# 4

# Algorithms for the Solution of Linear Systems of Equations

The procedures of the previous two chapters produce a linear system of equations that must be solved to determine the unknown coefficients of the basis functions. In this chapter we discuss some of the algorithms in widespread use for solving matrix equations. Initially, we review direct methods for the solution of general full- and sparse-matrix equations. The error introduced by finite precision arithmetic is explored. One iterative algorithm, the conjugate gradient method, is considered in detail. The conjugate gradient–fast Fourier transform implementation is discussed for the solution of certain specialized integral equations. Finally, we briefly introduce the fast multipole method, which offers similar computational advantages for the iterative solution of general problems.

## 4.1 NAIVE GAUSSIAN ELIMINATION

Consider the general matrix equation $\mathbf{Ax} = \mathbf{b}$, or

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \ddots & \ddots \\ a_{31} & a_{32} & a_{33} & & \\ \cdot & \ddots & \ddots & & \\ \cdot & & & & \\ \cdot & & & & \\ a_{N1} & \cdots & \cdots & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdot \\ \cdot \\ \cdot \\ b_N \end{bmatrix} \tag{4.1}$$

A solution to this system of equations may be found using one of the various forms of Gaussian elimination. The procedure we describe requires the row equations of the system

to be systematically replaced with linear combinations of other rows of the matrix in order to recast Equation (4.1) into the form of an upper triangular system:

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & 1 & u_{23} & \ddots & \ddots \\ 0 & 0 & 1 & & \\ \cdot & \cdot & & & \\ \cdot & \cdot & & & \\ \cdot & \cdot & & & \\ 0 & 0 & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \cdot \\ \cdot \\ \cdot \\ c_N \end{bmatrix} \tag{4.2}$$

Once the system of equations is converted to the form of Equation (4.2), the solution can be obtained by back substitution, following the recipe

$$x_N = c_N \tag{4.3}$$

$$x_m = c_m - \sum_{i=m+1}^{N} u_{mi} x_i \qquad m = N-1,\ N-2, \ldots, 1 \tag{4.4}$$

To systematically convert Equation (4.1) into the form of (4.2), consider the first row of the system

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = b_1 \tag{4.5}$$

This equation may be divided by $a_{11}$ to produce a new first equation

$$x_1 + u_{12}x_2 + u_{13}x_3 + \cdots + u_{1N}x_N = c_1 \tag{4.6}$$

The new equation is to be employed in "sweeping out" the first column of the matrix. The operation proceeds as follows. After multiplication by $a_{21}$, the new equation can be subtracted from the second row of the system, altering the matrix equation to the equivalent form

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & a_{22} - a_{21}u_{12} & a_{23} - a_{21}u_{13} & \cdots & a_{2N} - a_{21}u_{1N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} c_1 \\ b_2 - a_{21}c_1 \\ b_3 \\ \cdot \\ \cdot \\ \cdot \\ b_N \end{bmatrix} \tag{4.7}$$

We have "eliminated" the (2, 1) entry from the matrix. This procedure is repeated until the first column of the matrix has been completely swept out, leaving an equivalent linear system

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & a_{22} - a_{21}u_{12} & a_{23} - a_{21}u_{13} & \cdots & a_{2N} - a_{21}u_{1N} \\ 0 & a_{32} - a_{31}u_{12} & a_{33} - a_{31}u_{13} & \cdots & a_{3N} - a_{31}u_{1N} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 0 & a_{N2} - a_{N1}u_{12} & a_{N3} - a_{N1}u_{13} & \cdots & a_{NN} - a_{N1}u_{1N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} c_1 \\ b_2 - a_{21}c_1 \\ b_3 - a_{31}c_1 \\ \cdot \\ \cdot \\ \cdot \\ b_N - a_{N1}c_1 \end{bmatrix} \tag{4.8}$$

For notational convenience, we divide the second equation by $a_{22} - a_{21}u_{12}$ to obtain

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & 1 & u_{23} & \cdots & u_{2N} \\ 0 & a_{32} - a_{31}u_{12} & a_{33} - a_{31}u_{13} & \cdots & a_{3N} - a_{31}u_{1N} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 0 & a_{N2} - a_{N1}u_{12} & a_{N3} - a_{N1}u_{13} & \cdots & a_{NN} - a_{N1}u_{1N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ b_3 - a_{31}c_1 \\ \cdot \\ \cdot \\ \cdot \\ b_N - a_{N1}c_1 \end{bmatrix} \tag{4.9}$$

The new second equation is used to sweep out the second column of the matrix. We continue this procedure throughout the remaining columns of the matrix, finally producing an upper triangular system having the form of Equation (4.2).

Numerous variations exist on the basic algorithm outlined in the preceding paragraph, and the reader is referred to the literature where Gaussian elimination is studied in detail [1–9]. In practice, the procedure discussed above has a drawback: The operations performed on the right-hand side **b** must be remembered if additional systems involving the same matrix **A** are to be treated. (This need arises in the solution of scattering problems involving more than one incident field; refer to Chapters 2 and 3.) This difficulty can be remedied by observing that Gaussian elimination is equivalent to factorizing the matrix **A** into the product of two matrices **L** and **U** according to

$$\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & \ddots & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ l_{N1} & l_{N2} & l_{N3} & \cdots & l_{NN} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1N} \\ 0 & 1 & u_{23} & \ddots & \ddots \\ 0 & 0 & 1 & & \\ \cdot & \cdot & & & \\ \cdot & \cdot & & & \\ \cdot & \cdot & & & \\ 0 & 0 & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdot \\ \cdot \\ \cdot \\ b_N \end{bmatrix} \tag{4.10}$$

The two triangular systems can be obtained with the same number of arithmetic operations as the procedure described above and stored in place of the original matrix as they are constructed. Once **L** and **U** are obtained, the vector $\mathbf{c} = \mathbf{L}^{-1}\mathbf{b}$ can be computed for each right-hand side **b** by forward substitution, yielding an equivalent system having the form of Equation (4.2).

The factorization of **A** into the LU form of Equation (4.10) requires approximately $N^3/3$ multiplications and additions. After factorization, the additional forward and backward substitution involves $N^2$ multiplications and additions for each right-hand side. (The identical effort would be required to produce each solution using the inverse $\mathbf{A}^{-1}$, whose computation requires many more operations than the factorization. Consequently, the factorization can be thought of as an implicit inverse matrix.) For large $N$ the factorization accounts for the bulk of the computational effort. In the context of the electromagnetic scattering applications discussed in Chapters 2 and 3, only a single factorization is needed to characterize a scatterer illuminated by a number of different incident fields.

There are numerous forms of LU factorization described in the literature, including the Gauss, Crout, Doolittle, and bifactorization algorithms [1–9]. These procedures require a similar number of arithmetic operations and differ little from each other in overall performance on traditional serial-architecture computers. However, these algorithms differ substantially in the order in which matrix entries are accessed from computer memory and in the number of times certain entries are altered. Consequently, one algorithm may

give substantially better performance than another on a particular machine, especially in a distributed-memory environment. The recently introduced LAPACK library [10] for linear equations and eigenvalue problems was designed for efficient data handling on modern high-performance computers.

## 4.2 PIVOTING

An obvious difficulty arises in the implementation of the elimination algorithm if the $i$th diagonal entry is zero at the start of sweeping out the $i$th column. The algorithm will fail attempting to divide by zero. This potential problem is readily circumvented by the use of pivoting. Partial row pivoting is the exchange of two rows of the system in order to replace one diagonal entry with a "stronger entry" (i.e., one that is greater in magnitude than the original entry). Row pivoting requires the algorithm to search for the best pivot among the $i$th column of the rows appearing below row $i$ in the matrix. In the simplest scheme, the row having the strongest entry in column $i$ is exchanged with the original row $i$ before sweeping out column $i$ of the matrix. This procedure, if repeated for each column of the matrix, ensures that the elimination algorithm will never require a division by zero unless the original system of equations is singular.

Although any form of pivoting will prevent a catastrophic division by zero, alternative pivoting strategies are sometimes sought to minimize the growth of round-off errors. Even if all the pivots are nonzero, large errors can arise during elimination in finite machine precision from the combined effects of division by small pivots followed by the required subtractions. A variety of different pivoting schemes are in use. For instance, scaling can be used to alter the manner in which the best pivots are selected [4, 5]. Column pivoting is possible, as is complete pivoting (complete pivoting selects the best pivot available in any of the remaining columns of the remaining rows of the system) [3, 4]. It is noteworthy that the additional computation associated with complete pivoting is usually not offset by improved accuracy over partial pivoting. General-purpose LU factorization codes, such as those of the LINPACK [6] or LAPACK [10] libraries, always provide some form of pivoting to ensure stability.

By rearranging operations, pivoting can provide stability in order to control unnecessary numerical error during elimination in finite precision. However, it cannot compensate for some of the errors that may arise. In order to gain an appreciation for the fundamental limits associated with the solution of linear systems, the following section discusses the propagation of error in more detail.

## 4.3 CONDITION NUMBERS AND ERROR PROPAGATION IN THE SOLUTION OF LINEAR SYSTEMS

Numerical errors arise during factorization due to the finite precision of the machine and the limited accuracy of the various quantities. The *matrix condition number* is a useful concept for treating error propagation. The condition number of a general, $N$th-order

complex-valued matrix $\mathbf{A}$ can be defined as

$$\kappa(\mathbf{A}) = \sqrt{\frac{\lambda_{\max}}{\lambda_{\min}}} \tag{4.11}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ denote the smallest and largest eigenvalues, respectively, of the matrix $\mathbf{A}^\dagger\mathbf{A}$, where $\mathbf{A}^\dagger$ is the transpose conjugate of $\mathbf{A}$. A more general definition of the condition number is given by

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\| \tag{4.12}$$

where the matrix norm is

$$\|\mathbf{A}\| = \sup_{x \neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \tag{4.13}$$

and a variety of vector norms are possible. Use of the Euclidean 2-norm

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^\dagger\mathbf{x}} \tag{4.14}$$

is common, as is the use of the infinity norm. The actual condition number varies with the definition of the norm.

The condition number is a measure of the stability of the linear system of equations. A singular matrix, containing at least one degenerate row, has at least one zero eigenvalue and an infinite condition number. The identity matrix, with every eigenvalue equal to 1, has a condition number of 1. If the condition number is much greater than unity, we say that the matrix is *ill-conditioned.* The greater the condition number of a linear system, the more sensitive the equations to slight perturbations and the more numerical error likely to appear in the solution.

Since the matrix equations generated using the procedures of Chapters 2 and 3 can occasionally be poorly conditioned, we want to consider the effects of ill-conditioning in more detail. Because of numerical errors, the computed solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ will differ from the desired $\mathbf{x}$ by an amount $\Delta\mathbf{x}$. Two distinct sources of solution error are (1) error due to approximations in the construction of the entries of $\mathbf{A}$ and $\mathbf{b}$ or the truncation of the entries of $\mathbf{A}$ or $\mathbf{b}$ due to finite machine precision and (2) additional round-off and truncation error introduced by the Gaussian elimination process.

The effect of errors in the entries of $\mathbf{A}$ or $\mathbf{b}$ can be estimated by considering the perturbed equation

$$(\mathbf{A} + \Delta\mathbf{A})(\mathbf{x} + \Delta\mathbf{x}) = (\mathbf{b} + \Delta\mathbf{b}) \tag{4.15}$$

To estimate bounds on the error $\Delta\mathbf{x}$, consider Equation (4.15) under the condition when $\Delta\mathbf{A} = \mathbf{0}$. The Schwartz inequality can be applied to $\mathbf{A}\mathbf{x} = \mathbf{b}$ to produce

$$\|\mathbf{b}\| \leq \|\mathbf{A}\|\|\mathbf{x}\| \tag{4.16}$$

The combination of Equation (4.15) with $\Delta\mathbf{A} = \mathbf{0}$ and the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ yields the inequality

$$\|\Delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\|\|\Delta\mathbf{b}\| \tag{4.17}$$

The preceding two inequalities can be combined to produce

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(\mathbf{A})\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \tag{4.18}$$

Similar inequalities constructed under the assumption that $\Delta\mathbf{b} = \mathbf{0}$ can be combined to yield

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x} + \Delta\mathbf{x}\|} \leq \kappa(\mathbf{A})\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \tag{4.19}$$

Equations (4.18) and (4.19) provide upper bounds on the error in the solution $\mathbf{x}$ due to errors in the entries of $\mathbf{A}$ or $\mathbf{b}$. Note the presence of the condition number $\kappa(\mathbf{A})$ in these inequalities. Since the condition number is always greater than 1 in practice, the solution error $\Delta\mathbf{x}$ can be significantly larger than the errors in the individual matrix entries. In a sense, the error originally present in the matrix entries can be amplified in proportion to the condition number.

The effect of machine precision on the entries of $\mathbf{A}$ and $\mathbf{b}$ can be estimated using Equations (4.18) and (4.19) with $\Delta\mathbf{A}$ and $\Delta\mathbf{b}$ set to the value associated with the machine word length. For instance, on a machine with 32-bit (24-bit mantissa, 8-bit exponent) binary words, this truncation error $\varepsilon$ is on the order of $2^{-24}$. Of course, unless the entries of $\mathbf{A}$ and $\mathbf{b}$ are accurate to this order, the additional error due to machine truncation will be negligible compared to the uncertainty already present.

In summary, the finite accuracy of the entries in the matrix and the right-hand side can produce an error $\Delta\mathbf{x}$ in the solution that is proportional to the error in $\mathbf{A}$ and $\mathbf{b}$ amplified by the condition number $\kappa(\mathbf{A})$. Since this error is present in the original system, it cannot be reduced by subsequent operations. It can only be reduced by a more accurate evaluation of the matrix entries and, if necessary, by increased machine precision. In addition, this error is completely independent of additional errors introduced by the solution process.

The additional numerical error introduced during the factorization procedure was originally analyzed by Wilkinson [11] and is discussed in several recent references [3, 4, 7, 12]. Wilkinson's backward-error analysis postulated that the actual solution $(\mathbf{x} + \Delta\mathbf{x})$ exactly satisfies a perturbed equation

$$(\mathbf{A} + \Delta\mathbf{A})(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{b} \tag{4.20}$$

A bound on the equivalent error $\Delta\mathbf{A}$ can be obtained in terms of the infinity norm and has the form [3, 4, 7–10, 12]

$$\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} \leq \alpha\rho\varepsilon + O(\varepsilon^2) \tag{4.21}$$

where $\varepsilon$ is the machine unit round-off [approximately $2^{-24}$ on a 32-bit machine employing the Institute of Electrical and Electronics Engineers (IEEE) arithmetic], $\alpha$ is approximately $N^3$ (where $N$ is the order of $\mathbf{A}$), and $\rho$ is Wilkinson's growth factor. The growth factor is proportional to the largest number arising throughout the elimination process and is difficult to estimate. However, for general matrices factorized with partial pivoting, the growth factor is bounded by

$$\rho < 2^{N-1} \tag{4.22}$$

Combining these estimates with Equation (4.19) produces the approximate solution error bound

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x} + \Delta\mathbf{x}\|} \leq \kappa(\mathbf{A})N^3 2^{N-1}\varepsilon \tag{4.23}$$

It is necessary to employ a variety of assumptions in order to obtain a simple expression for the error bound in Equation (4.21). These include the assumption that $N\varepsilon < 0.1$, the

assumption that every entry of the matrix is nonzero, and the worst-case assumption that each matrix element contributes to the error [11, 12]. The resulting expression in Equation (4.23) is a very loose bound but does indicate that the expected error is proportional to the condition number and can be improved by increasing the machine precision. Unfortunately, the bound is almost useless for estimating the error as a function of $N$, since authorities appear to agree (!) that in practice the dependence on $N^3$ is extremely pessimistic and that the growth factor is almost never as bad as $2^{N-1}$ for actual applications (although matrices can be constructed for which $\rho$ does grow as $2^{N-1}$) [3, 4]. Because the bound in (4.23) is so loose, the additional error introduced by the elimination process is usually estimated as being comparable to the error introduced by the truncation of the matrix entries. This error estimate is embodied in the often-quoted rule of thumb: *Gaussian elimination produces a solution that has about*

$$t \log_{10} \beta - \log_{10} \kappa(\mathbf{A}) \tag{4.24}$$

*correct decimal places*, where $\beta$ is the base ($\beta = 2$ for binary operations) and $t$ is the bit length of the mantissa ($t = 24$ for a typical 32-bit binary word). This is equivalent to saying that the solution will lose about $\log_{10}\kappa$ digits of accuracy during elimination.

