1

Systems of Linear Algebraic Equations

1.1 INTRODUCTION

The static mechanical spring-mass system illustrated in Figure 1.1 consists of three masses m_1 to m_3, having weights W_1 to W_3, interconnected by five linear springs K_1 to K_5. In the configuration illustrated on the left, the three masses are supported by forces F_1 to F_3 equal to weights W_1 to W_3, respectively, so that the five springs are in a stable static equilibrium configuration. When the supporting forces F_1 to F_3 are removed, the masses move downward and reach a new static equilibrium configuration, denoted by x_1, x_2, and x_3, where x_1, x_2, and x_3 are measured from the original locations of the corresponding masses. Free-body diagrams of the three masses are presented at the bottom of Figure 1.1. Performing a static force balance on the three masses yields the following system of three linear algebraic equations:

$$(K_1 + K_2 + K_3)x_1 - K_2x_2 - K_3x_3 = W_1 \tag{1.1a}$$

$$-K_2x_1 + (K_2 + K_4)x_2 - K_4x_3 = W_2 \tag{1.1b}$$

$$-K_3x_1 - K_4x_2 + (K_3 + K_4 + K_5)x_3 = W_3 \tag{1.1c}$$

When values of K_1 to K_5 and W_1 to W_3 are specified, the equilibrium displacements x_1 to x_3 can be determined by solving Eq. (1.1).

The static mechanical spring-mass system illustrated in Figure 1.1 is used as the example problem in this chapter to illustrate methods for solving systems of linear

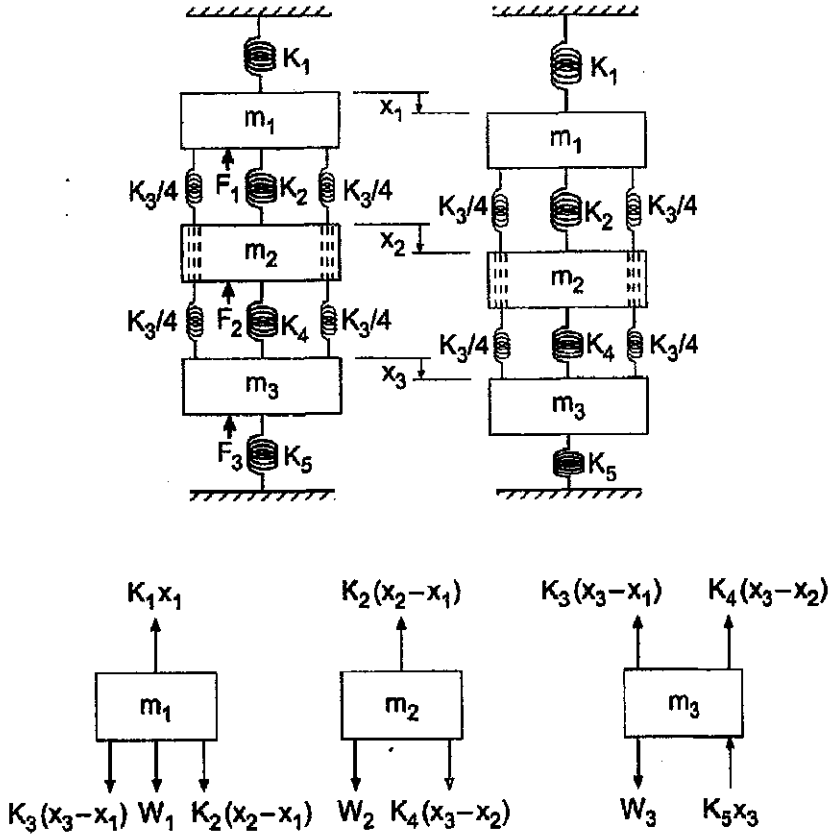

Figure 1.1 Static mechanical spring-mass system.

algebraic equations. For that purpose, let $K_1 = 40\,\text{N/cm}$, $K_2 = K_3 = K_4 = 20\,\text{N/cm}$, and $K_5 = 90\,\text{N/cm}$. Let $W_1 = W_2 = W_3 = 20\,\text{N}$. For these values, Eq. (1.1) becomes:

$$\boxed{\begin{aligned} 80x_1 - 20x_2 - 20x_3 &= 20 \quad (1.2a)\\ -20x_1 + 40x_2 - 20x_3 &= 20 \quad (1.2b)\\ -20x_1 - 20x_2 + 130x_3 &= 20 \quad (1.2c) \end{aligned}}$$

The solution to Eq. (1.2) is $x_1 = 0.6\,\text{cm}$, $x_2 = 1.0\,\text{cm}$, and $x_3 = 0.4\,\text{cm}$, which can be verified by direct substitution.

Systems of equations arise in all branches of engineering and science. These equations may be algebraic, transcendental (i.e., involving trigonometric, logarithmetic, exponential, etc. functions), ordinary differential equations, or partial differential equations. The equations may be linear or nonlinear. Chapter 1 is devoted to the solution of systems of linear algebraic equations of the following form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (1.3a)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \quad (1.3b)$$

$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \quad (1.3n)$$

where x_j $(j = 1, 2, \ldots, n)$ denotes the unknown variables, $a_{i,j}$ $(i, j = 1, 2, \ldots, n)$ denotes the constant coefficients of the unknown variables, and b_i $(i = 1, 2, \ldots, n)$ denotes the nonhomogeneous terms. For the coefficients $a_{i,j}$, the first subscript, i, denotes equation i, and the second subscript, j, denotes variable x_j. The number of equations can range from two to hundreds, thousands, and even millions.

