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & & & \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & & \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{D.34b}$$

This may be regarded as n sets of n simultaneous equations with identical coefficient matrix. The ith set of n simultaneous equations, for example, is

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{ni} & a_{ni} & \dots & a_{ni} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} c_{1i} \\ c_{2i} \\ \vdots \\ c_{ii} \\ \vdots \\ c_{ni} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \tag{D.35}$$

Thus, the inversion of $[A]$ may be accomplished by solving n sets of equations like Eq. (D.35). A common approach for matrix inversion is applying elimination method, with or without pivotal compensation. This implies that any elimination technique

(Gauss, Gauss-Jordan, or Cholesky's method) can be modified to calculate an inverse matrix. Here we apply the Gauss-Jordan elimination method.

To apply the Gauss-Jordan method, we first augment the coefficient matrix by the identity matrix to obtain

$$[A \vdots I] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & \vdots & 1 & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & a_{2n} & \vdots & 0 & 1 & \dots & 0 \\ \vdots & & & & \vdots & \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} & \vdots & 0 & 0 & \dots & 1 \end{bmatrix} \tag{D.36}$$

The goal is to transform this augmented matrix to another augmented matrix of the form

$$[I \vdots C] = \begin{bmatrix} 1 & 0 & \dots & 0 & \vdots & c_{11} & c_{12} & \dots & c_{1n} \\ 0 & 1 & \dots & 0 & \vdots & c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & & & & \vdots & \vdots & & & \\ 0 & 0 & \dots & 1 & \vdots & c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix} \tag{D.37}$$

where $[C]$ is the inverse of $[A]$. The transformation is achieved using the Gauss-Jordan method, which involves applying the following equations in the order listed [2]:

$$\begin{aligned} a'_{kj} &= a_{kj}/a_{kk}, \qquad j = 1, 2, \dots, n, \quad j \neq k \\ a'_{kk} &= 1/a_{kk}, \\ a'_{ij} &= a_{ij} - a_{ik}a'_{kj}, \qquad i = 1, 2, \dots, n, \quad i \neq k \\ &\qquad\qquad j = 1, 2, \dots, n, \quad j \neq k \\ a'_{ik} &= -a_{ik}a'_{kk}, \qquad i = 1, 2, \dots, n, \quad i \neq k \end{aligned} \tag{D.38}$$

We apply Eq. (D.38) for $k = 1, 2, \dots, n$. A computer program applying Eq. (D.38) is presented in Fig. D.5. An iterative method of correcting the elements of the inverse matrix is available in [14].

D.4 Eigenvalue Problems

The nature of these problems is discussed in Section 1.3. Here we are concerned with the so-called standard eigenproblems

$$[A - \lambda I][X] = 0 \tag{D.39}$$

```
0001    C*********************************************************
0002    C     SUBROUTINE FOR MATRIX INVERSION USING GAUSS-JORDAN
0003    C     ELIMINATION METHOD
0004    C     A IS THE MATRIX TO BE INVERTED; IT IS DESTROYED
0005    C     IN THE COMPUTATION AND REPLACED BY THE INVERSE
0006    C     N   = THE ORDER OF A
0007    C     IDM = THE DIMENSION OF A
0008    C
0009    C     REF: [2, p. 197 ]
0010    C*********************************************************
0011          SUBROUTINE INVERSE(A,N,IDM)
0012          DIMENSION A(IDM,IDM)
0013    C
0014    C CALCULATE  ELEMENTS OF REDUCED MATRIX
0015    C
0016           DO  60 K=1,N
0017    C
0018    C CALCULATE NEW ELEMENTS OF PIVOT ROW
0019    C
0020           DO 40 J=1,N
0021           IF(J.EQ.K) GO TO 40
0022           A(K,J) = A(K,J)/A(K,K)
0023    40     CONTINUE
0024    C
0025    C  CALCULATE ELEMENT REPLACING PIVOT ELEMENT
0026    C
0027           A(K,K) = 1.0/A(K,K)
0028    C
0029    C  CALCULATE NEW ELEMENTS NOT IN PIVOT ROW OR PIVOT COLUMN
0030    C
0031           DO 50 I=1,N
0032           IF(I.EQ.K) GO TO 50
0033           DO 50 J=1,N
0034           IF(J.EQ.K) GO TO 50
0035           A(I,J) = A(I,J) - A(I,K)*A(K,J)
0036    50     CONTINUE
0037    C
0038    C  CALCULATE REPLACEMENT ELEMENTS FOR PIVOT
0039    C  COLUMN-EXCEPT PIVOT ELEMENT
0040    C
0041           DO 60 I=1,N
0042           IF(I.EQ.K) GO TO 60
0043           A(I,K) = - A(I,K)*A(K,K)
0044    60     CONTINUE
0045            RETURN
0046            END
```