In summary, there are two independent sources of error in the solution of a matrix equation. The first is introduced by the inaccuracy of the matrix entries, the second by the elimination process. Although the latter could easily dominate for large-order systems, it is difficult to estimate and can only be controlled by increasing the precision. Practical experience suggests that the primary source of error is likely to be due to inaccuracies in the entries of $\mathbf{A}$ and $\mathbf{b}$, assuming that the entries are far less accurate than machine precision would allow. Since these inaccuracies may produce an error $\kappa(\mathbf{A})$ times as large in the solution, it is important to ensure the accurate evaluation of the matrix entries to more digits than $\log_{10}\kappa$. However, the matrix entries arising from the integral equation formulations of Chapter 2 were seldom available in exact, closed-form expressions. In practice, these entries are usually computed numerically or approximated by some convenient expression. (The finite-element modeling described in Chapter 3 permitted an accurate closed-form evaluation of some of the matrix entries.) In addition, both the integral and differential equation formulations required scatterer models that were, at best, only approximations to the desired geometry. These inherent approximations place a fundamental limit on the accuracy possible in entries of the matrix equation. In order to monitor the likely presence of error in the numerical solution, the condition number is usually estimated during elimination. For example, the LINPACK routine CGECO performs an LU factorization and estimates the condition number of a complex matrix [6]. Based upon the known accuracy of the matrix entries, the condition number, and the machine precision, a decision can be made as to the likely accuracy of the numerical result.

## 4.4 CHOLESKY DECOMPOSITION FOR COMPLEX-SYMMETRIC SYSTEMS

Often, the matrix equations arising from the formulations of Chapter 2 and 3 are complex symmetric in nature. For symmetric systems, an alternative factorization known as Cholesky decomposition can exploit symmetry to reduce the required computer storage

and the required computation by a factor of 2 over that required by general-purpose LU factorization routines.

Naive Cholesky decomposition seeks a factorization in the form

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T \tag{4.25}$$

where $\mathbf{L}^T$ is the transpose of the matrix $\mathbf{L}$, which is lower triangular. (Note that the transpose is different from the transpose conjugate.) It is readily verified that such a factorization is only possible if $\mathbf{A}$ is symmetric, that is,

$$\mathbf{A} = \mathbf{A}^T \tag{4.26}$$

The entries of $\mathbf{L}$ may be found by expanding (4.25), producing the formulas

$$l_{11} = \sqrt{a_{11}} \tag{4.27}$$

$$l_{n1} = \frac{a_{n1}}{l_{11}} \qquad n = 2, 3, \ldots, N \tag{4.28}$$

$$l_{22} = \sqrt{a_{22} - (l_{21})^2} \tag{4.29}$$

$$l_{n2} = \frac{a_{n2} - l_{21}l_{n1}}{l_{22}} \qquad n = 3, 4, \ldots, N \tag{4.30}$$

and so on. Since the operations are performed with complex arithmetic, the square root of a number having a negative real part is not a difficulty.

Although it is generally not possible to incorporate complete or partial pivoting into the Cholesky factorization procedure, alternative pivoting schemes have been proposed [13–16]. One approach, the diagonal pivoting method, is incorporated into the LINPACK library [16].

The advantage of the Cholesky algorithm is that the factorization requires approximately $N^3/6$ multiplications and additions, half of that necessary for LU factorization. Because of symmetry, it is only necessary to compute half of the matrix $\mathbf{A}$. The lower triangular matrix $\mathbf{L}$ is stored in the same space originally used by $\mathbf{A}$. As an additional hedge against rounding errors, the summations appearing in Equations (4.29), (4.30), and the rest that follow can be accumulated in double precision.

## 4.5 REORDERING ALGORITHMS FOR SPARSE SYSTEMS OF EQUATIONS

The differential equation formulations discussed in Chapter 3 produce matrix equations that are *sparse* (most of the entries are zero). It is obviously not economical to employ full-storage Gaussian elimination with sparse systems, and we turn our attention to special-purpose methods that exploit the matrix sparsity. Unfortunately, the LU factorization is usually not as sparse as the original system. *Fill-in* occurs whenever entries that were zero in the original matrix are replaced by nonzero entries in the factorization. However, a property of Gaussian elimination ensures that, in the absence of pivoting, all entries located to the left of the first nonzero entry in a given row of the matrix remain

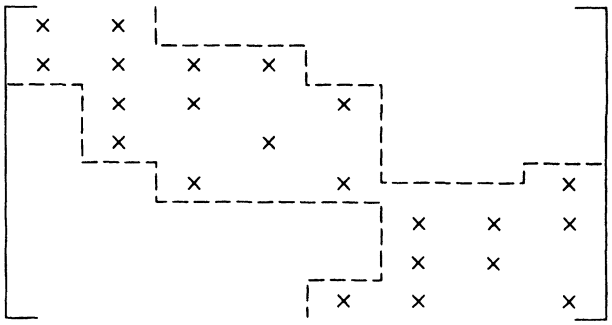


**Figure 4.1** Envelope of a matrix.

zero after factorization [2, 7, 17, 18]. This property can be exploited to provide a simple way of reducing the necessary storage to the *envelope* containing the nonzero entries (Figure 4.1).

One way of attempting to optimize the storage and computation associated with factorization algorithms is to minimize the size of the envelope. The actual sparsity pattern depends on the ordering of the matrix rows and columns, which in turn depends on the numbering scheme used within the finite-element mesh. Thus, it is possible in principle to optimize the numbering scheme used to construct the scatterer model in order to enhance the efficiency of the matrix solution algorithm. A variety of automatic reordering algorithms have been proposed for this purpose.

Chapter 3 considered the first-order finite-element discretization of the scalar Helmholtz equation. A two-dimensional model and the associated matrix sparsity pattern are depicted in Figure 3.11. The specific sparsity pattern is a direct result of the scheme used to assign numbers to different nodes of the triangular-cell mesh representing the cylinder cross section. Because adjacent nodes in Figure 3.11 are occasionally assigned widely differing numbers, there are nonzero entries of the matrix very far from the main diagonal.

Figure 4.2 depicts an identical model employing a numbering scheme that attempts to minimize the effective *bandwidth* of the matrix (the maximum deviation from the diagonal). Note that nodes have been numbered in an attempt to avoid large differences between the indices of adjacent nodes. Figure 4.3 shows the resulting sparsity pattern. In this case, the automatic node-reordering algorithm of Puttonen [19] was employed to produce the numbering scheme of Figures 4.2 and 4.3. Algorithms that attempt to minimize the bandwidth of the matrix are generally used in connection with a solver that stores only a banded portion of the matrix.

Of the numerous automatic reordering algorithms that have been proposed, we will examine only the Cuthill–McKee algorithm [2, 7, 17, 20–23]. Although the Cuthill–McKee algorithm will generally not produce a sparsity pattern with optimum bandwidth, variations on it are widely used in connection with solvers that store only the envelope of the matrix. We will describe the algorithm by example while using it to optimize the ordering of the mesh depicted in Figure 3.11.

The starting point in the Cuthill–McKee algorithm is to determine the *degree* of each node in the mesh. For first-order interpolation functions, the degree of a node is the number of immediately adjacent nodes in the mesh. Table 4.1 lists the degree of each node in the mesh of Figure 3.11.

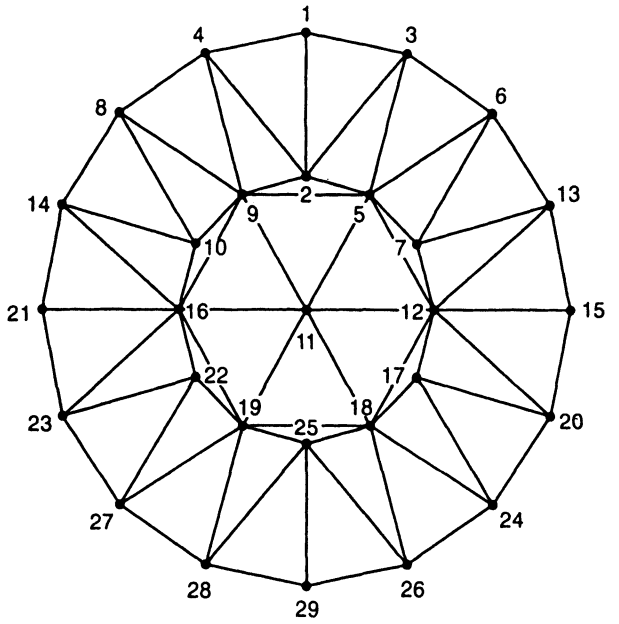



**Figure 4.2** Triangular-cell model showing a node-indexing scheme that attempts to minimize the bandwidth of the associated finite-element matrix.

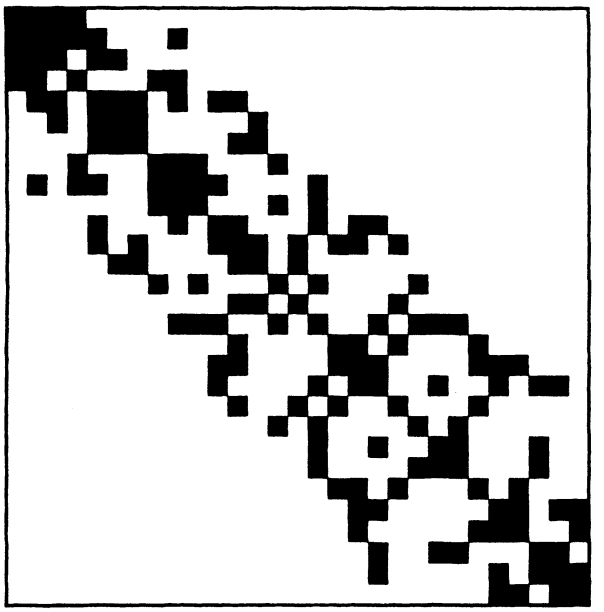


**Figure 4.3** Matrix sparsity pattern produced by the node-numbering scheme of Figure 4.2 for a first-order finite-element representation.

The procedure begins with the identification of the node with the smallest degree of connectivity (nodes numbered 14, 18, 22, or 26 in Table 4.1 are each connected to three other nodes). We select node 14 arbitrarily and relabel it as the new node 1. The second step of the procedure is to consider all the nodes connected to the "new" node 1 and order these so that they are arranged from smallest to largest degree of connectivity. Nodes 2, 15,

**TABLE 4.1** Degree of Connectivity of Each Node in Mesh of Figure 3.11 Assuming First-Order Interpolation Functions

| Node Number | Connectivity (deg) | Node Number | Connectivity (deg) |
|---|---|---|---|
| 1 | 6 | 16 | 4 |
| 2 | 8 | 17 | 4 |
| 3 | 7 | 18 | 3 |
| 4 | 4 | 19 | 4 |
| 5 | 7 | 20 | 4 |
| 6 | 5 | 21 | 4 |
| 7 | 8 | 22 | 3 |
| 8 | 4 | 23 | 4 |
| 9 | 7 | 24 | 4 |
| 10 | 4 | 25 | 4 |
| 11 | 7 | 26 | 3 |
| 12 | 5 | 27 | 4 |
| 13 | 4 | 28 | 4 |
| 14 | 3 | 29 | 4 |
| 15 | 4 | | |

and 29 are connected to the new node 1, and we renumber these according to the following permutation list:

| Old Number | New Number |
|---|---|
| 14 | 1 |
| 15 | 2 |
| 29 | 3 |
| 2 | 4 |

(Since nodes 15 and 29 have the same degree, we arbitrarily place the lowest index first.) We next consider each of the remaining nodes connected to any on the new list, adding them to the new list in the order of increasing degree. In this case, nodes 4 and 16 are connected to the original node 15 (the new node 2); nodes 13 and 28 are connected to the original node 29 (the new node 3); and nodes 1, 3, and 11 are connected to the original node 2 (new node 4). After adding these in order of lowest degree, we obtain the following:

| Old Number | New Number |
|---|---|
| 4 | 5 |
| 16 | 6 |
| 13 | 7 |
| 28 | 8 |
| 1 | 9 |
| 3 | 10 |
| 11 | 11 |

We now continue this procedure, identifying any nodes connected to new nodes 5, 6, and

so on, and adding them to the list in the order of increasing degree. There are no nodes connected to new node 5 that have not already been added to the permutation list. New node 6 is connected to node 17, so we renumber it as "new" node 12. We add nodes 27, 5, 7, 9, 6, and 12 in a similar fashion. Continuing through the entire list, we finally obtain the following:

| Old Number | New Number |
|---|---|
| 17 | 12 |
| 27 | 13 |
| 5 | 14 |
| 9 | 15 |
| 7 | 16 |
| 6 | 17 |
| 12 | 18 |
| 18 | 19 |
| 26 | 20 |
| 8 | 21 |
| 19 | 22 |
| 20 | 23 |
| 10 | 24 |
| 24 | 25 |
| 25 | 26 |
| 22 | 27 |
| 21 | 28 |
| 23 | 29 |

(If at some point in the procedure we find that we have exhausted the nodes connected to those on the permutation list, we begin anew by identifying the remaining node with the smallest connectivity. This would only happen if the model consisted of several isolated parts.) After permutation according to the Cuthill–McKee procedure, the matrix sparsity pattern for this example is displayed in Figure 4.4.

In several places throughout the above procedure, an arbitrary choice was made about the order of nodes. Consequently, there may be numerous different permutation lists produced by this procedure, and there is no guarantee that the specific permutation we obtained is the best that can be found (see Prob. P4.6). In addition, it is important to note that the Cuthill–McKee (CM) procedure is a relatively simple algorithm that will not, in general, produce the best possible numbering. In fact, the numbering employed to generate Figure 4.2 appears to be slightly better than this permutation (Figure 4.4).

George observed that the Cuthill–McKee algorithm could be improved for the purpose of minimizing the envelope of a sparse matrix by reversing the order of the Cuthill–McKee numbering [2, 7]. This produces the common implementation of the procedure, known as the *reverse Cuthill–McKee* (RCM) algorithm. Figure 4.5 shows the sparsity pattern and envelope obtained using the reverse permutation of the preceding example. Table 4.2 summarizes the relative efficiency of the four numbering schemes presented in this section.

Alternative reordering algorithms are available that may be more effective than the RCM approach [7, 17–26]. These procedures are beyond the scope of the present text.

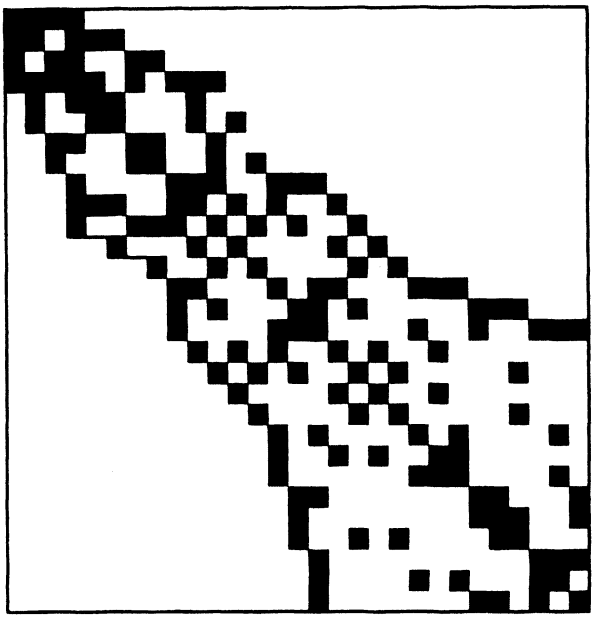


**Figure 4.4** Matrix sparsity pattern produced by the Cuthill–McKee algorithm.

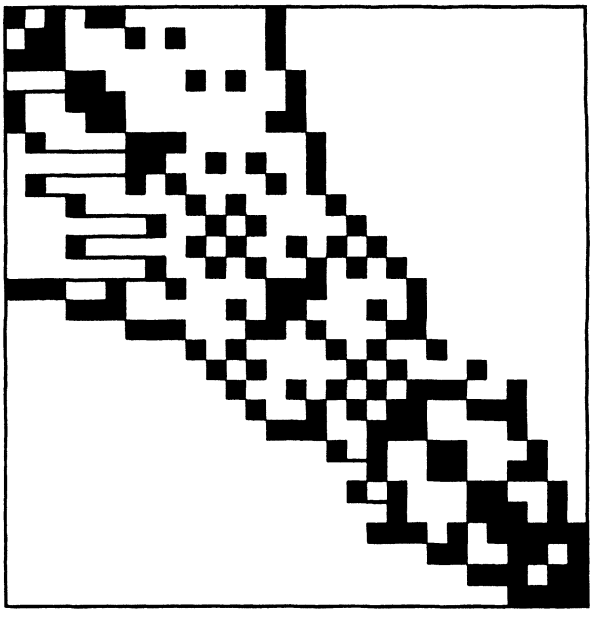


**Figure 4.5** Matrix sparsity pattern produced by the reverse Cuthill–McKee algorithm.

Although our example involved the nodes of a two-dimensional mesh, reordering algorithms are readily applied to order the indices associated with cells, edges, or any other quantity of interest. Commercial preprocessors used for general mesh generation often incorporate some type of automatic reordering algorithm [27].