In the most general case, the number of variables is not required to be the same as the number of equations. However, in most practical problems, they are the same. That is the case considered in this chapter. Even when the number of variables is the same as the number of equations, several solution possibilities exist, as illustrated in Figure 1.2 for the following system of two linear algebraic equations:

$$a_{11}x_1 + a_{12}x_2 = b_1 \quad (1.4a)$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \quad (1.4b)$$

The four solution possibilities are:

1. A unique solution (a consistent set of equations), as illustrated in Figure 1.2a
2. No solution (an inconsistent set of equations), as illustrated in Figure 1.2b
3. An infinite number of solutions (a redundant set of equations), as illustrated in Figure 1.2c
4. The trivial solution, $x_j = 0$ $(j = 1, 2, \ldots, n)$, for a homogeneous set of equations, as illustrated in Figure 1.2d

Chapter 1 is concerned with the first case where a unique solution exists.

Systems of linear algebraic equations arise in many different types of problems, for example:

1. Network problems (e.g., electrical networks)
2. Fitting approximating functions (see Chapter 4)

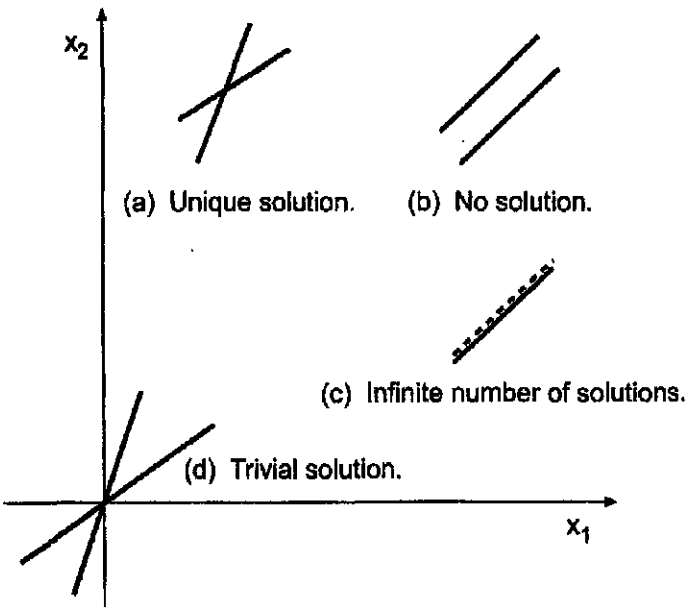

Figure 1.2 Solution of a system of two linear algebraic equations.

3. Systems of finite difference equations that arise in the numerical solution of differential equations (see Parts II and III)

The list is endless.

There are two fundamentally different approaches for solving systems of linear algebraic equations:

1. Direct elimination methods
2. Iterative methods

Direct elimination methods are systematic procedures based on algebraic elimination, which obtain the solution in a fixed number of operations. Examples of direct elimination methods are *Gauss elimination*, *Gauss-Jordan elimination*, the *matrix inverse method*, and *Doolittle LU factorization*. Iterative methods, on the other hand, obtain the solution asymptotically by an iterative procedure. A trial solution is assumed, the trial solution is substituted into the system of equations to determine the mismatch, or error, in the trial solution, and an improved solution is obtained from the mismatch data. Examples of iterative methods are *Jacobi iteration*, *Gauss-Seidel iteration*, and *successive-over-relaxation (SOR)*.

Although no absolutely rigid rules apply, direct elimination methods are generally used when one or more of the following conditions holds: (a) The number of equations is small (100 or less), (b) most of the coefficients in the equations are nonzero, (c) the system of equations is not diagonally dominant [see Eq. (1.15)], or (d) the system of equations is ill conditioned (see Section 1.6.2). Iterative methods are used when the number of equations is large and most of the coefficients are zero (i.e., a sparse matrix). Iterative methods generally diverge unless the system of equations is diagonally dominant [see Eq. (1.15)].

The organization of Chapter 1 is illustrated in Figure 1.3. Following the introductory material discussed in this section, the properties of matrices and determinants are reviewed. The presentation then splits into a discussion of direct elimination methods