Figure D.5
Matrix inversion using Gauss-Jordan elimination method.

or the generalized eigenproblem

$$[A - \lambda B][X] = 0 \tag{D.40}$$

To show that Eqs. (D.39) and (D.40) are solved in the same way, we premultiply Eq. (D.40) by B^{-1} to obtain

$$[B^{-1}A - \lambda I][X] = 0 \tag{D.41}$$

Assuming $C = B^{-1}A$ gives

$$[C - \lambda I][X] = 0 \tag{D.42}$$

showing that Eq. (D.39) is a special case of Eq. (D.40) in which $B = I$. Thus, the procedure for solving Eq. (D.39) applies to Eq. (D.40) or (D.42).

The eigenvalue problems of Eqs. (D.39) and (D.40) are solved by either direct or indirect methods. In direct methods, such as Jacobi's method, the relevant matrix elements are stored in the computer, and an explicit procedure is used to obtain some or all of the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and eigenvalues $X_1, X_2, \ldots, X_n$. Indirect methods are basically iterative, and the matrix elements are usually generated rather than stored.

D.4.1 Iteration (or Power) Method

The most commonly used iterative method is the *power method.* The method is suitable in situations where either the greatest or the least eigenvalue is required. Suppose that one of the eigenvalues of A, say λ_1, satisfies the condition

$$|\lambda_1| > |\lambda_i|, \qquad i \neq 1, \tag{D.43}$$

then $|\lambda_1|$ is said to be the *dominant* eigenvalue of A. In many applications, the dominant eigenvalue is the most important and is probably the only eigenvalue we are interested in. The iteration method is specifically used for finding the dominant eigenvalues.

The iterative procedure is essentially based on the condition that should a trial vector $[X]_i$ be assumed, an approximate eigenvalue and a new trial eigenvector $[X]_{i+1}$ can be determined from Eq. (D.39) or Eq. (D.40). To find the largest eigenvalue $|\lambda_1|$, we rewrite Eq. (D.39) as

$$[A][X] = \lambda[X] \tag{D.44}$$

and follow these steps [2]:

(1) Assume a trial vector $[X]_0 = (x_1, x_2, \ldots, x_n)$, e.g., $[X]_0 = (1, 1, \ldots, 1)$. Substituting $[X]_0$ to the left-hand side of Eq. (D.44) gives the first approximation to the corresponding eigenvector, i.e.,

$$\lambda[X]_1 = (\lambda x_1, \lambda x_2, \ldots, \lambda x_n)$$

(2) Normalize the new vector $\lambda[X]$ by dividing it by the magnitude of its first component or by dividing the vector $[X]$ by the magnitude of the first component.

(3) Substitute the normalized vector into the left-hand side of Eq. (D.44) and obtain a new approximate eigenvector.

(4) Repeat steps (2) and (3) until the components of the new and previous eigenvectors differ by some prescribed tolerance. The normalizing factor will be the *largest eigenvalue* λ_1 while $[X]$ is the associated eigenvector.

To find the smallest eigenvalue, we first premultiply Eq. (D.44) by the inverse of $[A]$ to obtain

$$[X] = \lambda[A]^{-1}[X]$$

or

$$[A]^{-1}[X] = \frac{1}{\lambda}[X] \tag{D.45}$$

Thus the iteration formula becomes

$$[A]^{-1}[X]_i = \frac{1}{\lambda}[X]_{i+1} \tag{D.46}$$

In this form, the iteration converges to the largest value $1/\lambda$, which corresponds to the smallest eigenvalue λ of $[A]$.

Once the largest eigenvalue is found, the method can be used to obtain the next largest eigenvalue by transforming $[A]$ to another matrix possessing only the remaining eigenvalues [2]. This so-called *matrix deflation* procedure assumes that $[A]$ is symmetric. The matrix deflation is continued until all the eigenvalues have been extracted. Error propagation from one stage of the deflation to the next leads to inaccurate results, specially for large eigenproblems. Jacobi's method, to be discussed in the next section, is recommended for large eigenproblems.