**TABLE 4.2** Comparison of Storage Requirements Associated with Matrices of Figures 3.11, 4.3, 4.4, and 4.5 for Fixed-Bandwidth and Variable-Bandwidth (Envelope) Schemes

| Figure Number | Banded | Envelope |
|---|---|---|
| 3.11 (Original) | 839 | 631 |
| 4.3 (Puttonen) | 461 | 337 |
| 4.4 (CM) | 601 | 403 |
| 4.5 (RCM) | 601 | 335 |

*Note:* Numbers reflect only the storage of the matrix entries and do not include additional overhead or pointer arrays. Full-matrix storage for this example requires 841 locations, although there are only 165 nonzero entries in the original matrix.

## 4.6 BANDED STORAGE FOR GAUSSIAN ELIMINATION

If the numbering scheme effectively optimizes the bandwidth of the system (as perhaps the case with Figure 4.3), an efficient solution may be obtained by storing only a fixed band of the matrix entries located around the main diagonal. The LU factorization algorithms incorporating banded matrix storage are readily available in libraries such as LINPACK [6]. Factorization requires about $NB^2/2$ multiplications and additions, where $N$ is the order of the system and $B$ is the half-bandwidth (the maximum distance that a nonzero entry is displaced from the main diagonal). Advantages of banded-storage elimination algorithms include the simple storage allocation scheme and the fact that a modified form of pivoting is easily incorporated [2].

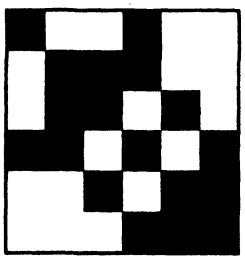
## 4.7 VARIABLE-BANDWIDTH OR ENVELOPE STORAGE FOR GAUSSIAN ELIMINATION

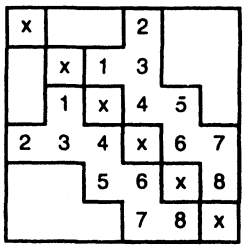
An improvement in efficiency over the banded matrix solvers can usually be realized through the use of factorization algorithms that store only the envelope of the matrix (Figure 4.1). Since they do not require a fixed band, these algorithms permit more flexibility in the numbering scheme used within the scatterer model. As illustrated by the comparisons presented in Table 4.2, envelope schemes require less storage than banded solvers. The improvement in efficiency comes at a cost that includes (1) the increased complexity of the storage scheme, which must use an additional array as a pointer to specify the shape of the envelope, and (2) the fact that it is generally more difficult to incorporate pivoting into the factorization algorithm.

In practice, the matrix envelope is stored in three parts: the main diagonal of the matrix (the *diagonal*), the matrix entries located below the main diagonal (the *lower enve-*

*lope*), and the matrix entries located above the main diagonal (the *upper envelope*). (If the matrix is symmetric and Cholesky decomposition is to be employed, the upper envelope is unnecessary.) Each of these three parts can be stored in a one-dimensional array. A pointer of some sort is necessary to specify the location of the original $(i, j)$th matrix entry in the one-dimensional arrays. The applications considered in this text always involve a matrix sparsity pattern that is symmetric across the main diagonal, which simplifies the required bookkeeping (i.e., only one pointer is necessary to specify the shape of the upper and lower envelopes, even if the matrix entries are not symmetric). The pointer array used to identify the location of the $(i, j)$ matrix entry in the one-dimensional arrays must be constructed prior to the "matrix fill" procedure and may be generated as part of the scatterer model in anticipation of a variable-bandwidth factorization. (To construct the pointer, it is only necessary to identify the locations of nonzero entries in the matrix. Since these are completely determined by the numbers assigned to the nodes of the mesh, the pointer is easily constructed once the mesh is specified.)

A simple storage scheme is proposed by George and Liu [17]. For the lower envelope, a pointer identifies the location in the single dimensional array of the first entry in each row of the matrix. The number of entries in a given row can be found from the difference between successive indices within the pointer. As an example, consider the matrix depicted in Figure 4.6, which has a lower envelope containing eight entries. The pointer $P = [1, 1, 1, 2, 5, 7, 9]$ identifies the location in the envelope of the first entry in each row of the $N \times N$ matrix. One additional entry at the end of the pointer acts as a flag to the end






| Row | Entries | Starting entry | Pointer |
|---|---|---|---|
| 1 | 0 | — | 1 |
| 2 | 0 | — | 1 |
| 3 | 1 | 1 | 1 |
| 4 | 3 | 2 | 2 |
| 5 | 2 | 5 | 5 |
| 6 | 2 | 7 | 7 |
| | | | 9 |

**Figure 4.6** Illustration of a scheme for storing the lower envelope of a sparse matrix in a one-dimensional array.

of the envelope. Assuming that $i > j$, the $(i, j)$th entry of the matrix is located at the $P[i + 1] - i + j$ location in the lower envelope array. For instance, the (4, 2) entry of the matrix is stored at the third position in the envelope (location $P[5] - 4 + 2 = 3$).

Factorization algorithms employing variable-bandwidth storage have been presented in FORTRAN or pseudocode by Jennings [2], George and Liu [17], and Hoole [23] for the case of positive-definite, real symmetric sparse matrices. It is possible to modify these algorithms to perform the naive LU factorization of complex matrices. Unfortunately, it is not easy to incorporate pivoting into the factorization process (since the interchange of rows or columns necessarily alters the matrix envelope).

## 4.8 SPARSE MATRIX METHODS EMPLOYING DYNAMIC STORAGE ALLOCATION

Banded or envelope solution schemes have fixed or *static* storage requirements that are established prior to the start of factorization. These algorithms are relatively unsophisticated and, since many zeros may be stored within the envelope, are often far from optimum for large-order systems. To avoid storing unnecessary zeros, more sophisticated factorization algorithms allocate storage *dynamically*. Examples of these algorithms include the Yale Sparse Matrix Package (YSMP) [28], MA28 and ME28 from the Harwell library [7, 29, 30], and Y12M [31, 32].

A matrix can be stored in packed form in a variety of ways, including (1) a linked list of the nonzero entries and (2) a one-dimensional array with pointers describing the row and column locations of each entry. These storage structures are flexible enough to incorporate any new entries that arise during factorization at the end of the list. For user convenience, dynamic factorization algorithms usually accept the original matrix entries in any order and sort them prior to factorization. Additional sorting, or "garbage collecting," may be performed throughout factorization in order to free workspace [32]. Since the storage structure can incorporate fill-in in any region of the matrix, pivoting can be included in the algorithm. However, in contrast to the full-storage case, pivots are primarily chosen in an attempt to minimize fill-in or minimize the number of arithmetic operations. By ensuring that pivots are always larger than some predetermined value, sufficient numerical stability can be provided as well. The details of the available algorithms are beyond the scope of the present discussion, and readers are encouraged to consult the texts by Duff, Erisman, and Reid [7], George and Liu [17], and Pissanetzky [18] for comprehensive overviews of these procedures.

We illustrate the performance of one algorithm, a complex-valued adaptation of Y12M, when used to solve the sparse system arising from a discretization of the scalar Helmholtz equation augmented with the Bayliss–Turkel radiation condition (Section 3.9). Piecewise-linear interpolation functions are used, as described in Chapter 3. No node reordering is applied prior to factorization. As an indication of the storage efficiency, Figure 4.7 shows the number of nonzero entries in the packed matrix before and after factorization by Y12M. For these two-dimensional finite-element problems, Figure 4.7 suggests that fill-in during factorization amplifies the storage requirements by a factor of 5. For large systems, this factor translates into a storage requirement of roughly 30 entries per row of the matrix, a smaller number than usually possible with banded or envelope storage.

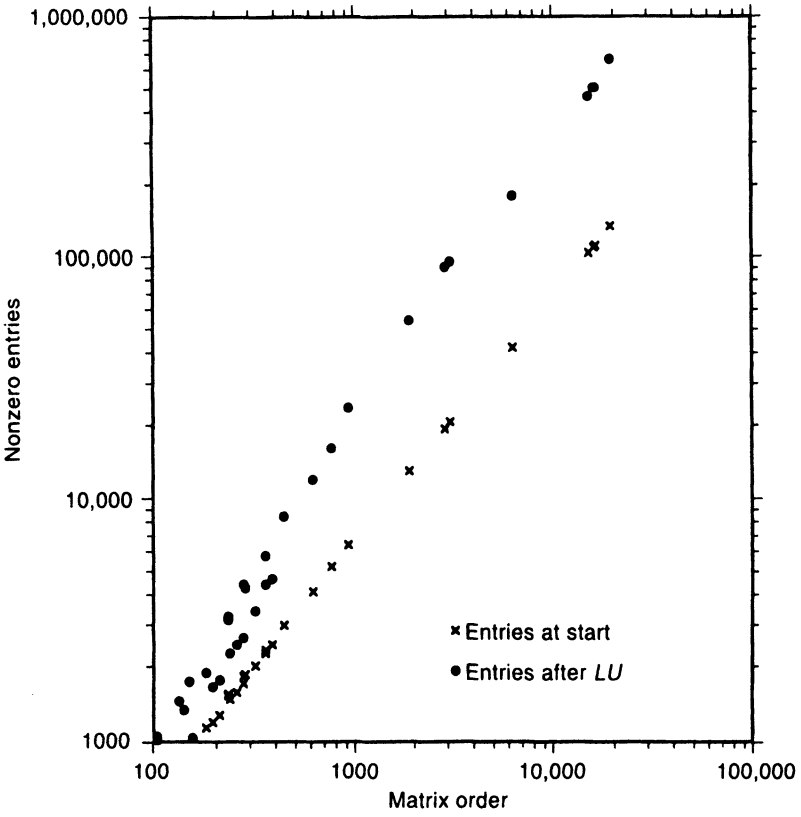



**Figure 4.7** Performance of the sparse LU factorization code Y12M when used to solve a number of first-order finite-element systems. To illustrate fill-in, the graph shows the number of entries before and after factorization as a function of matrix order.

## 4.9 FRONTAL ALGORITHM FOR GAUSSIAN ELIMINATION

The frontal solution method is an alternative direct method pioneered by Irons [33] as a way to reduce the amount of fast-access memory required to solve large systems. It is of interest for use with machines having a limited amount of fast-access memory and in modified form for certain parallel-architecture computers. To illustrate the frontal algorithm, consider the finite-element mesh shown in Figure 4.8. Figure 4.8 depicts eight triangular elements $a, b, \ldots, h$. Nine nodes in the mesh are identified by number. The initial step of the frontal method requires the ordering of the nodes to determine the order by which equations are eliminated from the system. In step 1 of the procedure, we assemble element $a$ and activate the equations associated with nodes 1, 2, and 3. Step 2 involves the recognition that element $a$ is the only cell that contributes to equation 1 of the global system. Thus, equation 1 is complete and Gaussian elimination can be performed to eliminate the corresponding unknown from the system. At this point, the reduced equation is written to secondary storage (i.e., a hard disk, solid-state disk, etc.) to release primary computer memory for future use.

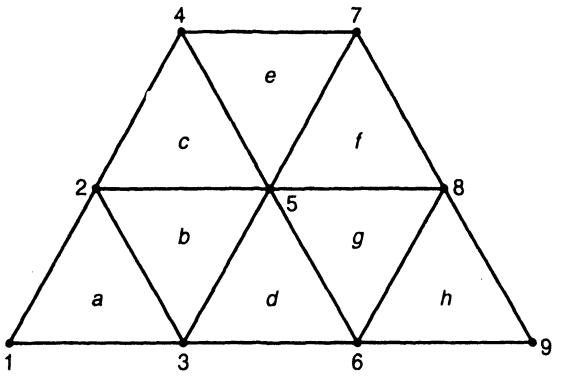



**Figure 4.8** Hypothetical triangular-cell mesh.

Step 3 of the frontal procedure is to assemble element *b* into the "front" containing the two equations surviving from step 2 and activate the additional equation corresponding to node 5. Step 4 is to assemble element *c* into the front and activate the equation associated with node 4. Step 5 is to recognize that equation 2 is now complete and can be eliminated from those equations remaining in primary memory (i.e., transferred to disk). Step 6 of the algorithm assembles element *d* and activates the equation for node 6. Step 7 is to eliminate equation 3 from the system. In step 8, element *e* is assembled and the equation at node 7 activated. Step 8 involves the elimination of the equation corresponding to node 4.

At this point, the algorithm has activated seven equations and eliminated four of these from primary computer memory. The process continues with the remaining cells, assembling *f*, *g*, and *h* until the frontal algorithm has reduced the entire system by Gaussian elimination. To obtain the solution, the final step employs back substitution in reverse order through the equations. For this simple example, only four equations are held in primary storage at any given time, compared to nine equations that would have to be retained if the complete system was treated in a conventional manner. As the mesh grows in size, the savings become more apparent. Memory savings will be improved as the aspect ratio of the mesh becomes more extreme. For example, a mesh that is 400 elements long and 10 elements wide yields an aspect ratio of 40. If the elements are sequenced across the width of the mesh, the maximum "frontwidth" at any time will be 22 nodes. On the other hand, if elements are improperly sequenced along the length of the mesh, the frontwidth will exceed 800 nodes. Reordering algorithms can be adapted to minimize frontwidth and can greatly improve the efficiency of this type of solution.

The drawback to the frontal solver is the speed of solution compared to in-core factorization algorithms. Since the frontal scheme requires transfers to secondary memory, its relative efficiency is machine dependent. However, the idea behind the frontal algorithm can also be applied to decompose problem domains for distributed-memory parallel-architecture computers.

## 4.10 ITERATIVE METHODS FOR MATRIX SOLUTION

The direct methods of the preceding sections are generally the preferred avenue of approach for treating general linear systems of equations. However, it is often difficult to fully exploit sparsity or special structure with direct methods. Furthermore, direct methods for sparse

systems are difficult to adapt to parallel-architecture computers. A wide variety of iterative methods have been developed [34, 35], but few are applicable to the general complex-valued matrix equations arising in electromagnetic radiation and scattering formulations. One approach that is applicable to general complex systems is the conjugate gradient algorithm [36, 37], and we consider this technique in detail in the following sections. For problems formulated in terms of integral equations, a common iterative implementation is the conjugate gradient–fast Fourier transform (CG–FFT) procedure. The CG–FFT implementation has found widespread use for treating systems with slightly perturbed Toeplitz symmetries. For problems lacking Toeplitz symmetries, the CG algorithm can be used with the fast multipole procedure (Section 4.13) to realize an efficiency similar to that of the CG–FFT approach.

Various extensions of the CG algorithm appear to offer better efficiency for poorly conditioned systems and may be more effective for treating the sparse systems arising from differential equation formulations than the CG algorithm we describe below. We refer the reader to the literature for a discussion of alternative approaches such as the biconjugate gradient procedure [38–40], the generalized minimal-residual method (GMRES) [40–42], and the quasi-minimal-residual (QMR) algorithm [40, 42–44].

## 4.11 THE CONJUGATE GRADIENT ALGORITHM FOR GENERAL LINEAR SYSTEMS [45]

There are several different ways in which the CG algorithm can be developed. For instance, we can view the algorithm as a procedure for the minimization of an error functional or as a process for constructing an orthogonal expansion of the solution. In actuality, these two ideas are linked together, that is, each functional is associated with a specific orthogonal expansion. The CG algorithm reduces to the process of generating the orthogonal vectors and finding the proper coefficients to represent the desired solution.