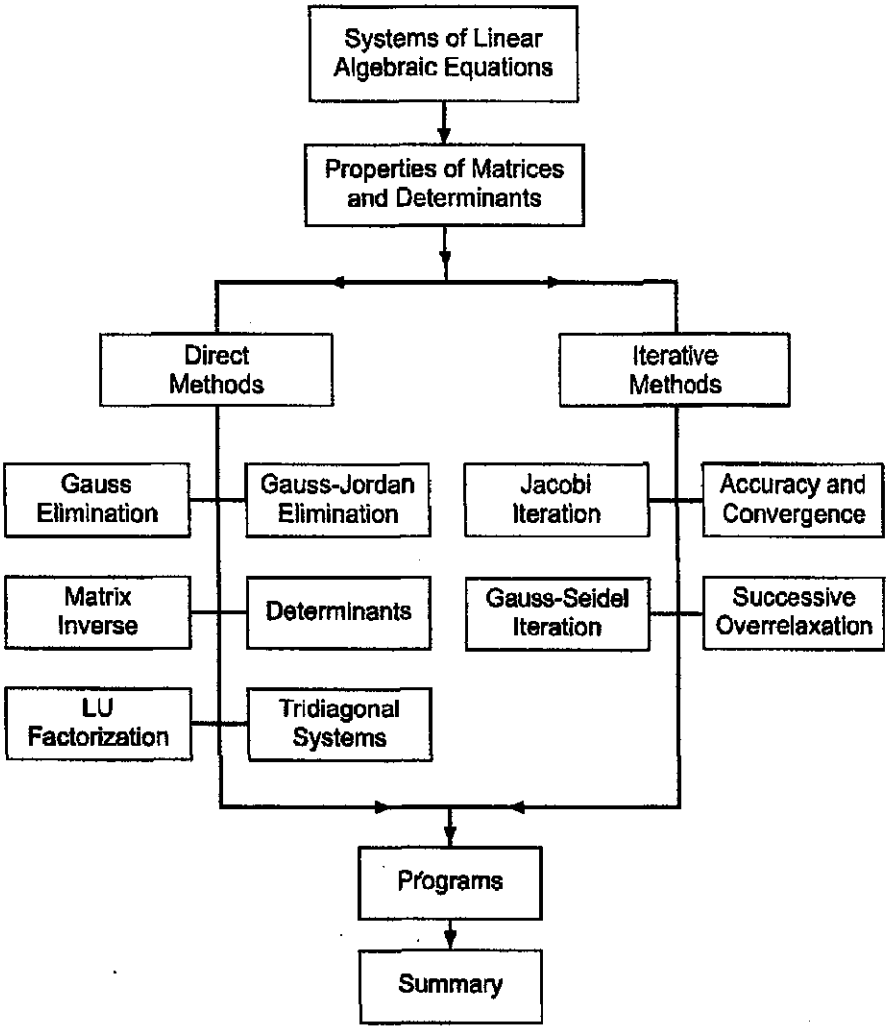

Figure 1.3 Organization of Chapter 1.

followed by a discussion of iterative methods. Several methods, both direct elimination and iterative, for solving systems of linear algebraic equations are presented in this chapter. Procedures for special problems, such as tridiagonal systems of equations, are presented. All these procedures are illustrated by examples. Although the methods apply to large systems of equations, they are illustrated by applying them to the small system of only three equations given by Eq. (1.2). After the presentation of the methods, three computer programs are presented for implementing the Gauss elimination method, the Thomas algorithm, and successive-over-relaxation (SOR). The chapter closes with a Summary, which discusses some philosophy to help you choose the right method for every problem and lists the things you should be able to do after studying Chapter 1.

1.2 PROPERTIES OF MATRICES AND DETERMINANTS

Systems of linear algebraic equations can be expressed very conveniently in terms of matrix notation. Solution methods for systems of linear algebraic equations can be

developed very compactly using matrix algebra. Consequently, the elementary properties of matrices and determinants are presented in this section.

1.2.1. Matrix Definitions

A *matrix* is a rectangular array of elements (either numbers or symbols), which are arranged in orderly rows and columns. Each element of the matrix is distinct and separate. The location of an element in the matrix is important. Elements of a matrix are generally identified by a double subscripted lowercase letter, for example, $a_{i,j}$, where the first subscript i identifies the row of the matrix and the second subscript j identifies the column of the matrix. The size of a matrix is specified by the number of rows times the number of columns. A matrix with n rows and m columns is said to be an n by m, or $n \times m$, matrix. Matrices are generally represented by either a boldface capital letter, for example, **A**, the general element enclosed in brackets, for example, $[a_{i,j}]$, or the full array of elements, as illustrated in Eq. (1.5):

$$\mathbf{A} = [a_{i,j}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nm} \end{bmatrix} \qquad (i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, m) \tag{1.5}$$

Comparing Eqs. (1.3) and (1.5) shows that the coefficients of a system of linear algebraic equations form the elements of an $n \times n$ matrix.

Equation (1.5) illustrates a convention used throughout this book for simplicity of appearance. When the general element $a_{i,j}$ is considered, the subscripts i and j are separated by a comma. When a specific element is specified, for example, a_{31}, the subscripts 3 and 1, which denote the element in row 3 and column 1, will not be separated by a comma, unless i or j is greater than 9. For example, a_{37} denotes the element in row 3 and column 7, whereas $a_{13,17}$ denotes the element in row 13 and column 17.

Vectors are a special type of matrix which has only one column or one row. Vectors are represented by either a boldface lowercase letter, for example, **x** or **y**, the general element enclosed in brackets, for example, $[x_i]$ or $[y_i]$, or the full column or row of elements. A *column vector* is an $n \times 1$ matrix. Thus,

$$\mathbf{x} = [x_i] = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \qquad (i = 1, 2, \ldots, n) \tag{1.6a}$$

A *row vector* is a $1 \times n$ matrix. For example,

$$\mathbf{y} = [y_j] = [y_1 \quad y_2 \quad \cdots \quad y_n] \qquad (j = 1, 2, \ldots, n) \tag{1.6b}$$

Unit vectors, **i**, are special vectors which have a magnitude of unity. Thus,

$$\|\mathbf{i}\| = (i_1^2 + i_2^2 + \cdots + i_n^2)^{1/2} = 1 \tag{1.7}$$

where the notation $\|\mathbf{i}\|$ denotes the length of vector **i**. Orthogonal systems of unit vectors, in which all of the elements of each unit vector except one are zero, are used to define coordinate systems.

There are several special matrices of interest. A *square matrix* **S** is a matrix which has the same number of rows and columns, that is, $m = n$. For example,

$$\mathbf{S} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \tag{1.8}$$

is a square $n \times n$ matrix. Our interest will be devoted entirely to square matrices. The left-to-right downward-sloping line of elements from a_{11} to a_{nn} is called the *major diagonal* of the matrix. A *diagonal matrix* **D** is a square matrix with all elements equal to zero except the elements on the major diagonal. For example,

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix} . \tag{1.9}$$

is a 4×4 diagonal matrix. The *identity matrix* **I** is a diagonal matrix with unity diagonal elements. The identity matrix is the matrix equivalent of the scalar number unity. The matrix

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

is the 4×4 identity matrix.