The subroutine in Fig. D.6 is useful for finding the largest eigenvalues of a matrix.

D.4.2 Jacobi's Method

Jacobi's method is perhaps the most reliable method for solving eigenvalue problems. Its major advantage is that it finds all eigenvalues and eigenvectors simultaneously with excellent accuracy.

The method transforms a symmetric matrix $[A]$ into a diagonal matrix having the same eigenvalues as $[A]$. This is achieved by eliminating one pair of off-diagonal elements of $[A]$ at a time. Given

$$[A][X] = \lambda[X]\,, \tag{D.47}$$

let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues and $[V_1], [V_2], \ldots, [V_n]$ the corresponding eigenvectors. Then,

$$\begin{aligned} \big[A\big]\big[V_1\big] &= \lambda_1\big[V_1\big] \\ \big[A\big]\big[V_2\big] &= \lambda_2\big[V_2\big] \\ &\vdots \\ \big[A\big]\big[V_n\big] &= \lambda_1\big[V_n\big] \end{aligned} \tag{D.48}$$

or simply

$$[A][V] = [V][\lambda] \tag{D.49}$$

```
0001    C*****************************************************
0002    C    THIS SUBROUTINE FINDS THE LARGEST EIGENVALUE AND
0003    C    THE ASSOCIATED EIGENVECTOR OF
0004    C    A MATRIX EQUATION [A][X] = LAMBDA*[X]
0005    C    IDM  = THE DIMENSION OF MATRIX [A]
0006    C    N    = THE ACTUAL SIZE OF MATRIX [A]
0007    C    IT   = THE NO. OF ITERATIONS USED
0008    C    REF: [2, p. 238,239]
0009    C*****************************************************
0010          SUBROUTINE POWER(A,LAMBDA,X,IDM,N,IT)
0011          REAL LAMBDA
0012          PARAMETER (ISIZE=100) !INCREASE ISIZE IF IDM > 100
0013          DIMENSION  A(IDM,IDM),X(IDM)
0014          DIMENSION  D(ISIZE), Z(ISIZE)
0015
0016          EPSI = 1.0E-30  !TOLERANCE FOR ZERO
0017    C
0018    C  INITIALIZATION
0019    C
0020          DO 20 I=1,N
0021          X(I) = 1.0
0022    20    CONTINUE
0023    C
0024    C  CALCULATE [A][X] AND CALL IT [D]
0025    C
0026          IT = 0
0027    30    DO 40 I=1,N
0028          D(I) = 0.0
0029          DO 40 J=1,N
0030          D(I) = D(I) + A(I,J)*X(J)
0031    40    CONTINUE
0032          IT = IT + 1
0033    C
0034    C  NORMALIZE VECTOR [D] AND CALL IT [Z]
0035    C
0036          DO 50 I=1,N
0037          Z(I) = D(I)/D(1)
0038    50    CONTINUE
0039    C
0040    C  CHECK IF RESULT IS SUFFICIENTLY ACCURATE
0041    C
0042          DO 60 I=1,N
0043          DIFF = X(I) - Z(I)
0044          IF( ABS(DIFF) - ESPI*ABS(Z(I)) ) 60, 60, 70
0045    60    CONTINUE
0046          GO TO 90
0047    C
0048    C   SUFFICIENT ACCURACY HAS NOT BEEN ATTAINED
0049    C   LET  [X] = [Z] AND REPEAT THE PROCESS
0050    C
0051    70    DO 80 I=1,N
0052          X(I) = Z(I)
0053    80    CONTINUE
0054          IF(IT.GE.100) GO TO 110
0055          GO TO 30
```

Figure D.6

Subroutine for finding the largest eigenvalue of equation $[A][X] = \mathbf{LAMBDA}[X]$ ***(Continued).***

```
0056    C
0057    C  SUFFICIENT ACCURACY HAS BEEN ATTAINED
0058    C
0059    90      DO 100 I=1,N
0060            X(I) = Z(I)
0061    100     CONTINUE
0062    C
0063    C     OR MAXIMUM NUMBER OF ITERATIONS HAS BEEN REACHED
0064    C
0065    110     LAMBDA = D(1)
0066            RETURN
0067            END
```