Consider the nonsingular matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.31}$$

where $\mathbf{A}$ denotes an $N \times N$ matrix, $\mathbf{x}$ is the unknown $N \times 1$ column vector to be determined, and $\mathbf{b}$ is a given $N \times 1$ column vector that we usually denote as the "right-hand side." It is necessary to define an inner product, and we employ the conventional Euclidean scalar product

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^{\dagger}\mathbf{y} \tag{4.32}$$

and its associated norm

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} \tag{4.33}$$

where the dagger denotes the transpose-conjugate matrix.

All iterative algorithms for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ seek an estimate of the solution in the form

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n \mathbf{p}_n \tag{4.34}$$

where $\mathbf{x}_{n-1}$ is a previous estimate of the solution, $\mathbf{p}_n$ is a "direction" vector ($\mathbf{p}_n$ determines the direction in the $N$-dimensional space in which the algorithm moves to correct the estimate

of $\mathbf{x}$), and $\alpha_n$ is a scalar coefficient that determines how far the algorithm moves in the $\mathbf{p}_n$ direction. Although all iterative methods are similar in that they follow the form of Equation (4.34), they differ in the procedure by which they generate $\alpha_n$ and $\mathbf{p}_n$. Nondivergence can be guaranteed by selecting $\alpha_n$ in order to minimize an error functional. The CG algorithm to be presented is based on the error functional

$$E_n(\mathbf{x}_n) = \|\mathbf{A}\mathbf{x}_n - \mathbf{b}\|^2 \tag{4.35}$$

Note that other functionals have been used and give rise to related members of the family of CG algorithms. The coefficient $\alpha_n$ from (4.34) that minimizes the functional is given by (Prob. P4.7)

$$\alpha_n = \frac{-\langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_{n-1}\rangle}{\|\mathbf{A}\mathbf{p}_n\|^2} \tag{4.36}$$

where for convenience we define the residual vector

$$\mathbf{r}_n = \mathbf{A}\mathbf{x}_n - \mathbf{b} \tag{4.37}$$

This process has the geometric interpretation of minimizing $E_n$ along the line in $N$-dimensional space defined by $\mathbf{p}_n$.

It is reasonable to expect that an improved algorithm would seek the minimum of $E_n$ in a plane spanned by two direction vectors. For example, consider a solution estimate of the form

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n(\mathbf{p}_n + \beta_n \mathbf{q}_n) \tag{4.38}$$

where the direction vectors $\mathbf{p}_n$ and $\mathbf{q}_n$ span a plane in $N$-dimensional space and the scalar coefficients $\alpha_n$ and $\beta_n$ are to be obtained in order to simultaneously minimize the error functional $E_n$. Carrying out the simultaneous minimization, we find that $\alpha_n$ is given by Equation (4.36) with $\mathbf{p}_n$ replaced by $\mathbf{p}_n + \beta_n \mathbf{q}_n$ and $\beta_n$ is given by

$$\beta_n = \frac{\langle \mathbf{A}\mathbf{q}_n, \mathbf{r}_{n-1}\rangle \|\mathbf{A}\mathbf{p}_n\|^2 - \langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_{n-1}\rangle \langle \mathbf{A}\mathbf{q}_n, \mathbf{A}\mathbf{p}_n\rangle}{\langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_{n-1}\rangle \|\mathbf{A}\mathbf{q}_n\|^2 - \langle \mathbf{A}\mathbf{q}_n, \mathbf{r}_{n-1}\rangle \langle \mathbf{A}\mathbf{p}_n, \mathbf{A}\mathbf{q}_n\rangle} \tag{4.39}$$

Suppose that $\mathbf{p}_n$ and $\mathbf{q}_n$ are arbitrary direction vectors but that $\mathbf{q}_n$ has been previously used in the iterative procedure, so that

$$\mathbf{x}_{n-1} = \mathbf{x}_{n-2} + \alpha_{n-1}\mathbf{q}_n \tag{4.40}$$

where $\alpha_{n-1}$ was previously found to minimize the error functional along the line defined by $\mathbf{q}_n$, that is,

$$\alpha_{n-1} = \frac{-\langle \mathbf{A}\mathbf{q}_n, \mathbf{r}_{n-2}\rangle}{\|\mathbf{A}\mathbf{q}_n\|^2} \tag{4.41}$$

It immediately follows that

$$\langle \mathbf{A}\mathbf{q}_n, \mathbf{r}_{n-1}\rangle = \langle \mathbf{A}\mathbf{q}_n, \mathbf{r}_{n-2} + \alpha_{n-1}\mathbf{A}\mathbf{q}_n\rangle = 0 \tag{4.42}$$

$$\beta_n = \frac{-\langle \mathbf{A}\mathbf{q}_n, \mathbf{A}\mathbf{p}_n\rangle}{\|\mathbf{A}\mathbf{q}_n\|^2} \tag{4.43}$$

and

$$\langle \mathbf{A}(\mathbf{p}_n + \beta_n \mathbf{q}_n), \mathbf{A}\mathbf{q}_n\rangle = 0 \tag{4.44}$$

Therefore, if the solution is already optimized along the $\mathbf{q}_n$ direction, the best solution in the plane spanned by $\mathbf{p}_n$ and $\mathbf{q}_n$ is obtained in a direction orthogonal to $\mathbf{q}_n$ in the sense of Equation (4.44).

This analysis shows that the process of selecting direction vectors and coefficients to minimize the error functional $E_n$ is optimized when vectors satisfying the orthogonality condition

$$\langle \mathbf{A}\mathbf{p}_i, \mathbf{A}\mathbf{p}_j \rangle = 0 \qquad i \neq j \tag{4.45}$$

are used. If an arbitrary set of direction vectors is employed, the process of minimizing $E_n$ will adjust their coefficients in order to generate a sequence satisfying this generalized orthogonality. Furthermore, if direction vectors satisfying (4.45) are used, there is no advantage in simultaneously minimizing the error functional along more than one direction. Vectors that satisfy (4.45) are said to be mutually conjugate with respect to the operator $\mathbf{A}^\dagger\mathbf{A}$, where $\mathbf{A}^\dagger$ is the adjoint with respect to the inner product, that is,

$$\langle \mathbf{A}^\dagger\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{y} \rangle \tag{4.46}$$

In accordance with our definition for the inner product, the matrix $\mathbf{A}^\dagger$ is the transpose conjugate of $\mathbf{A}$.

Suppose that a set of direction vectors satisfying the orthogonality condition of Equation (4.45) is readily available. Since $\mathbf{A}$ is nonsingular, these vectors are linearly independent and span the $N$-dimensional space. The solution can be expressed in the form

$$\mathbf{x} = \mathbf{x}_0 + \alpha_1\mathbf{p}_1 + \alpha_2\mathbf{p}_2 + \cdots + \alpha_N\mathbf{p}_N \tag{4.47}$$

where, for generality, the arbitrary vector $\mathbf{x}_0$ can be thought of as an initial estimate, or "guess," for the solution $\mathbf{x}$. Because of the orthogonality of (4.45), each coefficient can be found independently according to

$$\alpha_n = \frac{-\langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_0 \rangle}{\|\mathbf{A}\mathbf{p}_n\|^2} \tag{4.48}$$

where $\mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$.

From the above relationships, it is apparent that

$$\mathbf{r}_n = \mathbf{r}_0 + \alpha_1\mathbf{A}\mathbf{p}_1 + \alpha_2\mathbf{A}\mathbf{p}_2 + \cdots + \alpha_N\mathbf{A}\mathbf{p}_N \tag{4.49}$$

and recursive relationships are given as

$$\mathbf{r}_n = \mathbf{r}_{n-1} + \alpha_n\mathbf{A}\mathbf{p}_n \tag{4.50}$$

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n\mathbf{p}_n \tag{4.51}$$

and

$$\|\mathbf{r}_n\|^2 = \|\mathbf{r}_{n-1}\|^2 - |\alpha_n|^2\|\mathbf{A}\mathbf{p}_n\|^2 \tag{4.52}$$

From Equations (4.45), (4.48), and (4.49), we can readily deduce that

$$\langle \mathbf{A}\mathbf{p}_m, \mathbf{r}_n \rangle = \begin{cases} \langle \mathbf{A}\mathbf{p}_m, \mathbf{r}_0 \rangle & n < m \\ 0 & n \geq m \end{cases} \tag{4.53}$$

It follows that (4.36) and (4.48) are equivalent, indicating that the error minimization process and the orthogonal expansion procedure yield the same results.

The process of expanding a solution in terms of mutually conjugate direction vectors is known as the *conjugate direction method*, after Hestenes and Stiefel [36]. The conjugate direction method does not specify the means for generating a mutually conjugate sequence, however. The CG method is a conjugate direction method that includes a recursive procedure for generating the **p**-vectors. The CG algorithm begins with the choice

$$\mathbf{p}_1 = -\mathbf{A}^\dagger \mathbf{r}_0 \tag{4.54}$$

which is proportional to the gradient of the functional $E_n$ at $\mathbf{x} = \mathbf{x}_0$ (Prob. P4.10). Subsequent functions are found from

$$\mathbf{p}_{n+1} = -\mathbf{A}^\dagger \mathbf{r}_n + \beta_n \mathbf{p}_n \tag{4.55}$$

where the scalar coefficient $\beta_n$ is chosen to ensure

$$\langle \mathbf{A}^\dagger \mathbf{A}\mathbf{p}_n, \mathbf{p}_{n+1} \rangle = 0 \tag{4.56}$$

We will demonstrate that enforcing Equation (4.56) is sufficient to ensure that the **p**-vectors form a mutually conjugate set satisfying (4.45). To illustrate, we first present several relationships involving the vectors generated within the CG algorithm.

From Equation (4.55), we write

$$\langle \mathbf{A}^\dagger \mathbf{r}_m, \mathbf{p}_{n+1} \rangle = -\langle \mathbf{A}^\dagger \mathbf{r}_m, \mathbf{A}^\dagger \mathbf{r}_n \rangle + \beta_n \langle \mathbf{A}^\dagger \mathbf{r}_m, \mathbf{p}_n \rangle \tag{4.57}$$

According to Equation (4.53), the first and last inner product in (4.57) vanish for $m > n$, leaving

$$\langle \mathbf{A}^\dagger \mathbf{r}_n, \mathbf{A}^\dagger \mathbf{r}_m \rangle = 0 \qquad m \neq n \tag{4.58}$$

which is the orthogonality associated with the residual vectors. Equations (4.53) and (4.55) can be combined to produce

$$\langle \mathbf{p}_{n+1}, \mathbf{A}^\dagger \mathbf{r}_n \rangle = -\langle \mathbf{A}^\dagger \mathbf{r}_n, \mathbf{A}^\dagger \mathbf{r}_n \rangle = -\|\mathbf{A}^\dagger \mathbf{r}_n\|^2 \tag{4.59}$$

Therefore, $\alpha_n$ can be expressed in the alternate form

$$\alpha_n = \frac{\|\mathbf{A}^\dagger \mathbf{r}_{n-1}\|^2}{\|\mathbf{A}\mathbf{p}_n\|^2} \tag{4.60}$$

From Equation (4.50),

$$\mathbf{A}^\dagger \mathbf{r}_n = \mathbf{A}^\dagger \mathbf{r}_{n-1} + \alpha_n \mathbf{A}^\dagger \mathbf{A}\mathbf{p}_n \tag{4.61}$$

Because of the orthogonality expressed in Equation (4.58), an inner product between $\mathbf{A}^\dagger \mathbf{r}_m$ and Equation (4.61) leads to the result

$$\langle \mathbf{A}^\dagger \mathbf{r}_m, \mathbf{A}^\dagger \mathbf{A}\mathbf{p}_n \rangle = \begin{cases} \dfrac{\|\mathbf{A}^\dagger \mathbf{r}_m\|^2}{\alpha_n} & m = n \\ \dfrac{-\|\mathbf{A}^\dagger \mathbf{r}_m\|^2}{\alpha_n} & m = n-1 \\ 0 & \text{otherwise} \end{cases} \tag{4.62}$$

Using Equation (4.62) with $m = n$, we find the value of $\beta_n$ from Equations (4.55) and (4.56) to be

$$\beta_n = \frac{\|\mathbf{A}^\dagger \mathbf{r}_n\|^2}{\|\mathbf{A}^\dagger \mathbf{r}_{n-1}\|^2} \tag{4.63}$$

To see that this formula for $\beta_n$ guarantees the proper orthogonality between vectors when (4.45) is not explicitly enforced, consider the following. During the first iteration, Equations (4.36) and (4.50) are imposed, so that

$$\langle \mathbf{p}_1, \mathbf{A}^\dagger \mathbf{r}_1 \rangle = 0 \tag{4.64}$$

From Equation (4.54), this is equivalent to

$$\langle \mathbf{A}^\dagger \mathbf{r}_1, \mathbf{A}^\dagger \mathbf{r}_0 \rangle = 0 \tag{4.65}$$

Because of Equation (4.65), the expression for $\beta_1$ from (4.63) is sufficient to ensure that

$$\langle \mathbf{p}_1, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_2 \rangle = 0 \tag{4.66}$$

On the second iteration, Equations (4.50), (4.64), and (4.66) guarantee that

$$\langle \mathbf{p}_1, \mathbf{A}^\dagger \mathbf{r}_2 \rangle = -\langle \mathbf{A}^\dagger \mathbf{r}_0, \mathbf{A}^\dagger \mathbf{r}_2 \rangle = 0 \tag{4.67}$$

Taking an inner product of Equation (4.55) (with $n = 1$) and $\mathbf{A}^\dagger \mathbf{r}_2$, we find that

$$\langle \mathbf{A}^\dagger \mathbf{r}_2, \mathbf{A}^\dagger \mathbf{r}_1 \rangle = 0 \tag{4.68}$$

Therefore, the value of $\beta_2$ from (4.63) is sufficient to ensure that

$$\langle \mathbf{p}_2, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_3 \rangle = 0 \tag{4.69}$$

What remains is the validity of

$$\langle \mathbf{p}_1, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_3 \rangle = 0 \tag{4.70}$$

From Equations (4.55) and (4.63), $\mathbf{p}_n$ can be written as

$$\mathbf{p}_n = -\|\mathbf{A}^\dagger \mathbf{r}_{n-1}\|^2 \sum_{i=0}^{n-1} \frac{\mathbf{A}^\dagger \mathbf{r}_i}{\|\mathbf{A}^\dagger \mathbf{r}_i\|^2} \tag{4.71}$$

Using Equation (4.71) with $n = 3$ we obtain

$$\langle \mathbf{p}_3, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_1 \rangle = -\|\mathbf{A}^\dagger \mathbf{r}_2\|^2 \sum_{i=0}^{2} \frac{\langle \mathbf{A}^\dagger \mathbf{r}_i, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_1 \rangle}{\|\mathbf{A}^\dagger \mathbf{r}_i\|^2} \tag{4.72}$$

But, by the relationship established in (4.62), which is valid for these values of $n$ and $m$ as established in Equations (4.65), (4.67), and (4.68), the above reduces to

$$\langle \mathbf{p}_3, \mathbf{A}^\dagger \mathbf{A} \mathbf{p}_1 \rangle = -\|\mathbf{A}^\dagger \mathbf{r}_2\|^2 \left( \frac{-1}{\alpha_1} + \frac{1}{\alpha_1} \right) = 0 \tag{4.73}$$

Thus, in an inductive fashion we see that the direction vectors generated by the above procedure satisfy the assumed orthogonality properties of the conjugate direction method.

The CG algorithm is summarized in Table 4.3. In the computer science literature, this particular form of the CG algorithm is referred to as the "conjugate gradient method applied to the normal equations." The conventional CG algorithm discussed in many texts is restricted to the special case of a Hermitian positive-definite matrix **A**. To extend the algorithm to arbitrary linear systems, the matrix equation is premultiplied by $\mathbf{A}^\dagger$ to produce the *normal equations* $\mathbf{A}^\dagger \mathbf{A} \mathbf{x} = \mathbf{A}^\dagger \mathbf{b}$. Note that it is unnecessary to compute the product $\mathbf{A}^\dagger \mathbf{A}$. By using a different error functional or a different definition of the inner product, a variety of related CG algorithms can be constructed [46].