A *triangular matrix* is a square matrix in which all of the elements on one side of the major diagonal are zero. The remaining elements may be zero or nonzero. An *upper triangular matrix* **U** has all zero elements below the major diagonal. The matrix

$$\mathbf{U} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \tag{1.11}$$

is a 4×4 upper triangular matrix. A *lower triangular matrix* **L** has all zero elements above the major diagonal. The matrix

$$\mathbf{L} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{1.12}$$

is a 4×4 lower triangular matrix.

A *tridiagonal matrix* **T** is a square matrix in which all of the elements not on the major diagonal and the two diagonals surrounding the major diagonal are zero. The elements on these three diagonals may or may not be zero. The matrix

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \tag{1.13}$$

is a 5×5 tridiagonal matrix.

A *banded matrix* **B** has all zero elements except along particular diagonals. For example,

$$\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ a_{41} & 0 & a_{43} & a_{44} & a_{45} \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix} \tag{1.14}$$

is a 5×5 banded matrix.

The *transpose* of an $n \times m$ matrix **A** is the $m \times n$ matrix, $\mathbf{A}^{\mathrm{T}}$, which has elements $a^T_{i,j} = a_{j,i}$. The transpose of a column vector, is a row vector and vice versa. *Symmetric* square matrices have identical corresponding elements on either side of the major diagonal. That is, $a_{i,j} = a_{j,i}$. In that case, $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$.

A *sparse matrix* is one in which most of the elements are zero. Most large matrices arising in the solution of ordinary and partial differential equations are sparse matrices.

A matrix is *diagonally dominant* if the absolute value of each element on the major diagonal is equal to, or larger than, the sum of the absolute values of all the other elements in that row, with the diagonal element being larger than the corresponding sum of the other elements for at least one row. Thus, diagonal dominance is defined as

$$|a_{i,i}| \geq \sum_{j=1, j\neq i}^{n} |a_{i,j}| \qquad (i = 1, \ldots, n) \tag{1.15}$$

with $>$ true for at least one row.

1.2.2. Matrix Algebra

Matrix algebra consists of *matrix addition*, *matrix subtraction*, and *matrix multiplication*. Matrix division is not defined. An analogous operation is accomplished using the matrix inverse.

Matrix addition and subtraction consist of adding or subtracting the corresponding elements of two matrices of equal size. Let **A** and **B** be two matrices of equal size. Then,

$$\mathbf{A} + \mathbf{B} = [a_{i,j}] + [b_{i,j}] = [a_{i,j} + b_{i,j}] = [c_{i,j}] = \mathbf{C} \tag{1.16a}$$

$$\mathbf{A} - \mathbf{B} = [a_{i,j}] - [b_{i,j}] = [a_{i,j} - b_{i,j}] = [c_{i,j}] = \mathbf{C} \tag{1.16b}$$

Unequal size matrices cannot be added or subtracted. Matrices of the same size are *associative* on addition. Thus,

$$\mathbf{A} + (\mathbf{B} + \mathbf{C}) = (\mathbf{A} + \mathbf{B}) + \mathbf{C} \tag{1.17}$$

Matrices of the same size are *commutative* on addition. Thus,

$$\mathbf{A}+\mathbf{B}=\mathbf{B}+\mathbf{A} \tag{1.18}$$

Example 1.1. Matrix addition.

Add the two 3×3 matrices **A** and **B** to obtain the 3×3 matrix **C**, where

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 2 & 1 & 4\\ 1 & 4 & 3\end{bmatrix} \quad \text{and} \quad \mathbf{B}=\begin{bmatrix}3 & 2 & 1\\ -4 & 1 & 2\\ 2 & 3 & -1\end{bmatrix} \tag{1.19}$$

From Eq. (1.16a),

$$c_{i,j}=a_{i,j}+b_{i,j} \tag{1.20}$$

Thus, $c_{11}=a_{11}+b_{11}=1+3=4$, $c_{12}=a_{12}+b_{12}=2+2=4$, etc. The result is

$$\mathbf{A}+\mathbf{B}=\begin{bmatrix}(1+3) & (2+2) & (3+1)\\ (2-4) & (1+1) & (4+2)\\ (1+2) & (4+3) & (3-1)\end{bmatrix}=\begin{bmatrix}4 & 4 & 4\\ -2 & 2 & 6\\ 3 & 7 & 2\end{bmatrix}=\mathbf{C} \tag{1.21}$$

Matrix multiplication consists of row-element to column-element multiplication and summation of the resulting products. Multiplication of the two matrices **A** and **B** is defined only when the number of columns of matrix **A** is the same as the number of rows of matrix **B**. Matrices that satisfy this condition are called *conformable* in the order **AB**. Thus, if the size of matrix **A** is $n\times m$ and the size of matrix **B** is $m\times r$, then

$$\mathbf{AB}=[a_{i,j}][b_{i,j}]=[c_{i,j}]=\mathbf{C} \quad c_{i,j}=\sum_{k=1}^{m}a_{i,k}b_{k,j} \qquad (i=1,2,\ldots,n,\ j=1,2,\ldots,r) \tag{1.22}$$

The size of matrix **C** is $n\times r$. Matrices that are not conformable cannot be multiplied.

It is easy to make errors when performing matrix multiplication by hand. It is helpful to trace across the rows of **A** with the left index finger while tracing down the columns of **B** with the right index finger, multiplying the corresponding elements, and summing the products. Matrix algebra is much better suited to computers than to humans.