Figure D.6
(Cont.) **Subroutine for finding the largest eigenvalue of equation** $[A][X] =$ **LAMBDA**$[X]$**.**

where

$$[V] = \left[\left[V_1\right], \left[V_2\right], \ldots, \left[V_n\right]\right] \tag{D.50a}$$

$$[\lambda] = \begin{bmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & \lambda_n \end{bmatrix} \tag{D.50b}$$

From the theory of matrices, if $[A]$ is symmetric, $[V]$ is orthogonal, i.e.,

$$[V]^t = [V]^{-1} \tag{D.51}$$

hence, premultiplying Eq. (D.49) by $[V]^t$ leads to

$$[V]^t[A][V] = [\lambda] \tag{D.52}$$

signifying that the eigenvalues of $[V]^t[A][V]$, which is known as the orthogonal transformation of $[A]$, are the same as those of $[A]$. Thus the problem of finding the eigenvalues is reduced to finding the $[V]$ matrix.

The $[V]$ matrix is constructed iteratively by using unitary matrix (or plane rotation matrix) $[R]$. If we let

$$\begin{aligned} \left[A_1\right] &= \left[A\right] \\ \left[A_2\right] &= \left[R_1\right]^t\left[A_1\right]\left[R_1\right] \\ \left[A_3\right] &= \left[R_2\right]^t\left[A_2\right]\left[R_2\right] = \left[R_2\right]^t\left[R_1\right]^t\left[A\right]\left[R_1\right]\left[R_2\right] \\ &\vdots \\ \left[A_k\right] &= \left[R_{k-1}\right]^t \ldots \left[R_1\right]^t\left[A\right]\left[R_1\right]\ldots\left[R_{k-1}\right], \end{aligned} \tag{D.53}$$

then as $k \to \infty$

$$\begin{aligned} \left[A_k\right] &\to \left[\lambda\right] \\ \left[R_1\right]\left[R_2\right]\ldots\left[R_{k-1}\right] &\to \left[V\right] \end{aligned} \tag{D.54}$$

The unitary transformation matrix $[R]$ eliminates the pair of equal elements a_{pq} and a_{qp}. It is given by [1, 2, 7]

$$\begin{matrix} & & p & & q & \\ \end{matrix}$$

$$\left[R_k\right] = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \cos\theta & -\sin\theta & & p \\ & & & 1 & & \\ & & \sin\theta & \cos\theta & & q \\ & & & & 1 & \end{bmatrix} \tag{D.55a}$$

i.e.,

$$\begin{aligned} R_{qq} &= R_{pp} = \cos\theta \\ -R_{pq} &= R_{qp} = \sin\theta \\ R_{ii} &= 1, \quad i \neq p, q \\ R_{ij} &= 0, \quad \text{elsewhere} \end{aligned} \tag{D.55b}$$

The choice of θ in the transformation matrix must be such that new elements $a'_{pq} = a'_{qp} = 0$, i.e.,

$$a'_{pq} = \left(-a_{pp} + a_{qq}\right)\cos\theta\sin\theta + a_{pq}\left(\cos^2\theta - \sin^2\theta\right) = 0 \tag{D.56}$$

Hence

$$\tan 2\theta = \frac{2a_{pq}}{a_{pp} - a_{qq}}, \qquad -45^\circ < \theta < 45^\circ \tag{D.57}$$

An alternative manipulation of Eq. (D.56) gives

$$\cos\theta = \left[\frac{\sqrt{(a_{pp} - a_{qq})^2 + 4a_{pq}^2} + (a_{pp} - a_{qq})}{2\sqrt{(a_{pp} - a_{qq})^2 + 4a_{pq}^2}}\right]^{1/2} \tag{D.58a}$$

$$\sin\theta = \frac{a_{pq}}{\sqrt{(a_{pp} - a_{qq})^2 + 4a_{pq}^2}\cos\theta} \tag{D.58b}$$

Notice that Eq. (D.53) requires an infinite number of transformations because the elimination of elements a_{pq} and a_{qp} in one step will in general undo the elimination of previously treated elements in the same row or column. However, the transformation converges rapidly and ceases when all the off-diagonal elements become negligible in magnitude.