For an arbitrary nonsingular matrix **A**, the CG algorithm outlined above produces a solution in at most $N$ iteration steps (assuming infinite-precision arithmetic). This is a direct

**TABLE 4.3** Conjugate Gradient Algorithm

Initial steps:

Guess $\mathbf{x}_0$

$\mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$

$\mathbf{p}_1 = -\mathbf{A}^\dagger \mathbf{r}_0$

Iterate ($n = 1,2,\ldots$):

$$\alpha_n = -\frac{\langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_{n-1} \rangle}{\|\mathbf{A}\mathbf{p}_n\|^2} = \frac{\|\mathbf{A}^\dagger \mathbf{r}_{n-1}\|^2}{\|\mathbf{A}\mathbf{p}_n\|^2}$$

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n \mathbf{p}_n$$

$$\mathbf{r}_n = \mathbf{A}\mathbf{x}_n - \mathbf{b} = \mathbf{r}_{n-1} + \alpha_n \mathbf{A}\mathbf{p}_n$$

$$\beta_n = \frac{\|\mathbf{A}^\dagger \mathbf{r}_n\|^2}{\|\mathbf{A}^\dagger \mathbf{r}_{n-1}\|^2}$$

$$\mathbf{p}_{n+1} = -\mathbf{A}^\dagger \mathbf{r}_n + \beta_n \mathbf{p}_n$$

Terminate when a norm of $\mathbf{r}_n$ falls below some predetermined value; see cautionary note following Equation (4.76).

consequence of the fact that $N$ **p**-vectors span the solution space. Finite-step termination is a significant advantage of the CG method over other iterative algorithms. In addition, the CG algorithm produces solution estimates that satisfy (Prob. P4.11)

$$\|\mathbf{x} - \mathbf{x}_n\| \leq \|\mathbf{x} - \mathbf{x}_m\| \qquad n > m \tag{4.74}$$

In words, the error in $\mathbf{x}_n$ decreases monotonically as the algorithm progresses. Consequently, it will usually be possible to terminate the algorithm prior to the $N$th iteration step.

For the purpose of terminating the CG algorithm, it is necessary to estimate the accuracy of $x_n$ at each iteration step. Since the solution $\mathbf{x}$ is not known, the error vector

$$\mathbf{e}_n = \mathbf{x} - \mathbf{x}_n \tag{4.75}$$

is not directly computable. Instead, it is convenient to compute the residual norm

$$N_n = \frac{\|\mathbf{r}_n\|}{\|\mathbf{b}\|} = \frac{\|\mathbf{A}\mathbf{x}_n - \mathbf{b}\|}{\|\mathbf{b}\|} \tag{4.76}$$

As illustrated by Equation (4.52), the residual norm decreases monotonically (a direct consequence of minimizing $E_n$ at each iteration step). The CG algorithm can be terminated when the residual norm decreases to some predetermined value. As long as $\mathbf{A}$ is fairly well conditioned, $N_n < 10^{-4}$ suggests that several decimal places of accuracy are obtained in $\mathbf{x}_n$. However, note that the residual norm only provides an indirect bound on the error, according to

$$\frac{\|\mathbf{e}_n\|}{\|\mathbf{e}_0\|} \leq \kappa(\mathbf{A}) \frac{\|\mathbf{r}_n\|}{\|\mathbf{r}_0\|} \tag{4.77}$$

where $\kappa(\mathbf{A})$ is the condition number of the matrix $\mathbf{A}$. If the matrix $\mathbf{A}$ becomes ill-conditioned, $\kappa(\mathbf{A})$ will grow large and the residual norm $N_n$ may be a poor indication of the accuracy of $\mathbf{x}_n$.

The convergence rate of the CG algorithm depends on specific properties of the matrix equation under consideration. An upper bound on the accuracy of the solution estimate has

been obtained in terms of the condition number of **A** in the form [47]

$$\frac{\|\mathbf{e}_n\|}{\|\mathbf{e}_0\|} \leq 2\left[\frac{\kappa(\mathbf{A})-1}{\kappa(\mathbf{A}+1}\right]^n \tag{4.78}$$

According to Equation (4.78), if the matrix is poorly conditioned, the convergence may be quite slow. As a general observation, it has been reported that roughly $P\kappa(\mathbf{A})$ iterations are required to produce accuracy to $P$ decimal places [47]. When applied to integral equation formulations of electromagnetics, the observed convergence rate of the algorithm usually approximates a linear function of the decimal places of accuracy [48]. As an illustration, Figure 4.9 shows the performance of the CG algorithm applied to several examples of perfectly conducting strips or cylinders illuminated by a TM wave, following the approach discussed in Section 2.1.

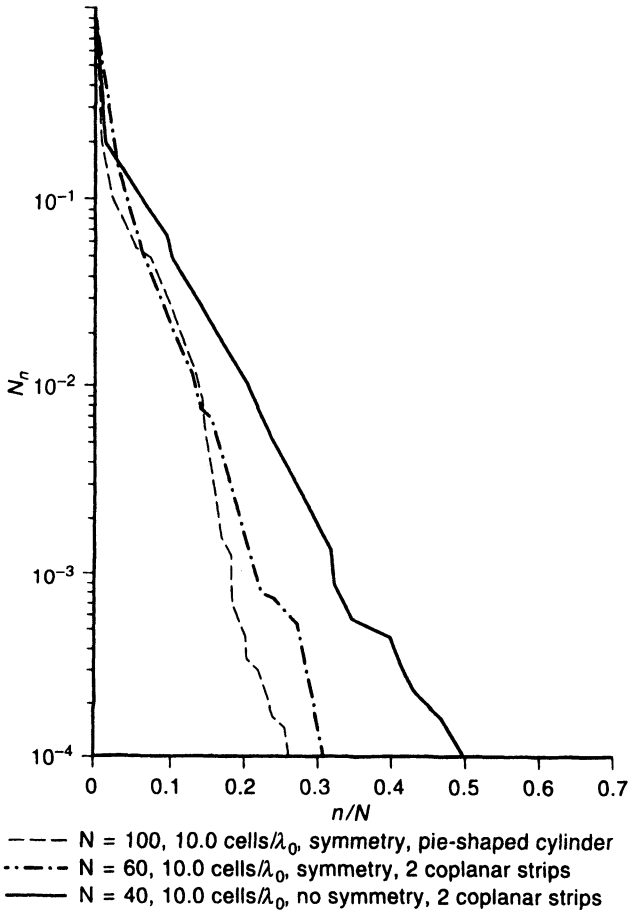



**Figure 4.9** Plot of the residual norm versus normalized iteration step for the conjugate gradient algorithm. The matrix equations represent the TM EFIE formulation for p.e.c. cylinders (Section 2.1). After [48]. ©1986 IEEE.

A precise characterization of the convergence behavior of the CG algorithm can be obtained in terms of the matrix **A** and the excitation **b** [49–51]. Observe that the residual

after the first iteration step can be written as

$$\mathbf{r}_1 = [\mathbf{I} - \alpha_1 \mathbf{A}\mathbf{A}^\dagger]\mathbf{r}_0 \tag{4.79}$$

After the second iteration step, the residual can be expressed as

$$\mathbf{r}_2 = [\mathbf{I} - (\alpha_1 + \alpha_2 + \alpha_2\beta_1)\mathbf{A}\mathbf{A}^\dagger + \alpha_1\alpha_2(\mathbf{A}\mathbf{A}^\dagger)^2]\mathbf{r}_0 \tag{4.80}$$

In general, the residual at the $n$th iteration step can be written as

$$\mathbf{r}_n = R_n(\mathbf{A}\mathbf{A}^\dagger)\mathbf{r}_0 \tag{4.81}$$

where $R_n(\mathbf{A}\mathbf{A}^\dagger)$ is a polynomial of order $n$ in the matrix $\mathbf{A}\mathbf{A}^\dagger$, that is,

$$R_n(\mathbf{A}\mathbf{A}^\dagger) = \sum_{k=0}^{n} \xi_{nk}(\mathbf{A}\mathbf{A}^\dagger)^k \tag{4.82}$$

Since the coefficients $\{\xi_{nk}\}$ in Equation (4.82) are combinations of the previous scalars $\alpha$ and $\beta$ from (4.60) and (4.63), they are real valued. Here, $R_n$ is known as the *residual polynomial* of order $n$.

The residual polynomial can be related to the error norm produced by the CG algorithm at the $n$th iteration step. Let $\{\lambda_i\}$ denote the eigenvalues of the matrix $\mathbf{A}\mathbf{A}^\dagger$ (these are also the eigenvalues of the matrix $\mathbf{A}^\dagger\mathbf{A}$) and let $\{\mathbf{u}_i\}$ denote the orthonormal eigenvectors of $\mathbf{A}\mathbf{A}^\dagger$. The initial residual vector can be decomposed into an eigenvector expansion according to

$$\mathbf{r}_0 = \sum_{i=1}^{N} \langle \mathbf{u}_i, \mathbf{r}_0 \rangle \mathbf{u}_i \tag{4.83}$$

Inserting this expansion into Equation (4.81) and replacing $(\mathbf{A}\mathbf{A}^\dagger)^k\mathbf{u}_i$ by $(\lambda_i)^k\mathbf{u}_i$ produce

$$\mathbf{r}_n = \sum_{i=1}^{N} \langle \mathbf{u}_i, \mathbf{r}_0 \rangle R_n(\lambda_i)\mathbf{u}_i \tag{4.84}$$

Finally, using the orthonormal property of the eigenvectors, the error functional being minimized by CG at the $n$th step can be written

$$E_n = \langle \mathbf{r}_n, \mathbf{r}_n \rangle = \sum_{i=1}^{N} |\langle \mathbf{u}_i, \mathbf{r}_0 \rangle|^2 [R_n(\lambda_i)]^2 \tag{4.85}$$

Since the coefficients $\langle \mathbf{u}_i, \mathbf{r}_0 \rangle$ are fixed by the excitation $\mathbf{b}$ and the initial estimate $\mathbf{x}_0$, the reduction in $E_n$ as the algorithm progresses is entirely due to changes in the residual polynomial $R_n$.

Equation (4.85) can be used to draw a variety of conclusions about the CG algorithm. First, note that at some point in the iteration process the algorithm will terminate with $E_n = 0$. At this point, the algorithm will have generated a residual polynomial $R_n(\lambda)$ having zeros at each of the eigenvalues $\lambda_i$ (Figure 4.10). Since the algorithm can place one additional zero in this polynomial at each iteration step and will place these zeros in order to minimize $E_n$, it follows that the CG algorithm will require at most $M$ steps to converge, where $M$ is the number of independent eigenvalues of $\mathbf{A}\mathbf{A}^\dagger$. In addition, if eigenvalues are repeated or clustered together in groups, the algorithm may only need to place one zero somewhere within the cluster in order to significantly reduce the error as measured by $E_n$.

The terms in Equation (4.85) are weighted by the coefficients of the eigenvector decomposition of the initial residual. If some of these coefficients vanish or are very

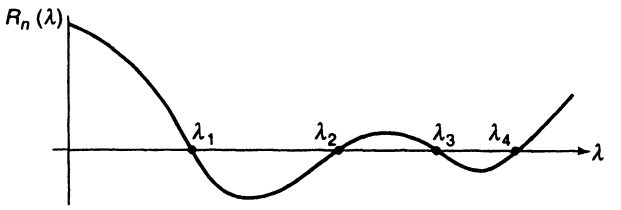



**Figure 4.10** Residual polynomial after the CG algorithm has converged.

small, the algorithm will not need to place a zero of the polynomial at the corresponding eigenvalues. This suggests that fewer than $M$ iteration steps may be required. In fact, if the initial residual can be represented by exactly $Q$ eigenvectors, the CG algorithm will converge in exactly $Q$ iterations!

The effect of the initial estimate $x_0$ on the convergence behavior is to alter the initial residual and therefore the coefficients in (4.85). The choice of $\mathbf{x}_0 = 0$ as an initial estimate of the solution produces an initial residual norm of $N_0 = 1$ in accordance with Equation (4.76). Intuitively, a "good" initial guess would appear be one that reduces $N_0$ from unity to some smaller value ($N_0 = 10^{-2}$?). However, it would be counterproductive to employ an initial estimate of the solution that excites more eigenvectors in the decomposition of (4.83) than would be excited by $\mathbf{x}_0 = 0$, because more iteration steps would be required despite a smaller initial residual. In other words, a good initial guess would attempt to minimize the number of nonzero coefficients in (4.83), rather than the initial residual norm. Because of the difficulty of ensuring this property, the zero estimate for $\mathbf{x}_0$ is often employed in practice and is sometimes called an "optimal" starting value. (Of course, the "best" starting value would be the solution $\mathbf{x}$.)

Equation (4.85) also sheds light on the observation that the convergence rate of the CG algorithm is typically slower for poorly conditioned systems. At termination, the residual polynomial $R_n(\lambda)$ must vanish at each important eigenvalue. However, the residual polynomial always has unity value at the origin ($\lambda = 0$). For a poorly conditioned matrix where the eigenvalue spectrum of $\mathbf{A}\mathbf{A}^\dagger$ is widely spread along the positive real axis, more degrees in the polynomial are necessary in order to initially reduce $E_n$, even in infinite-precision arithmetic. In finite-precision arithmetic, increased round-off errors during the computation of the matrix–vector operations exacerbate the generation of the polynomial and negate the finite step termination property of the CG algorithm. If the equation is very badly conditioned, convergence may slow to the point of stagnation. Even if the algorithm does appear to converge, because of (4.77), the residual norm usually employed to estimate the accuracy of the solution will not be reliable if $\mathbf{A}$ is poorly conditioned.

The observed performance of the CG algorithm for electromagnetic applications supports the preceding conclusions [51]. As an example, consider the EFIE formulation of Section 2.1 for TM scattering by a perfectly conducting circular cylinder of one wavelength circumference. Table 4.4 presents the residual norm versus the iteration step for a number of different discretizations. As the discretization is refined, the error $E_n$ observed after each iteration step appears to stabilize at certain values. For this example there are clearly only five important eigenvalues in the spectrum of the matrix operator, and refining the discretization beyond 10 cells/$\lambda$ does not significantly change the important part of the spectrum. (The eigenvalues of the matrix operator are necessarily related to those of the continuous integral operator; this relationship is explored in Chapter 5.)

For EM scattering problems formulated in terms of integral equations and involving uniform plane-wave excitations, numerical experiments suggest that there are typically

**TABLE 4.4** Values of Residual Norm Produced by CG Versus Iteration Step for Four Different Discretizations of Same Integral Equation

| | $N_n$ | | | |
|---|---|---|---|---|
| $n$ | $N = 4, \frac{4.0 \text{ cells}}{\lambda_0}$ | $N = 8, \frac{8.0 \text{ cells}}{\lambda_0}$ | $N = 16, \frac{16.0 \text{ cells}}{\lambda_0}$ | $N = 32, \frac{32.0 \text{ cells}}{\lambda_0}$ |
| 0 | 1.0 | 1.0 | 1.0 | 1.0 |
| 1 | 0.359 | 0.366 | 0.361 | 0.358 |
| 2 | 0.100 | 0.114 | 0.115 | 0.115 |
| 3 | $8.9 \times 10^{-10}$ | 0.0142 | 0.0161 | 0.0161 |
| 4 | — | 0.000706 | 0.00128 | 0.00132 |
| 5 | — | $2.2 \times 10^{-7}$ | $6.9 \times 10^{-5}$ | $8.0 \times 10^{-5}$ |

on the order of $N/3$ important eigenvalues in the spectrum of $\mathbf{A}\mathbf{A}^{\dagger}$ when a discretization involving about 10 subsectional cells per wavelength is employed [48, 51]. This suggests that the algorithm will require roughly $N/3$ iterations to produce a solution. For differential equation formulations such as those discussed in Chapter 3, the convergence rate is usually much slower [39].

In infinite-precision arithmetic, the CG algorithm produces a solution in at most $N$ iteration steps. Unfortunately, for a general linear system, CG requires approximately six times the number of operations as LU factorization to attain $N$ complete steps. Thus, to be competitive with direct methods, the CG algorithm would have to converge to necessary accuracy in fewer than $N/6$ iteration steps. As noted above, seldom is the observed convergence sufficiently rapid for general linear systems. Consequently, the CG algorithm is usually reserved for treating matrix equations having sparsity or special structure that cannot easily be exploited using direct methods of solution. Because the integral equations of electromagnetics involve convolutional kernels, the matrix equations of interest often possess discrete-convolutional symmetries. An implementation of the CG method for treating these systems is discussed in the following section.