Multiplication of the matrix **A** by the scalar α consists of multiplying each element of **A** by α. Thus,

$$\alpha\mathbf{A}=\alpha[a_{i,j}]=[\alpha a_{i,j}]=[b_{i,j}]=\mathbf{B} \tag{1.23}$$

Example 1.2. Matrix multiplication.

Multiply the 3×3 matrix **A** and the 3×2 matrix **B** to obtain the 3×2 matrix **C**, where

$$\mathbf{A}=\begin{bmatrix}1 & 2 & 3\\ 2 & 1 & 4\\ 1 & 4 & 3\end{bmatrix} \quad \text{and} \quad \mathbf{B}=\begin{bmatrix}2 & 1\\ 1 & 2\\ 2 & 1\end{bmatrix} \tag{1.24}$$

From Eq. (1.22),

$$c_{i,j} = \sum_{k=1}^{3} a_{i,k} b_{k,j} \qquad (i = 1, 2, 3,\ j = 1, 2) \tag{1.25}$$

Evaluating Eq. (1.25) yields

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} = (1)(2) + (2)(1) + (3)(2) = 10 \tag{1.26a}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} = (1)(1) + (2)(2) + (3)(1) = 8 \tag{1.26b}$$

.....

$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} = (1)(1) + (4)(2) + (3)(1) = 12 \tag{1.26c}$$

Thus,

$$\mathbf{C} = [c_{i,j}] = \begin{bmatrix} 10 & 8 \\ 13 & 8 \\ 12 & 12 \end{bmatrix}. \tag{1.27}$$

Multiply the 3×2 matrix **C** by the scalar $\alpha = 2$ to obtain the 3×2 matrix **D**. From Eq. (1.23), $d_{11} = \alpha c_{11} = (2)(10) = 20$, $d_{12} = \alpha c_{12} = (2)(8) = 16$, etc. The result is

$$\mathbf{D} = \alpha\mathbf{C} = 2\mathbf{C} = \begin{bmatrix} (2)(10) & (2)(8) \\ (2)(13) & (2)(8) \\ (2)(12) & (2)(12) \end{bmatrix} = \begin{bmatrix} 20 & 16 \\ 26 & 16 \\ 24 & 24 \end{bmatrix} \tag{1.28}$$

Matrices that are suitably conformable are *associative* on multiplication. Thus,

$$\mathbf{A(BC)} = \mathbf{(AB)C} \tag{1.29}$$

Square matrices are *conformable* in either order. Thus, if **A** and **B** are $n \times n$ matrices,

$$\mathbf{AB} = \mathbf{C} \qquad \text{and} \qquad \mathbf{BA} = \mathbf{D} \tag{1.30}$$

where **C** and **D** are $n \times n$ matrices. However square matrices in general are not *commutative* on multiplication. That is, in general,

$$\mathbf{AB} \neq \mathbf{BA} \tag{1.31}$$

Matrices **A**, **B**, and **C** are *distributive* if **B** and **C** are the same size and **A** is *conformable* to **B** and **C**. Thus,

$$\mathbf{A(B + C)} = \mathbf{AB} + \mathbf{AC} \tag{1.32}$$

Consider the two square matrices **A** and **B**. Multiplying yields

$$\mathbf{AB} = \mathbf{C} \tag{1.33}$$

It might appear logical that the inverse operation of multiplication, that is, division, would give

$$\mathbf{A} = \mathbf{C/B} \tag{1.34}$$

Unfortunately, matrix division is not defined. However, for square matrices, an analogous concept is provided by the matrix inverse.

Consider the two square matrices **A** and **B**. If $\mathbf{AB} = \mathbf{I}$, then **B** is the inverse of **A**, which is denoted as $\mathbf{A}^{-1}$. Matrix inverses *commute* on multiplication. Thus,

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \tag{1.35}$$

The operation desired by Eq. (1.34) can be accomplished using the matrix inverse. Thus, the inverse of the matrix multiplication specified by Eq. (1.33) is accomplished by matrix multiplication using the inverse matrix. Thus, the matrix equivalent of Eq. (1.34) is given by

$$\mathbf{A} = \mathbf{B}^{-1}\mathbf{C} \tag{1.36}$$

Procedures for evaluating the inverse of a square matrix are presented in Examples 1.12 and 1.16.

Matrix factorization refers to the representation of a matrix as the product of two other matrices. For example, a known matrix **A** can be represented as the product of two unknown matrices **B** and **C**. Thus,

$$\mathbf{A} = \mathbf{BC} \tag{1.37}$$

Factorization is not a unique process. There are, in general, an infinite number of matrices **B** and **C** whose product is **A**. A particularly useful factorization for square matrices is

$$\mathbf{A} = \mathbf{LU} \tag{1.38}$$

where **L** and **U** are lower and upper triangular matrices, respectively. The LU factorization method for solving systems of linear algebraic equations, which is presented in Section 1.4, is based on such a factorization.

A matrix can be *partitioned* by grouping the elements of the matrix into submatrices. These submatrices can then be treated as elements of a smaller matrix. To ensure that the operations of matrix algebra can be applied to the submatrices of two partitioned matrices, the partitioning is generally into square submatrices of equal size. Matrix partitioning is especially convenient when solving systems of algebraic equations that arise in the finite difference solution of systems of differential equations.