The program in Fig. D.7 determines all the eigenvalues and eigenvectors of symmetric matrices employing Jacobi's method.

```
0001    C*****************************************************
0002    C    USING JACOBI METHOD,
0003    C    THIS SUBROUTINE FINDS ALL THE EIGENVALUES AND
0004    C    THE ASSOCIATED EIGENVECTORS OF
0005    C    A MATRIX EQUATION [A][X] = LAMBDA*[X]
0006    C    A     = SYMMETRIC MATRIX
0007    C    LAMBDA(J) = EIGENVALUES, ORDERED FROM ALGEBRAICALLY
0008    C                LARGEST TO SMALLEST
0009    C    RT(I,J)   = MATRIX THAT WILL EVENTUALLY CONTAIN THE
0010    C                EIGENVECTORS (J) ASSOCIATED WITH LAMBDA (J)
0011    C    IDM   = THE DIMENSION OF MATRIX [A]
0012    C    N     = THE ACTUAL SIZE OF MATRIX [A]
0013    C    IT    = THE NO. OF ITERATIONS USED
0014    C    REF: [2, p. 268-271]
0015    C*****************************************************
0016          SUBROUTINE EIGEN(A,RT,IDM,N,LAMBDA)
0017          DIMENSION  A(IDM,IDM),RT(IDM,IDM)
0018          REAL LAMBDA(IDM)
0019    C
0020    C   GENERATE AN N X N IDENTITY MATRIX RT WHICH WILL
0021    C   EVENTUALLY CONTAIN THE EIGENVECTORS
0022    C
0023          DO 20 I=1,N
0024          DO 10 J=1,N
0025          RT(I,J) = 0.0
0026    10    CONTINUE
0027          RT(I,I) = 1.0
0028    20    CONTINUE
0029          NSWEEP = 0
0030    30    NRSKIP = 0
0031    C
0032    C   BEGIN A SWEEP WHICH WILL TRANSFORM EACH OFF-DIAGONAL
0033    C   ELEMENT IN TURN TO ZERO
0034    C
0035          NMIN1 = N-1
0036          DO 130 I=1,NMIN1
0037          IP1 = I + 1
0038          DO 120 J=IP1.N
0039          AV = 0.5*( A(I,J) + A(J,I) )
0040          DIFF = A(I,I) - A(J,J)
0041          RAD = SQRT( DIFF*DIFF + 4.0*AV*AV )
0042    C
0043    C   CHECK IF RAD IS ZERO.  IF SO, NO NEED OF
0044    C   ROTATION
0045    C
0046          IF(RAD.EQ.0.0) GO TO 80
0047    C
0048    C  CHECK IF DIFF IS NEGATIVE.  IF SO, INTERCHANGE
0049    C  A(I,I) AND A(J,J) AND PERFORM ROTATION
0050    C
0051          IF(DIFF.LT.0.0) GO TO 60
0052          IF( ABS(A(I,I)).EQ.ABS(A(I,I))+100.*ABS(AV) ) GO TO 40
0053          GO TO 40
0054    40    IF( ABS(A(J,J)).EQ.ABS(A(J,J))+100.*ABS(AV) ) GO TO 80
0055    50    COSINE = SQRT( (RAD + DIFF)/(2.0*RAD) )
0056          SINE = AV/(RAD*COSINE)
0057          GO TO 70
0058    C
0059    C  FOR THE DIAGONAL ELEMENTS, PERFORM ROTATION
0060    C
0061    60    SINE = SQRT( (RAD-DIFF)/(2.0*RAD) )
0062          IF(AV.LT.0.0) SINE = -SINE
0063          COSINE = AV/(RAD*SINE)
```