## 4.12 THE CONJUGATE GRADIENT–FAST FOURIER TRANSFORM PROCEDURE [37, 52–59]

Direct methods for general matrix solution require the full $N \times N$ matrix to be stored in computer memory, placing a bottleneck on the solution process for large systems. If sufficient structure or sparsity is present in the equation of interest, iterative methods offer the possibility of avoiding this storage bottleneck. Iterative algorithms only require an implicit matrix operator (a subroutine that when given a column vector returns the product of the $N \times N$ system matrix with the column vector) and can easily exploit any type of matrix structure.

Electromagnetics problems posed in terms of integral equations with convolutional kernels can sometimes be discretized to yield matrices having discrete-convolutional symmetries. A general discrete convolution is an operation of the form [60, 61]

$$e_m = \sum_{n=0}^{N-1} j_n g_{m-n} \tag{4.86}$$

where $e$, $j$, and $g$ denote sequences of numbers. Equation (4.86) is equivalent to the matrix equation

$$\begin{bmatrix} g_0 & g_{-1} & g_{-2} & \cdots & g_{1-N} \\ g_1 & g_0 & g_{-1} & \ddots & \\ g_2 & g_1 & g_0 & \ddots & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ g_{N-1} & \cdots & \cdots & \cdots & g_0 \end{bmatrix} \begin{bmatrix} j_0 \\ j_1 \\ \cdot \\ \cdot \\ \cdot \\ j_{N-1} \end{bmatrix} = \begin{bmatrix} e_0 \\ e_1 \\ \cdot \\ \cdot \\ \cdot \\ e_{N-1} \end{bmatrix} \tag{4.87}$$

The $N \times N$ matrix depicted in Equation (4.87) is a general *Toeplitz* matrix. All of the elements of this matrix are described by the $2N - 1$ entries of the first row and column. If the elements of the sequence g repeat with period $N$, so that

$$g_{n-N} = g_n \qquad n = 1, 2, \ldots, N-1 \tag{4.88}$$

the operation is known as a *circular* discrete convolution and the $N \times N$ matrix in Equation (4.87) is *circulant*. Otherwise, the operation is a *linear* discrete convolution. Note that any linear discrete convolution of length $N$ can be embedded into a circular discrete convolution of length $2N - 1$. This can be accomplished by extending the original sequence $g$ to repeat with period $2N - 1$, zero padding the sequence $j$ to length $2N - 1$, and changing the upper limit of the summation in Equation (4.86) to $2N - 2$.

The FFT algorithm is an efficient way of implementing the discrete Fourier transform [60, 61]

$$\tilde{g}_n = \sum_{k=0}^{N-1} g_k e^{-j\frac{2\pi nk}{N}} \qquad n = 0, 1, \ldots, N-1 \tag{4.89}$$

The inverse discrete Fourier transform is defined

$$g_k = \frac{1}{N} \sum_{n=0}^{N-1} \tilde{g}_n e^{(j\frac{2\pi nk}{N})} \qquad k = 0, 1, \ldots, N-1 \tag{4.90}$$

For notational purposes, we use

$$\tilde{g} = \text{FFT}_N(g) \tag{4.91}$$

$$g = \text{FFT}_N^{-1}(\tilde{g}) \tag{4.92}$$

to denote the discrete Fourier transform pair for a sequence of length $N$. The discrete convolution theorem states that if Equation (4.86) is a circular discrete convolution of length $N$, it is equivalent to

$$\tilde{e}_n = \tilde{j}_n \tilde{g}_n \qquad n = 0, 1, \ldots, N-1 \tag{4.93}$$

If Equation (4.86) is a linear discrete convolution, the equivalence holds if the linear convolution is embedded in a circular convolution of length $2N - 1$ as described above.

To summarize, the discrete convolution operation of Equation (4.86) is equivalent to a Toeplitz matrix multiplication. Furthermore, either can be implemented using the FFT and inverse FFT algorithm according to the discrete convolution theorem [60, 61]

$$e = \text{FFT}_N^{-1}\left[\text{FFT}_N(j)\text{FFT}_N(g)\right] \tag{4.94}$$

If the discrete convolution is of the linear type, the FFTs must be of length $2N - 1$ rather than length $N$.

The above conclusions are easily generalized to two or three dimensions. A two-dimensional discrete convolution is an operation of the form

$$e_{pq} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} j_{nm} g_{p-n,q-m} \qquad \begin{Bmatrix} p \\ q \end{Bmatrix} = 0, 1, \ldots, \begin{Bmatrix} N-1 \\ M-1 \end{Bmatrix} \tag{4.95}$$

This equation is equivalent to the matrix operation

$$\begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_{-1} & \cdots & \mathbf{G}_{1-N} \\ \mathbf{G}_1 & \mathbf{G}_0 & \ddots & \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \mathbf{G}_{N-1} & \cdots & \cdots & \mathbf{G}_0 \end{bmatrix} \begin{bmatrix} \mathbf{J}_0 \\ \mathbf{J}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{J}_{N-1} \end{bmatrix} = \begin{bmatrix} \mathbf{E}_0 \\ \mathbf{E}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{E}_{N-1} \end{bmatrix} \tag{4.96}$$

where each element of the $N \times N$ *block Toeplitz* matrix of (4.96) is itself a Toeplitz matrix of the form depicted in (4.87). The relationship established in Equation (4.94) can be extended to multiple dimensions in an obvious manner.

To illustrate the appearance of discrete convolutional structure in electromagnetics equations, consider the scattering of a TM wave by an inhomogeneous dielectric cylinder, following the formulation of Section 2.5. Suppose we initially restrict our attention to the model appearing in Figure 4.11, which consists of one row of equal-sized square cells. Each cell in the model may represent a region of different permittivity, and the relative permittivity of the $n$th cell is denoted $\varepsilon_n$. Following the procedure of Section 2.5, we obtain the discrete system

$$e_m = \frac{\eta j_m}{jk(\varepsilon_m - 1)} + \sum_{n=0}^{N-1} j_n g_{m-n} \tag{4.97}$$

where

$$g_{m-n} = \frac{k\eta}{4} \iint_{\text{cell } n} H_0^{(2)} \left( k\sqrt{(x_m - x')^2 + (y_m - y')^2} \right) dx'dy' \tag{4.98}$$

This system can be written in matrix form as

$$\begin{bmatrix} g_0 + \chi_0 & g_{-1} & g_{-2} & \cdots & g_{1-N} \\ g_1 & g_0 + \chi_1 & g_{-1} & \ddots & \\ g_2 & g_1 & g_0 + \chi_2 & \ddots & \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ g_{N-1} & \cdots & \cdots & \cdots & g_0 + \chi_{N-1} \end{bmatrix} \begin{bmatrix} j_0 \\ j_1 \\ \cdot \\ \cdot \\ \cdot \\ j_{N-1} \end{bmatrix} = \begin{bmatrix} e_0 \\ e_1 \\ \cdot \\ \cdot \\ \cdot \\ e_{N-1} \end{bmatrix} \tag{4.99}$$

where

$$\chi_n = \frac{\eta}{jk(\varepsilon_n - 1)} \tag{4.100}$$

The presence of the relative permittivity on the main diagonal perturbs the $N \times N$ matrix from a purely Toeplitz form. Now consider a more general geometry modeled by the lattice

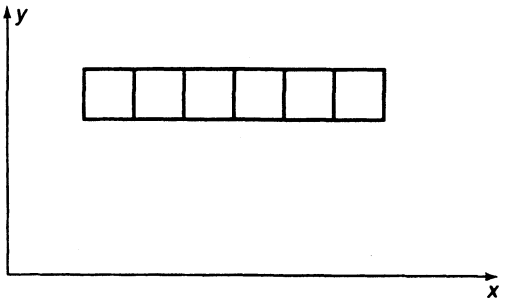



**Figure 4.11** One-dimensional lattice of uniform cells.

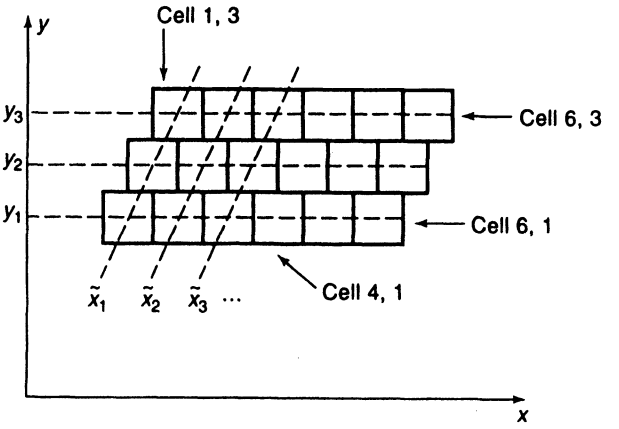



**Figure 4.12** Skewed two-dimensional lattice of uniform cells showing the indexing scheme used to build translational symmetries into the matrix equation.

of cells depicted in Figure 4.12. In this case, a two-dimensional numbering scheme is employed for the cells. For this geometry, the system of equations can be written as

$$e_{pq} = \frac{\eta j_{pq}}{jk(\varepsilon_{pq} - 1)} + \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} j_{mn} g_{p-m,q-n} \tag{4.101}$$

where

$$g_{p-m,q-n} = \frac{k\eta}{4} \iint_{\text{cell } m,n} H_0^{(2)} \left( k\sqrt{(x_p - x')^2 + (y_q - y')^2} \right) dx'\, dy' \tag{4.102}$$

The corresponding matrix has the perturbed block-Toeplitz form

$$\begin{bmatrix} \mathbf{G}_0 + \mathbf{X}_0 & \mathbf{G}_{-1} & \cdots & \mathbf{G}_{1-N} \\ \mathbf{G}_1 & \mathbf{G}_0 + \mathbf{X}_1 & \ddots & \\ \cdot & & & \\ \cdot & & & \\ \cdot & & & \\ \mathbf{G}_{N-1} & \cdots & \cdots & \mathbf{G}_0 + \mathbf{X}_{N-1} \end{bmatrix} \begin{bmatrix} \mathbf{J}_0 \\ \mathbf{J}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{J}_{N-1} \end{bmatrix} = \begin{bmatrix} \mathbf{E}_0 \\ \mathbf{E}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{E}_{N-1} \end{bmatrix} \tag{4.103}$$

where $\mathbf{X}_n$ denotes a diagonal matrix containing the perturbation due to the presence of the relative permittivity. Each matrix $\mathbf{G}_n$ is a Toeplitz matrix having the form of Equation (4.87). (The entries on the main diagonal are the only perturbation from a true block-Toeplitz system comprised of Toeplitz blocks.) In addition, for this example $\mathbf{G}_{-n} = \mathbf{G}_n$.

Because of the perturbation along the main diagonal, these systems are not suited for treatment with conventional Toeplitz or block-Toeplitz routines. However, they do contain a significant degree of structure. Iterative methods offer a way of exploiting this symmetry to reduce the storage to the level required by a conventional Toeplitz algorithm. For example, during each iteration step of the CG method, the only computations involving the matrix $\mathbf{A}$ are the product of $\mathbf{A}$ with a given column vector and the product of $\mathbf{A}^\dagger$ with a given column vector. The subroutines that perform the necessary matrix–vector multiplications can incorporate any sparsity or structure to minimize storage and computation costs. In addition, the specific type of matrix structure need not affect the organization of the part of the computer program that involves the main body of the iterative algorithm. The CG driver routine can be thought of as a "black box" similar in form to library routines that perform Gaussian elimination.

For the treatment of Equation (4.103), the CG algorithm can be implemented in two different ways. In the first approach, the matrix is stored in reduced form (one row and the main diagonal) and the required matrix–vector multiplications are performed directly. This implementation requires a minimum amount of storage but $O(N^2)$ operations to implement the matrix–vector multiplications. Since the discrete convolutions can be performed in $O(N \log N)$ operations using FFT algorithms, an alternative approach is to employ a multidimensional FFT to perform the discrete convolutions according to Equation (4.94). This latter approach is known as the CG–FFT implementation. Because of the zero padding required with linear discrete convolutions, use of the FFT involves a trade-off between storage and computational costs. For the two-dimensional scatterer depicted in Figure 4.12, use of the FFT roughly quadruples the required storage.

When the FFT is used to perform the required convolution between two sequences, the arrays must include terms corresponding to every possible location throughout the lattice. For scatterer geometries that do not completely fill the lattice shape, "dummy cells" are usually employed to complete the lattice. After each convolution is performed, locations in the resulting array that correspond to dummy cells are set to zero so that the iterative algorithm does not see additional unknowns at these cells.

The CG–FFT procedure has been a common use of the CG algorithm for treating integral equation formulations in electromagnetics. Unfortunately, to preserve the convolutional symmetry in the matrix, the scatterer geometry must be restricted to a relatively simple shape such as a flat plate, a surface of constant curvature, or a penetrable dielectric body discretized with uniform cells. The restrictions on the scatterer geometry necessary to preserve the convolutional symmetries limit the generality of the approach and prevent practical scatterers such as airplanes or bent wires from being analyzed this way. For electrically large geometries not imposing any type of structure on the $N \times N$ matrix, the fast multipole method introduced in the following section may offer an efficiency similar to that of the CG–FFT approach.

Iterative algorithms also suffer in comparison with direct methods of solution when it is necessary to treat multiple excitations. For example, Table 4.5 illustrates computation times associated with the analysis of flat-plate scattering using the CG–FFT procedure and LU factorization [62]. The CG–FFT procedure is more efficient for the solution of a large plate illuminated by a single incident field. However, when treating 91 different incident fields, the relative efficiency of LU factorization (which only requires an additional forward and back substitution for each additional excitation vector) is superior. Although some progress has been made in improving the performance of the CG algorithm for multiple

**TABLE 4.5** Comparison of CPU Time (CRAY XMP/24) Required to Solve the Matrix Equation Representing EFIE for Scattering from Square Conducting Plate of Side Dimension $a$ Using CG–FFT Algorithm and LU Factorization with Forward and Back Substitution

| | CG–FFT Solution | | | | LU Factorization | |
|---|---|---|---|---|---|---|
| $a\ (\lambda)$ | Unknowns | FFT Size | Average Number of Iterations | Average CPU Time for CG–FFT (sec) | Average CPU Time for 1 RHS (sec) | CPU Time for 91 RHS (sec) |
| 1.000 | 176 | 16 | 42 | 5.56 | 1.71 | 5.17 |
| 2.000 | 736 | 32 | 43 | 16.26 | 65.54 | 138.21 |
| 3.125 | 3,008 | 64 | 58 | 68.50 | — | — |
| 5.078 | 12,160 | 128 | 78 | 616.52 | — | — |

*Note*: Both procedures were used to treat 91 different excitations or "right-hand sides" (RHS). The averages were obtained by averaging over the 91 different excitations. After [62].

excitations [63], it is doubtful that iterative approaches will ever prove comparable to direct methods in this regard.

## 4.13 FAST MATRIX–VECTOR MULTIPLICATION: AN INTRODUCTION TO THE FAST MULTIPOLE METHOD

The choice of direct versus iterative methods is usually made based on the ultimate computational efficiency of the algorithm. Direct methods for dense matrices require $O(N^3)$ operations, while iterative methods require $O(PQ)$ operations, where $P$ is the number of iterations and $Q$ is the operation count per iteration. For a CG–FFT implementation, where $Q$ may grow as slowly as $N \log N$, an overall computational complexity of $O(PN \log N)$ may be obtained. The CG–FFT approach is attractive when $N$ is large, and the $O(N^3)$ operation count for dense matrices is prohibitive.

Unfortunately, the CG–FFT approach is restricted to translationally invariant geometries, which seriously limits its application to practical structures. An alternative scheme, known as the *fast multipole method*, offers the possibility of achieving $O(PN \log N)$ operations for arbitrarily shaped scatterers [64–68]. Variations on this approach appear to offer the most efficient possibilities yet proposed for the accurate numerical analysis of electrically large geometries, where $N$ may be far greater than $10^4$.