1.2.3. Systems of Linear Algebraic Equations

Systems of linear algebraic equations, such as Eq. (1.3), can be expressed very compactly in matrix notation. Thus, Eq. (1.3) can be written as the matrix equation

$$\boxed{\mathbf{Ax} = \mathbf{b}} \tag{1.39}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix} \tag{1.40}$$

Equation (1.3) can also be written as

$$\sum_{j=1}^{n} a_{i,j}x_j = b_i \quad (i = 1, \ldots, n) \tag{1.41}$$

or equivalently as

$$a_{i,j}x_j = b_i \qquad (i,j = 1,\ldots,n) \tag{1.42}$$

where the *summation convention* holds, that is, the repeated index j in Eq. (1.42) is summed over its range, 1 to n. Equation (1.39) will be used throughout this book to represent a system of linear algebraic equations.

There are three so-called *row operations* that are useful when solving systems of linear algebraic equations. They are:

1. Any row (equation) may be multiplied by a constant (a process known as *scaling*).
2. The order of the rows (equations) may be interchanged (a process known as *pivoting*).
3. Any row (equation) can be replaced by a weighted linear combination of that row (equation) with any other row (equation) (a process known as *elimination*).

In the context of the solution of a system of linear algebraic equations, these three row operations clearly do not change the solution. The appearance of the system of equations is obviously changed by any of these row operations, but the solution is unaffected. When solving systems of linear algebraic equations expressed in matrix notation, these row operations apply to the rows of the matrices representing the system of linear algebraic equations.

1.2.4. Determinants

The term *determinant* of a square matrix **A**, denoted det(**A**) or |**A**|, refers to both the collection of the elements of the square matrix, enclosed in vertical lines, and the scalar value represented by that array. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \tag{1.43}$$

Only square matrices have determinants.

The scalar value of the determinant of a 2×2 matrix is the product of the elements on the major diagonal minus the product of the elements on the minor diagonal.

Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12} \tag{1.44}$$

The scalar value of the determinant of a 3×3 matrix is composed of the sum of six triple products which can be obtained from the augmented determinant:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{matrix} \tag{1.45}$$

The 3×3 determinant is augmented by repeating the first two columns of the determinant on the right-hand side of the determinant. Three triple products are formed, starting with the elements of the first row multiplied by the two remaining elements on the right-

downward-sloping diagonals. Three more triple products are formed, starting with the elements of the third row multiplied by the two remaining elements on the right-upward-sloping diagonals. The value of the determinant is the sum of the first three triple products minus the sum of the last three triple products. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12} \tag{1.46}$$

Example 1.3. Evaluation of a 3×3 determinant by the diagonal method.

Let's evaluate the determinant of the coefficient matrix of Eq. (1.2) by the diagonal method. Thus,

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \tag{1.47}$$

The augmented determinant is

$$\begin{vmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{vmatrix} \begin{matrix} 80 & -20 \\ -20 & 40 \\ -20 & -20 \end{matrix} \tag{1.48}$$

Applying Eq. (1.46) yields

$$\begin{aligned} \det(\mathbf{A}) = |\mathbf{A}| &= (80)(40)(130) + (-20)(-20)(-20) + (-20)(-20)(-20) \\ &\quad - (-20)(40)(-20) - (-20)(-20)(80) \\ &\quad - (130)(-20)(-20) = 416{,}000 - 8{,}000 - 8{,}000 \\ &\quad - 16{,}000 - 32{,}000 - 52{,}000 = 300{,}000 \end{aligned} \tag{1.49}$$

The diagonal method of evaluating determinants applies only to 2×2 and 3×3 determinants. It is incorrect for 4×4 or larger determinants. In general, the expansion of an $n \times n$ determinant is the sum of all possible products formed by choosing one and only one element from each row and each column of the determinant, with a plus or minus sign determined by the number of permutations of the row and column elements. One formal procedure for evaluating determinants is called *expansion by minors*, or the *method of cofactors*. In this procedure there are $n!$ products to be summed, where each product has n elements. Thus, the expansion of a 10×10 determinant requires the summation of 10! products ($10! = 3{,}628{,}800$), where each product involves 9 multiplications (the product of 10 elements). This is a total of 32,659,000 multiplications and 3,627,999 additions, not counting the work needed to keep track of the signs. Consequently, the evaluation of determinants by the method of cofactors is impractical, except for very small determinants.

Although the method of cofactors is not recommended for anything larger than a 4×4 determinant, it is useful to understand the concepts involved. The *minor* $M_{i,j}$ is the determinant of the $(n-1) \times (n-1)$ submatrix of the $n \times n$ matrix $\mathbf{A}$ obtained by deleting the ith row and the jth column. The cofactor $A_{i,j}$ associated with the minor $M_{i,j}$ is defined as

$$A_{i,j} = (-1)^{i+j} M_{i,j} \tag{1.50}$$

Using cofactors, the determinant of matrix **A** is the sum of the products of the elements of any row or column, multiplied by their corresponding cofactors. Thus, expanding across any fixed row i yields

$$\det(\mathbf{A}) = |\mathbf{A}| = \sum_{j=1}^{n} a_{i,j}A_{i,j} = \sum_{j=1}^{n}(-1)^{i+j}a_{i,j}M_{i,j} \tag{1.51}$$

Alternatively, expanding down any fixed column j yields

$$\det(\mathbf{A}) = |\mathbf{A}| = \sum_{i=1}^{n} a_{i,j}A_{i,j} = \sum_{i=1}^{n}(-1)^{i+j}a_{i,j}M_{i,j} \tag{1.52}$$

Each cofactor expansion reduces the order of the determinant by one, so there are n determinants of order $n-1$ to evaluate. By repeated application, the cofactors are eventually reduced to 3×3 determinants which can be evaluated by the diagonal method. The amount of work can be reduced by choosing the expansion row or column with as many zeros as possible.

Example 1.4. Evaluation of a 3×3 determinant by the cofactor method.