Figure D.7

Subroutine for finding all the eigenvalues and eigenvectors of equation $[A][X] =$ LAMBDA$[x]$ ***(Continued.)***

```
0064    C
0065    C  CHECK IF SIN(THETA) IS NEGLIGIBLE
0066    C  IF SO, SKIP ROTATION
0067    C
0068    70      IF(1.0.LT.1.0 + ABS(SINE)) GO TO 90
0069    80      NRSKIP = NRSKIP + 1
0070            GO TO 120
0071    C
0072    C  FERFORM ROTATION
0073    C  PREMULTIPLY BY THE ROTATION MATRIX
0074    C
0075    90      DO 100 K=1,N
0076            Q = A(I,K)
0077            A(I,K) = COSINE*Q + SINE*A(J,K)
0078            A(J,K) = - SINE*Q + COSINE*A(J,K)
0079    100     CONTINUE
0080    C
0081    C  POSTMULTIPLY BY THE TRANSFORM OF THE ROTATION MATRIX
0082    C
0083            DO 110 K=1,N
0084            Q = A(K,I)
0085            A(K,I) = COSINE*Q + SINE*A(K,J)
0086            A(K,J) = - SINE*Q + COSINE*A(K,J)
0087    C
0088    C   POSTMULTIPLY THE CURRENT PRODUCT OF ALL THE RT MATRICES
0089    C   UP TO THIS POINT BY THE CURRENT RT MATRIX
0090    C
0091            Q = RT(K,I)
0092            RT(K,I) = COSINE*Q + SINE*RT(K,J)
0093            RT(K,J) = - SINE*Q + COSINE*RT(K,J)
0094    110     CONTINUE
0095    120     CONTINUE
0096    130     CONTINUE
0097    C
0098    C  KEEP A TALLY OF THE NUMBER OF SWEEPS
0099    C
0100            NSWEEP = NSWEEP + 1
0101            IF(NSWEEP.GT.50) GO TO 140
0102    C       WRITE(6,*)NRSKIP,NSWEEP
0103    C
0104    C  CHECKIF THE NUMBER OF ROTATIONS SKIPPED IS LESS/
0105    C  EQUAL TO THE NO. OF ELEMENTS ABOVE THE MAIN DIAGONAL
0106    C  IF EQUAL, CONVERGENCE HAS OCCURRED
0107    C
0108            IF(NRSKIP.LT.N*(N-1)/2) GO TO 30
0109    140     CONTINUE
0110    C       WRITE(6,*) NSWEEP
0111            DO 150 J=1,N
0112            LAMBDA(J) = A(J,J)
0113    150     CONTINUE
0114            RETURN
0115            END
```

Figure D.7

(Cont.) **Subroutine for finding all the eigenvalues and eigenvectors of equation** $[A][X] =$ **LAMBDA**$[x]$**.**

References

[1] A.W. Al-Khafaji and J.R. Tooley, *Numerical Methods in Engineering Practice*. New York: Rinehart and Winston, 1986, pp. 84–159, 203–270.

[2] M.L. James et al., *Applied Numerical Methods for Digital Computation*. 3rd ed., New York: Harper & Row, 1985, pp. 146–298.

[3] S.A. Hovanessian and L.A. Pipes, *Digital Computer Methods in Engineering*. New York: McGraw-Hill, 1969, pp. 1–48.

[4] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing,* 2nd ed., Monterey, CA: Brooks/Cole, 1985, pp. 201–257.

[5] R.L. Ketter and S.P. Prawel, *Modern Methods of Engineering Computation*. New York: MacGraw-Hill, 1969, pp. 66–117.

[6] A. Ralston and H.S. Wilf (eds.), *Mathematical Methods for Digital Computers*. New York: John Wiley, 1960, pp. 62–72.

[7] A. Jennings, *Matrix Computation for Engineers and Scientists*. New York: John Wiley, 1977, pp. 182–222, 250–254.

[8] J.C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimization*. New York: John Wiley, 1979, pp. 195–199.

[9] T.K. Sarkar, et al., "A limited survey of various conjugate gradient methods for solving complex matrix equations arising in electromagnetic wave interaction," Wave Motion, vol. 10, no. 6, 1988, pp. 527–546.

[10] A.F. Peterson and R. Mittra, "Method of conjugate gradients for the numerical solution of large-body electromagnetic scattering problems," *J. Opt. Soc. Am.,* Pt. A, vol. 2, no. 6, June 1985, pp. 971–977.

[11] T.K. Sarkar, "Application of the Fast Fourier transform and the conjugate gradient method for efficient solution of electromagnetic scattering from both electrically large and small conducting bodies," *Electromagnetics,* vol. 5, 1985, pp. 99–122.

[12] D.T. Borup and O.P. Gandhi, "Calculation of high-resolution SAR distributions in biological bodies using the FFT algorithm and conjugate gradient method," *IEEE Trans. Micro. Theo. Tech.,* vol. MTT-33, no. 5, May 1985, pp. 417–419.

[13] R.W. Southworth and S.L. Deleeuw, *Digital Computation and Numerical Methods*. New York: MacGraw-Hill, 1965, pp. 247–251.

[14] S. Hovanessian, *Computational Mathematics in Engineering*. Lexington, MA: Lexington Books, 1976, p. 25.