The central ideas behind the fast multipole method will be illustrated by considering a much simpler approximate scheme proposed by Lu and Chew [69]. Consider a TM-to-$z$ p.e.c. scatterer, and assume that the pulse basis–point matching discretization of Section 2.1 is used. Suppose that the scatterer perimeter is divided into $N$ cells and grouped into $p$ segments of roughly equal size and numbers of unknowns. The segments will be indexed $i = 1, 2, \ldots, p$. There will be $N/p$ cells per segment, which can be indexed $n = 0, 1, \ldots, N/p - 1$. One cell per segment will be centered at a local origin $(x_{i0}, y_{i0})$, while the other cell centroids will be denoted $(x_{in}, y_{in})$. Figure 4.13 illustrates two such segments.

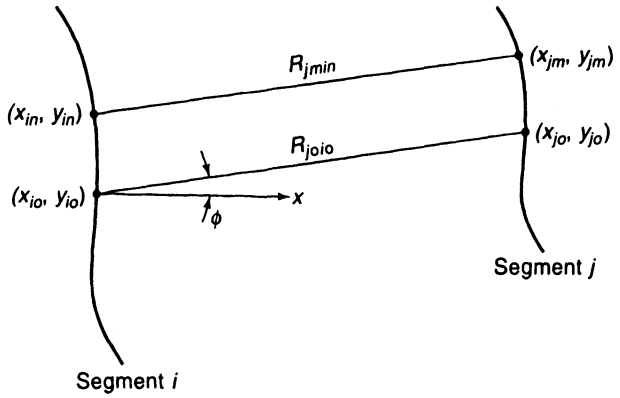



**Figure 4.13** Two segments in the far zone of each other, showing the parameter $R_{jmin}$.

Regardless of the iterative algorithm employed, each iteration step requires a matrix–vector multiplication operation equivalent to finding the scattered $E_z$-field at the center of each cell given the $N$ coefficients of the current density, $\{j_{in}\}$. To improve the efficiency of this step without sacrificing accuracy, the interactions between segments can be classified as either *near zone* or *far zone*, depending on the proximity of the segments. The near-zone interactions will be carried out by an explicit matrix–vector multiplication involving the conventional matrix entries, while the far-zone interactions will be replaced by a more efficient calculation. Let us assume that immediately adjacent segments are in the near zone of each other and all others fall in the far zone. Thus, the interactions between a given segment and its two adjacent neighbors are near zone and require $(3N/p)^2$ multiplications and additions. For all $p$ segments, the computational complexity of the near-zone interactions grows as $O(N^2/p)$.

The far-zone interactions can be simplified using a far-field approximation [69]. Consider the calculation of the field at $(x_{jm}, y_{jm})$ due to sources on segment $i$,

$$E_z^s(x_{jm}, y_{jm}) = -\frac{\omega\mu}{4} \sum_{n=0}^{N/p-1} j_n w_n H_0^{(2)}(kR_{jmin}) \tag{4.104}$$

where $R_{jmin}$ is defined in Figure 4.13 and we have incorporated the "single-point" approximation of the integral as used in Section 2.1. Segments $i$ and $j$ are in the far zone of each other, and we consider the far-field approximation (similar to that illustrated in Figure 1.17)

$$R_{jmin} \cong R_{j0i0} + R_{jm} - R_{in} \tag{4.105}$$

where

$$R_{j0i0} = \sqrt{(x_{j0} - x_{i0})^2 + (y_{j0} - y_{i0})^2} \tag{4.106}$$

$$R_{jm} = (x_{jm} - x_{j0}) \cos\phi + (y_{jm} - y_{j0}) \sin\phi \tag{4.107}$$

and

$$R_{in} = (x_{in} - x_{i0}) \cos\phi + (y_{in} - y_{i0}) \sin\phi \tag{4.108}$$

The angle $\phi$ denotes the orientation of $R_{j0i0}$ with respect to the $x$-axis (Figure 4.13). The

asymptotic form of the Hankel function for large arguments

$$H_0^{(2)}(k\rho) \approx \sqrt{\frac{2j}{\pi k\rho}} e^{-jk\rho} \tag{4.109}$$

motivates the approximation

$$H_0^{(2)}(kR_{jmin}) \cong H_0^{(2)}(kR_{j0i0}) e^{-jkR_{jm}} e^{+jkR_{in}} \tag{4.110}$$

and we replace Equation (4.104) with

$$E_z^s(x_{jm}, y_{jm}) \cong -\frac{\omega\mu}{4} H_0^{(2)}(kR_{j0i0}) e^{-jkR_{jm}} \sum_{n=0}^{N/p-1} j_n w_n e^{+jkR_{in}} \tag{4.111}$$

Using Equation (4.111), all interactions between the cells in segments $i$ and $j$ can be obtained from a single summation over the coefficients $\{j_{in}\}$ and one Hankel function calculation. This procedure involves $O(N/p)$ operations for a single segment pair. Since there are $p(p-3)$ combinations of far-zone segments in all, the entire far-zone computation grows with complexity $O(Np)$.

By combining the operation counts for the near-zone and the far-zone parts of the computation, we obtain a total that grows as

$$O\left(K_1 \frac{N^2}{p} + K_2 Np\right) \tag{4.112}$$

Clearly, the growth with respect to $N$ is minimized if the number of segments is proportional to the square root of $N$, producing an overall complexity for the matrix–vector multiplication process of $O(N^{1.5})$. For large $N$, this is superior to the $O(N^2)$ complexity of explicit matrix–vector multiplication.

To illustrate the relative efficiency of the procedure, suppose that the scatterer has perimeter dimension of 50,000 wavelengths, and a total of one million unknowns are employed. The perimeter is divided into 1000 segments. In this case, both the near-zone and far-zone interactions involve $O(10^9)$ operations, compared with $O(10^{12})$ for explicit matrix multiplication. Thus, the procedure requires approximately 1/1000 of the computation associated with a conventional implementation.

It is instructive to separate the far-zone calculation of Equation (4.111) into three parts, as suggested by Lu and Chew [69]. First, the sources on segment $i$ can be *aggregated* together via the summation

$$S_i \cong \sum_{n=0}^{N/p-1} j_n w_n e^{+jkR_{in}} \tag{4.113}$$

The second step, that of *translation*, uses the Hankel function

$$E_z^s(x_{j0}, y_{j0}) \cong -\frac{\omega\mu}{4} H_0^{(2)}(kR_{j0i0}) S_i \tag{4.114}$$

to shift the field to the center of segment $j$. Finally, the scattered field is *disaggregated* throughout segment $j$ via a multiplication with the phase correction

$$E_z^s(x_{jm}, y_{jm}) \cong e^{-jkR_{jm}} E_z^s(x_{j0}, y_{j0}) \tag{4.115}$$

These three distinct steps are analogous to those employed in the fast multipole method, which differs from the above primarily in the fact that it does not require a far-field approximation.

In the fast multipole method (FMM) of Rokhlin and his colleagues [64–67], the aggregation step replaces the field calculation in Equation (4.104) by a multipole expansion of the form

$$E_z^s(x_{jm}, y_{jm}) = \sum_q \alpha_q H_q^{(2)}(kR_{jmi0})e^{jq\phi} \tag{4.116}$$

whose coefficients can be obtained from (4.104) using the addition theorem (Prob. P3.21). Equation (4.116) is an expansion around a local origin on segment $i$. The translation step of the process is carried out using diagonalized translation operators associated with the multipole expansion to produce a similar expansion about a local origin on segment $j$. Finally, a disaggregation step is applied to obtain the scattered fields at individual cells [65, 66]. Unlike the simpler approach outlined above, Rokhlin's FMM does not involve a far-field approximation and can be carried out to a prescribed floating-point accuracy. The computational complexity per iteration step grows as $O(N^{1.5})$, and the storage requirements are similar. The efficiency improvement is obtained by replacing the individual interactions between cells in two far-zone segments with a single aggregated interaction.

A further reduction in computational complexity can be obtained by recursively nesting the segmentation concept developed above, by subdividing the segments into smaller segments, and by applying the same procedure to the internal interactions within each segment. In the limiting case, the multilevel algorithm involves $O(N \log N)$ operations per iteration [66]. Details of the FMM procedure for two- and three-dimensional geometries may be found in a number of recent articles [64–69]. (In the two-dimensional case, the FMM involves the multipole expansion described above. In contrast, the proposed three-dimensional FMM implementation employs a superposition of plane waves in geometrical directions adequate to reproduce the angular dependence of the original field, rather than an explicit multipole expansion [67].)

The FMM, and several other recently proposed algorithms similar to the FMM in many respects [70–73], permit a computational burden on the order of that obtained using the CG–FFT solution procedure but without the limitation of a translationally invariant scatterer geometry. These approaches are geared for electrically large structures, where direct methods of solution are prohibitive.

## 4.14 PRECONDITIONING STRATEGIES FOR ITERATIVE ALGORITHMS

The convergence rate of iterative algorithms, and the CG method in particular, is highly dependent on the eigenvalue spectrum of the matrix operator under consideration. Consequently, a scaling or transformation that converts the original system of equations into one with a more favorable eigenvalue spectrum might significantly improve the rate of convergence. This procedure is known as *preconditioning*. Although preconditioning strategies have been attempted for integral equation formulations [74–75], experience suggests that these formulations usually produce systems that are fairly well conditioned. However, the matrices arising from differential equation formulations (Chapter 3) are not always as well conditioned in general, and it is reasonable to expect that the performance of the CG algorithm applied to these systems can be improved by preconditioning.

A number of systematic approaches for sparse-matrix preconditioning have been developed, and an overview is presented by Evans [76]. The essential idea is to convert the system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{4.117}$$

into an equivalent equation

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{4.118}$$

where $\mathbf{M}^{-1}$ approximates $\mathbf{A}^{-1}$. For example, $\mathbf{M}^{-1}$ may be found by *incomplete LU factorization*. A complete LU factorization of $\mathbf{A}$ might require many times the storage originally required by $\mathbf{A}$ (Figure 4.7), exceeding available memory. By ignoring some or all of the fill-in, an approximate or incomplete factorization can be found with storage comparable to that of the original matrix. A number of other preconditioning schemes are possible [40, 42, 76]. As a general rule, the product of the matrix $\mathbf{M}^{-1}\mathbf{A}$ and a column vector can be constructed by repeated matrix–vector multiplications, so that it is not necessary to explicitly construct the product $\mathbf{M}^{-1}\mathbf{A}$.

If applied to Equation (4.118), the CG algorithm exhibits convergence behavior that depends on the operator $\mathbf{M}^{-1}\mathbf{A}$, which should prove much better conditioned than $\mathbf{A}$. Since the iterative algorithm constructs a solution to (4.118), the final solution will be correct regardless of the specific choice for $\mathbf{M}^{-1}$. In other words, there is no approximation inherent in preconditioning. However, preconditioning schemes can significantly increase the storage requirements and the computation required per iteration. Thus, they are usually reserved for problems where the convergence is prohibitively slow.

A simple preconditioning strategy that often proves worthwhile is to scale the system of equations so that the entries along the main diagonal are all equal. This procedure can compensate for orders-of-magnitude differences in scale that sometimes arise when several different equations are coupled together within a hybrid formulation. However, simple scaling is unlikely to have a significant impact on a system based on an inherently ill-conditioned underlying formulation.

## 4.15 SUMMARY

The matrix solution process is usually the largest computational task associated with the numerical treatment of an electromagnetic scattering problem. It is therefore important to employ an efficient, robust algorithm. This chapter has reviewed some of the techniques in use for full- and sparse-matrix solution. In addition, the relevant issue of matrix stability has been discussed. A variety of excellent software libraries have been developed for the purpose of matrix solution, and their use is generally recommended whenever possible.

The relative efficiency of direct and iterative methods depends on a number of factors. As a rule, direct methods tend to be more efficient for systems that can be stored entirely in the fast-access memory of the computer. This is especially true if it is necessary to solve one equation for many excitations. For moderately sized problems, iterative methods might be more efficient if the convergence rate is fast, but the convergence rate is difficult to determine in practice except by trial and error. Iterative methods are generally superior for specialized situations where significant matrix structure exists that cannot be easily exploited

by available direct methods or for electrically large problems that cannot be treated with direct methods.

## REFERENCES

[1] G. Strang, *Linear Algebra and Its Applications*, San Diego: Harcourt Brace Javanovich, 1988.

[2] A. Jennings, *Matrix Computation for Engineers and Scientists*, New York: Wiley, 1977.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Baltimore: Johns Hopkins University Press, 1983.

[4] K. E. Atkinson, *An Introduction to Numerical Analysis*, New York: Wiley, 1989.

[5] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Monterey: Brooks/Cole, 1980.

[6] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK User's Guide*, Philadelphia: SIAM, 1979.

[7] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford: Clarendon, 1986.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge: Cambridge University Press, 1986.

[9] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Englewood Cliffs, NJ: Prentice-Hall, 1989.

[10] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK User's Guide*, 2nd ed., Philadelphia: SIAM, 1995.

[11] J. H. Wilkinson, "Error analysis of direct methods of matrix inversion," *J. ACM*, vol. 8, pp. 281–330, July 1961.

[12] J. K. Reid, "A note on the stability of Gaussian elimination," *J. Inst. Math. Applicat.*, vol. 8, pp. 374–375, 1971.

[13] J. R. Bunch, "Partial pivoting strategies for symmetric matrices," *SIAM J. Num. Anal.*, vol. 11, pp. 521–528, 1974.

[14] J. R. Bunch, L. Kaufman, and B. N. Parlett, "Decomposition of a symmetric matrix," *Num. Math.*, vol. 27, pp. 95–110, 1976.

[15] V. Barwell and A. George, "A comparison of algorithms for solving symmetric indefinite systems of linear equations," *ACM Trans. Math. Software*, vol. 2, pp. 242–251, 1976.

[16] F. X. Canning, "Direct solution of the EFIE with half the computation," *IEEE Trans. Antennas Propagat.*, vol. 39, pp. 118–119, Jan. 1991.

[17] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1981.

[18] S. Pissanetzky, *Sparse Matrix Technology*, New York: Academic, 1984.

[19] J. Puttonen, "Simple and effective bandwidth reduction algorithm," *Int. J. Num. Methods Eng.*, vol. 19, pp. 1139–1152, 1983.

[20] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," *Proc. 24th Nat. Conf. Assoc. Comput. Mach.*, ACM Publ., pp. 157–172, 1969.

[21] E. Cuthill, "Several strategies for reducing the bandwidth of matrices," in *Sparse Matrices and Their Applications*, eds. D. J. Rose and R. A. Willoughby, New York: Plenum, 1972.

[22] H. R. Schwartz, *Finite Element Methods*, New York: Academic, 1988.

[23] S. R. H. Hoole, *Computer-Aided Analysis and Design of Electromagnetic Devices*, New York: Elsevier, 1989.

[24] H. R. Grooms, "Algorithm for matrix bandwidth reduction," *ASCE J. Struct. Div.*, vol. 98, pp. 203–214, 1972.

[25] R. J. Colins, "Bandwidth reduction by automatic renumbering," *Int. J. Num. Methods Eng.*, vol. 6, pp. 345–356, 1973.

[26] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix," *SIAM J. Num. Anal.*, vol. 13, April 1976.

[27] *PATRAN User's Guide*, Santa Ana, CA: PDA Engineering, 1984.

[28] S. C. Eisenstat, M C. Gursky, M. H. Schultz, and A. H. Sherman, "The Yale sparse matrix package II: The non-symmetric codes," Report 114, Department of Computer Science, Yale University, 1977.

[29] I. S. Duff, "MA28—A set of FORTRAN subroutines for sparse unsymmetric linear equations," Harwell Report AERE R-8730, Her Majesty's Stationery Office, London, 1977.

[30] I. S. Duff, "ME28: A sparse unsymmetric linear equation solver for complex equations," *ACM Trans. Math. Software*, vol. 7, pp. 505–511, Dec. 1981.

[31] Z. Zlatev, J. Wasniewski, and K. Schaumburg, *Y12M—Solution of Large and Sparse Systems of Linear Algebraic Equations* (Lecture Notes in Computer Science 121), Berlin: Springer-Verlag, 1981.

[32] O. Osterby and Z. Zlatev, *Direct Methods for Sparse Matrices* (Lecture Notes in Computer Science 157), Berlin: Springer-Verlag, 1983.

[33] B. M. Irons, "A frontal solution program for finite element analysis," *Int. J. Num. Methods Eng.*, vol 2, pp. 5–32, 1970.

[34] R. S. Varga, *Matrix Iterative Analysis*, Englewood Cliffs, NJ: Prentice-Hall, 1962.