Let's rework Example 1.3 using the cofactor method. Recall Eq. (1.47):

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \tag{1.53}$$

Evaluate $|\mathbf{A}|$ by expanding across the first row. Thus,

$$|\mathbf{A}| = (80)\begin{vmatrix} 40 & -20 \\ -20 & 130 \end{vmatrix} - (-20)\begin{vmatrix} -20 & -20 \\ -20 & 130 \end{vmatrix} + (-20)\begin{vmatrix} -20 & 40 \\ -20 & -20 \end{vmatrix} \tag{1.54}$$

$$|\mathbf{A}| = 80(5200 + 400) - (-20)(-2600 + 400) + (-20)(400 + 800)$$

$$= 384000 - 60000 - 24000 = 300000 \tag{1.55}$$

If the value of the determinant of a matrix is zero, the matrix is said to be *singular*. A *nonsingular matrix* has a determinant that is nonzero. If any row or column of a matrix has all zero elements, that matrix is singular.

The determinant of a triangular matrix, either upper or lower triangular, is the product of the elements on the major diagonal. It is possible to transform any nonsingular matrix into a triangular matrix in such a way that the value of the determinant is either unchanged or changed in a well-defined way. That procedure is presented in Section 1.3.6. The value of the determinant can then be evaluated quite easily as the product of the elements on the major diagonal.

1.3 DIRECT ELIMINATION METHODS

There are a number of methods for the direct solution of systems of linear algebraic equations. One of the more well-known methods is Cramer's rule, which requires the evaluation of numerous determinants. Cramer's rule is highly inefficient, and thus not recommended. More efficient methods, based on the elimination concept, are recom-

mended. Both Cramer's rule and elimination methods are presented in this section. After presenting *Cramer's rule*, the elimination concept is applied to develop Gauss elimination, Gauss-Jordan elimination, matrix inversion, and determinant evaluation. These concepts are extended to LU factorization and tridiagonal systems of equations in Sections 1.4 and 1.5, respectively.

1.3.1. Cramer's Rule

Although it is not an elimination method, *Cramer's rule* is a direct method for solving systems of linear algebraic equations. Consider the system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, which represents n equations. Cramer's rule states that the solution for x_j $(j = 1, \ldots, n)$ is given by

$$x_j = \frac{\det(\mathbf{A}^j)}{\det(\mathbf{A})} \qquad (j = 1, \ldots, n) \tag{1.56}$$

where $\mathbf{A}^j$ is the $n \times n$ matrix obtained by replacing column j in matrix $\mathbf{A}$ by the column vector $\mathbf{b}$. For example, consider the system of two linear algebraic equations:

$$a_{11}x_1 + a_{12}x_2 = b_1 \tag{1.57a}$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{1.57b}$$

Applying Cramer's rule yields

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} \quad \text{and} \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} \tag{1.58}$$

The determinants in Eqs. (1.58) can be evaluated by the diagonal method described in Section 1.2.4.

For systems containing more than three equations, the diagonal method presented in Section 1.2.4 does not work. In such cases, the method of cofactors presented in Section 1.2.4 could be used. The number of multiplications and divisions N required by the method of cofactors is $N = (n-1)(n+1)!$. For a relatively small system of 10 equations (i.e., $n = 10$), $N = 360{,}000{,}000$, which is an enormous number of calculations. For $n = 100$, $N = 10^{157}$, which is obviously ridiculous. The preferred method for evaluating determinants is the elimination method presented in Section 1.3.6. The number of multiplications and divisions required by the elimination method is approximately $N = n^3 + n^2 - n$. Thus, for $n = 10$, $N = 1090$, and for $n = 100$, $N = 1{,}009{,}900$. Obviously, the elimination method is preferred.

Example 1.5. Cramer's rule.

Let's illustrate Cramer's rule by solving Eq. (1.2). Thus,

$$80x_1 - 20x_2 - 20x_3 = 20 \tag{1.59a}$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \tag{1.59b}$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \tag{1.59c}$$

First, calculate det(**A**). From Example 1.4,

$$\det(\mathbf{A}) = \begin{vmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{vmatrix} = 300{,}000 \tag{1.60}$$

Next, calculate det($\mathbf{A}^1$), det($\mathbf{A}^2$), and det($\mathbf{A}^3$). For det($\mathbf{A}^1$),

$$\det(\mathbf{A}^1) = \begin{vmatrix} 20 & -20 & -20 \\ 20 & 40 & -20 \\ 20 & -20 & 130 \end{vmatrix} = 180{,}000 \tag{1.61}$$

In a similar manner, det($\mathbf{A}^2$) = 300,000 and det($\mathbf{A}^3$) = 120,000. Thus,

$$x_1 = \frac{\det(\mathbf{A}^1)}{\det(\mathbf{A})} = \frac{180{,}000}{300{,}000} = 0.60 \qquad x_2 = \frac{300{,}000}{300{,}000} = 1.00 \qquad x_3 = \frac{120{,}000}{300{,}000} = 0.40 \tag{1.62}$$

1.3.2. Elimination Methods

Elimination methods solve a system of linear algebraic equations by solving one equation, say the first equation, for one of the unknowns, say x_1, in terms of the remaining unknowns, x_2 to x_n, then substituting the expression for x_1 into the remaining $n-1$ equations to determine $n-1$ equations involving x_2 to x_n. This elimination procedure is performed $n-1$ times until the last step yields an equation involving only x_n. This process is called *elimination*.

The value of x_n can be calculated from the final equation in the elimination procedure. Then x_{n-1} can be calculated from modified equation $n-1$, which contains only x_n and x_{n-1}. Then x_{n-2} can be calculated from modified equation $n-2$, which contains only x_n, x_{n-1}, and x_{n-2}. This procedure is performed $n-1$ times to calculate x_{n-1} to x_1. This process is called *back substitution*.