[35] D. M. Young, *Iterative Solution of Large Linear Systems*, New York: Academic, 1971.

[36] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *J. Res. Nat. Bur. Stand.*, vol. 49, pp. 409–435, 1952.

[37] T. K. Sarkar, ed., *Application of Conjugate Gradient Method to Electromagnetics and Signal Analysis*, New York: Elsevier, 1991.

[38] D. A. H. Jacobs, "The exploitation of sparsity by iterative methods," in *Sparse Matrices and Their Uses*, ed. I. S. Duff, Berlin: Springer-Verlag, 1981.

[39] C. F. Smith, A. F. Peterson, and R. Mittra, "The biconjugate gradient method for electromagnetic scattering," *IEEE Trans. Antennas Propagat.*, vol. AP-38, pp. 938–940, June 1990.

[40] R. Barret, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romaine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Philadelphia: SIAM, 1994.

[41] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems," *SIAM J. Sci. Stat. Comp.*, vol. 7, pp. 856–869, July 1986.

[42] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Boston: PWS Publishing, 1996.

[43] R. Freund and N. Nachtigal, "An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices," RIACS Tech. Reports 90.45 and 90.46 (Parts I and II), Nov. 1990.

[44] N. Nachtigal, "A look-ahead variant of the Lanczos algorithm and its application to the quasi-minimal residual method for non-Hermitian linear systems," Ph.D. dissertation, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, Aug. 1991.

[45] A. F. Peterson, S. L. Ray, C. H. Chan, and R. Mittra, "Numerical implementations of the conjugate gradient method and the CG-FFT for electromagnetic scattering, in *Application of Conjugate Gradient Method to Electromagnetics and Signal Analysis*, ed. T. K. Sarkar, New York: Elsevier, 1991.

[46] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, "A taxonomy for conjugate gradient methods," Dept. of Computer Science Report UIUC-DCS-R-88-1414, UILU-ENG-88-1719, University of Illinois, Urbana, IL, April 1988.

[47] J. Stoer, "Solution of large linear systems of equations by conjugate gradient type methods," in *Mathematical Programming: The State of the Art*, eds. A. Bachem, M. Grötschel, and B. Korte, New York: Springer-Verlag, 1983.

[48] A. F. Peterson and R. Mittra, "Convergence of the conjugate gradient method when applied to matrix equations representing electromagnetic scattering problems," *IEEE Trans. Antennas Propagat.*, vol. AP-34, pp. 1447–1454, Dec. 1986.

[49] E. L. Stiefel, "Kernel polynomials in linear algebra and their numerical applications," *Nat. Bur. Stand. Appl. Math. Ser.*, vol. 49, pp. 1–22, 1958.

[50] A. Jennings, "Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method," *J. Inst. Math. Appl.*, vol. 20, pp. 61–72, 1977.

[51] A. F. Peterson, C. F. Smith, and R. Mittra, "Eigenvalues of the moment method matrix and their effect on the convergence of the conjugate gradient method," *IEEE Trans. Antennas Propagat.*, vol. AP-36, pp. 1177–1179, Aug. 1988.

[52] P. M. van den Berg, "Iterative computational techniques in scattering based upon the integrated square error criterion," *IEEE Trans. Antennas Propagat.*, vol. AP-32, pp. 1063–1071, Oct. 1984.

[53] D. T. Borup and O. P. Gandhi, "Calculation of high resolution SAR distribution in biological bodies using the FFT algorithm and conjugate gradient method," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-33, pp. 417–419, May 1985.

[54] A. F. Peterson and R. Mittra, "Method of conjugate gradients for the numerical solution of large body electromagnetic scattering problems," *J. Opt. Soc. Am. A*, vol. 2, pp. 971–977, June 1985.

[55] L. W. Pearson, "A technique for organizing large moment calculations for use with iterative solution methods," *IEEE Trans. Antennas Propagat.*, vol. AP-33, pp. 1031–1033, Sept. 1985.

[56] A. F. Peterson, "An analysis of the spectral iterative technique for electromagnetic scattering from individual and periodic structures," *Electromagnetics*, vol. 6, pp. 255–276, 1986.

[57] A. F. Peterson and R. Mittra, "Iterative-based computational methods for electromagnetic scattering from individual or periodic structures," *IEEE J. Oceanic Engineering*, Special Issue on Scattering, vol. OE-12, pp. 458–465, April 1987.

[58] C. C. Su, "Calculation of electromagnetic scattering from a dielectric cylinder using the conjugate gradient method and FFT," *IEEE Trans. Antennas Propagat.*, vol. AP-35, pp. 1418–1425, Dec. 1987.

[59] T. J. Peters and J. L. Volakis, "Application of a conjugate gradient FFT method to scattering from thin material plates," *IEEE Trans. Antennas Propagat.*, vol. AP-36, pp. 518–526, April 1988.

[60] E. O. Brigham, *The Fast Fourier Transform*, Englewood Cliffs, NJ: Prentice-Hall, 1974.

[61] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1975.

[62] C. H. Chan, "Investigation of iterative and spectral Galerkin techniques for solving electromagnetic boundary value problems," Ph.D. dissertation, University of Illinois, Urbana, IL, 1987.

[63] C. F. Smith, A. F. Peterson, and R. Mittra, "A conjugate gradient algorithm for the treatment of multiple incident electromagnetic fields," *IEEE Trans. Antennas Propagat.*, vol. AP-37, pp. 1490–1493, Nov. 1989.

[64] V. Rokhlin, "Rapid solution of integral equations of classical potential theory," *J. Computat. Phys.*, vol. 60, pp. 187–207, 1985.

[65] V. Rokhlin, "Rapid solution of integral equations of scattering theory in two dimensions," *J. Computat. Phys.*, vol. 86, pp. 414–439, 1990.

[66] N. Engheta, W. D. Murphy, V. Rokhlin, and M. S. Vassiliou, "The fast multipole method (FMM) for electromagnetic scattering problems," *IEEE Trans. Antennas Propagat.*, vol. 40, pp. 634–641, June 1992.

[67] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propagat. Mag.*, vol. 35, pp. 7–12, June 1993.

[68] C. C. Lu and W. C. Chew, "Fast algorithm for solving hybrid integral equations," *IEE Proc. Part H*, vol. 140, no. 6, pp. 455–460, Dec. 1993.

[69] C. C. Lu and W. C. Chew, "Fast far field approximation for calculating the RCS of large objects," *Proceedings of the Eleventh Annual Review of Progress in Applied Computational Electromagnetics*, The Applied Computational Electromagnetic Society, Monterey, CA, pp. 576–583, March 1995.

[70] C. R. Anderson, "An implementation of the fast multipole method without multipoles," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 4, pp. 923–947, July 1992.

[71] W. C. Chew, "Fast algorithms for wave scattering developed at the University of Illinois' Electromagnetics Laboratory," *IEEE Antennas Propagat. Mag.*, vol. 35, pp. 22–32, Aug. 1993.

[72] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, pp. 1225–1251, Sept./Oct. 1996.

[73] E. Michielssen and A. Boag, "A multilevel matrix decomposition algorithm for analyzing scattering from large structures," *IEEE Trans. Antennas Propagat.*, vol. 44, pp. 1086–1093, Aug. 1996.

[74] A. Kas and E. L. Yip, "Preconditioned conjugate gradient methods for solving electromagnetic problems," *IEEE Trans. Antennas Propagat.*, vol. AP-35, pp. 147–152, Feb. 1987.

[75] C. F. Smith, "The performance of preconditioned iterative methods in computational electromagnetics," Ph.D. dissertation, University of Illinois, Urbana, IL, 1987.

[76] D. J. Evans, ed., *Preconditioning Methods: Analysis and Application*, New York: Gordon & Breach, 1983.

## PROBLEMS

**P4.1** For the LU factorization procedure described in Section 4.1, calculate the number of additions, multiplications, and divisions required to treat an $N \times N$ system.

**P4.2** Find the condition number of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 10{,}000 \\ 0 & 2 \end{bmatrix}$$

using the definition in (4.11). How does the condition number $\kappa(\mathbf{A})$ compare with the ratio of the largest to smallest eigenvalues of $\mathbf{A}$?

**P4.3** The matrix

$$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 10^{10} \end{bmatrix}$$

is not normally considered ill conditioned, despite the fact that $\kappa(\mathbf{B}) = 10^{10}$. Why not? What is the essential difference between this matrix and the matrix $\mathbf{A}$ presented in Prob. P4.2?

**P4.4** By following a procedure similar to that outlined in Equations (4.15)–(4.18), derive (4.19).

**P4.5** The relation

$$\|(\mathbf{A} + \Delta\mathbf{A})^{-1}\| \leq \frac{\|\mathbf{A}^{-1}\|}{1 - \|\mathbf{A}^{-1}\|\|\Delta\mathbf{A}\|}$$

is valid when [4, p. 493]

$$\|\Delta\mathbf{A}\| < \frac{1}{\|\mathbf{A}^{-1}\|}$$

Using (4.15) and this relation, derive the inequality

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \frac{\kappa(\mathbf{A})}{1 - \kappa(\mathbf{A})\dfrac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}} \left\{ \frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} + \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} \right\}$$

This result is more general than (4.18) or (4.19).

**P4.6** Repeat the node-numbering exercise of Section 4.5, but begin the Cuthill–McKee procedure using node 18 in Figure 3.11 instead of node 14. Compare the resulting bandwidth and envelope size with the numbers in Table 4.2. Is this a better ordering?

**P4.7** Verify that (4.36) minimizes the error functional $E_n(\mathbf{x}_n)$ in (4.35). *Hint:* Since $\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n\mathbf{p}_n$, $E_n$ can be expanded as

$$E_n(\mathbf{x}_n) = \langle \mathbf{r}_{n-1}, \mathbf{r}_{n-1} \rangle + \alpha_n^\dagger \langle \mathbf{A}\mathbf{p}_n, \mathbf{r}_{n-1} \rangle + \alpha_n \langle \mathbf{r}_{n-1}, \mathbf{A}\mathbf{p}_n \rangle + \alpha_n\alpha_n^\dagger \langle \mathbf{A}\mathbf{p}_n, \mathbf{A}\mathbf{p}_n \rangle$$

where $\alpha_n^\dagger$ is the complex conjugate of $\alpha_n$ and the inner product is defined in (4.32). Because $\alpha_n$ may be complex valued, it is necessary to consider both

$$\frac{\partial E_n}{\partial \alpha_n} = 0 \quad \text{and} \quad \frac{\partial E_n}{\partial \alpha_n^\dagger} = 0$$

and show that these two conditions are satisfied by $\alpha_n$ in (4.36).

**P4.8** There are a number of alternative choices that could be used for the error functional $E_n$ on which a CG algorithm is based. For each of the following, determine the value of $\alpha_n$ that minimizes the functional $E_n(\mathbf{x}_n)$.

(a) $E^1(\mathbf{x}_n) = \langle \mathbf{x}_n - \mathbf{x}, \mathbf{x}_n - \mathbf{x} \rangle$
(b) $E^2(\mathbf{x}_n) = \langle \mathbf{A}\mathbf{x}_n - \mathbf{b}, \mathbf{x}_n - \mathbf{x} \rangle$
(c) $E^3(\mathbf{x}_n) = \langle \frac{1}{2}\mathbf{A}\mathbf{x}_n - \mathbf{b}, \mathbf{x}_n \rangle$

Use the definition in (4.32) for the inner product. For part (a), assume $\mathbf{A}$ is a general matrix operator. For parts (b) and (c), assume that $\mathbf{A}$ is self-adjoint, that is, $\mathbf{A} = \mathbf{A}^\dagger$.

**P4.9** For the case where $\mathbf{A}$ is a real, symmetric $3 \times 3$ matrix, demonstrate that the gradient of the functional

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

where $\mathbf{x}^T$ is the transpose of $\mathbf{x}$, is given by $\nabla \mathbf{f} = \mathbf{A}\mathbf{x} - \mathbf{b}$.

**P4.10** For the case where $\mathbf{A}$ is a real, nonsymmetric $3 \times 3$ matrix, demonstrate that the gradient of the functional

$$g(\mathbf{x}) = \langle \mathbf{A}\mathbf{x} - \mathbf{b}, \mathbf{A}\mathbf{x} - \mathbf{b} \rangle$$

is given by $\nabla g = 2\mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$, where $\mathbf{A}^T$ is the transpose of $\mathbf{A}$.

**P4.11** Prove Equation (4.74). *Hint:* One approach is to use (4.71) to show that

$$\langle \mathbf{p}_i, \mathbf{p}_j \rangle \geq 0$$

Then, use

$$\mathbf{x}_n - \mathbf{x}_m = \sum_{i=m+1}^{n} \alpha_i \mathbf{p}_i$$

to obtain the inequality

$$\langle \mathbf{x}_n - \mathbf{x}_m, \mathbf{x}_N - \mathbf{x}_n \rangle \geq 0 \qquad N > n > m$$

The identity

$$\|\mathbf{x} - \mathbf{x}_m\|^2 = \|\mathbf{x}_n - \mathbf{x}_m\|^2 + \|\mathbf{x} - \mathbf{x}_n\|^2 + 2\,\mathrm{Re}\,(\langle \mathbf{x}_n - \mathbf{x}_m, \mathbf{x} - \mathbf{x}_n \rangle)$$

may also be helpful.

**P4.12** An alternative conjugate gradient algorithm for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ may be based on the error functional $E_n(\mathbf{x}_n) = \langle \mathbf{x}_n - \mathbf{x}, \mathbf{x}_n - \mathbf{x} \rangle$, where the inner product is defined in (4.32) and the solution is expanded in direction vectors $\{\mathbf{q}_n\}$ according to $\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_n \mathbf{q}_n$. Using $\mathbf{q}_1 = -\mathbf{A}^\dagger \mathbf{r}_0$ and $\mathbf{q}_{n+1} = -\mathbf{A}^\dagger \mathbf{r}_n + \beta_n \mathbf{q}_n$, derive this CG algorithm by following a development similar to that used in Section 4.11. *Hint:* The appropriate direction vectors satisfy $\langle \mathbf{q}_i, \mathbf{q}_j \rangle = 0$, $i \neq j$, and the residual vectors satisfy $\langle \mathbf{r}_i, \mathbf{r}_j \rangle = 0$, $i \neq j$. The coefficients $\alpha_n$ and $\beta_n$ can be expressed as

$$\alpha_n = \frac{\|\mathbf{r}_{n-1}\|^2}{\|\mathbf{q}_n\|^2}$$

$$\beta_n = \frac{\|\mathbf{r}_n\|^2}{\|\mathbf{r}_{n-1}\|^2}$$

In the CG algorithm presented in Section 4.11, the error vectors $\mathbf{e}_n = \mathbf{x} - \mathbf{x}_n$ and the residual vectors $\mathbf{r}_n = \mathbf{A}\mathbf{x}_n - \mathbf{b}$ decrease monotonically. Is the same true for this CG algorithm?

**P4.13** Consider the problem of TM scattering from a dielectric cylinder using the volume integral equation formulation discussed in Sections 2.5 and 4.12. Suppose that the

cylinder has cross-sectional dimension $10\lambda \times 10\lambda$, relative permittivity $\varepsilon = 4$, and a uniform lattice of square cells employed with a cell density of 100 cells/$\lambda_d^2$. Compare the efficiency of a CG–FFT solution to a CG solution in which the FFT is not employed. For both methods, estimate the required computation per iteration and the required storage.

**P4.14** Repeat Prob. P4.13 for a cubical dielectric scatterer of side dimension $10\lambda$ and $\varepsilon_r = 4$ assuming that an analogous volume integral equation formulation is discretized using pulse basis functions, point matching, and a uniform lattice of cubical cells with a cell density of 1000 cells/$\lambda_d^3$. (Note that three components of the electric field are involved and that the matrix operator will therefore have a $3 \times 3$ block structure, with each block being a block-Toeplitz matrix. Each entry of the block-Toeplitz matrix is itself block-Toeplitz, with each entry of that matrix being Toeplitz.) Compare the efficiency of a CG–FFT solution to a CG solution in which the FFT is not employed by estimating the operations required per iteration for each approach. How much storage is required if (a) FFTs are used with minimum amount of zero padding, (b) no FFTs are used but all the Toeplitz symmetries are exploited to minimize storage, and (c) the entire matrix is stored?