1.3.2.1. Row Operations

The elimination process employs the row operations presented in Section 1.2.3, which are repeated below:

1. Any row (equation) may be multiplied by a constant (scaling).
2. The order of the rows (equations) may be interchanged (pivoting).
3. Any row (equation) can be replaced by a weighted linear combination of that row (equation) with any other row (equation) (elimination).

These row operations, which change the values of the elements of matrix **A** and **b**, do not change the solution **x** to the system of equations.

The first row operation is used to scale the equations, if necessary. The second row operation is used to prevent divisions by zero and to reduce round-off errors. The third row operation is used to implement the systematic elimination process described above.

1.3.2.2. Elimination

Let's illustrate the elimination method by solving Eq. (1.2). Thus,

$$80x_1 - 20x_2 - 20x_3 = 20 \tag{1.63a}$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \tag{1.63b}$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \tag{1.63c}$$

Solve Eq. (1.63a) for x_1. Thus,

$$x_1 = [20 - (-20)x_2 - (-20)x_3]/80 \tag{1.64}$$

Substituting Eq. (1.64) into Eq. (1.63b) gives

$$-20\{[20 - (-20)x_2 - (-20)x_3]/80\} + 40x_2 - 20x_3 = 20 \tag{1.65}$$

which can be simplified to give

$$35x_2 - 25x_3 = 25 \tag{1.66}$$

Substituting Eq. (1.64) into Eq. (1.63c) gives

$$-20\{[20 - (-20)x_2 - (-20)x_3]/80\} - 20x_2 + 130x_3 = 20 \tag{1.67}$$

which can be simplified to give

$$-25x_2 + 125x_3 = 25 \tag{1.68}$$

Next solve Eq. (1.66) for x_2. Thus,

$$x_2 = [25 - (-25)x_3]/35 \tag{1.69}$$

Substituting Eq. (1.69) into Eq. (1.68) yields

$$-25\{[25 - (-25)x_3]/35\} + 125x_3 = 25 \tag{1.70}$$

which can be simplified to give

$$\tfrac{750}{7}x_3 = \tfrac{300}{7} \tag{1.71}$$

Thus, Eq. (1.63) has been reduced to the upper triangular system

$$80x_1 - 20x_2 - 20x_3 = 20 \tag{1.72a}$$

$$35x_2 - 25x_3 = 25 \tag{1.72b}$$

$$\tfrac{750}{7}x_3 = \tfrac{300}{7} \tag{1.72c}$$

which is equivalent to the original equation, Eq. (1.63). This completes the elimination process.

1.3.2.3. Back Substitution

The solution to Eq. (1.72) is accomplished easily by *back substitution*. Starting with Eq. (1.72c) and working backward yields

$$x_3 = 300/750 = 0.40 \tag{1.73a}$$

$$x_2 = [25 - (-25)(0.40)]/35 = 1.00 \tag{1.73b}$$

$$x_1 = [20 - (-20)(1.00) - (-20)(0.40)]/80 = 0.60 \tag{1.73c}$$

Example 1.6. Elimination.

Let's solve Eq. (1.2) by elimination. Recall Eq. (1.2):

$$80x_1 - 20x_2 - 20x_3 = 20 \tag{1.74a}$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \tag{1.74b}$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \tag{1.74c}$$

Elimination involves normalizing the equation above the element to be eliminated by the element immediately above the element to be eliminated, which is called the *pivot element*, multiplying the normalized equation by the element to be eliminated, and subtracting the result from the equation containing the element to be eliminated. This process systematically eliminates terms below the major diagonal, column by column, as illustrated below. The notation $R_i - (\text{em})R_j$ next to the ith equation indicates that the ith equation is to be replaced by the ith equation minus em times the jth equation, where the elimination multiplier, em, is the quotient of the element to be eliminated and the pivot element.

For example, $R_2 - (-20/40)R_1$ beside Eq. (1.75.2) below means replace Eq. (1.75.2) by Eq. (1.75.2) $-$ $(-20/40) \times$ Eq. (1.75.1). The elimination multiplier, $\text{em} = (-20/40)$, is chosen to eliminate the first coefficient in Eq. (1.75.2). All of the coefficients below the major diagonal in the first column are eliminated by linear combinations of each equation with the first equation. Thus,

$$\left[\begin{array}{rl} 80x_1 - 20x_2 - 20x_3 = 20 & \\ -20x_1 + 40x_2 - 20x_3 = 20 & R_2 - (-20/80)R_1 \\ -20x_1 - 20x_2 + 135x_3 = 20 & R_3 - (-20/80)R_1 \end{array}\right] \begin{array}{l} \\ \text{(1.75.2)} \\ \text{(1.75.3)} \end{array} \tag{1.75.1}$$

The result of this first elimination step is presented in Eq. (1.76), which also shows the elimination operation for the second elimination step. Next the coefficients below the major diagonal in the second column are eliminated by linear combinations with the second equation. Thus,

$$\left[\begin{array}{rl} 80x_1 - 20x_2 - 20x_3 = 20 & \\ 0x_1 + 35x_2 - 25x_3 = 25 & \\ 0x_1 - 25x_2 + 125x_3 = 25 & R_3 - (-25/35)R_2 \end{array}\right] \tag{1.76}$$

The result of the second elimination step is presented in Eq. (1.77):

$$\left[\begin{array}{l} 80x_1 - 20x_2 - 20x_3 = 20 \\ 0x_1 + 35x_2 - 25x_3 = 25 \\ 0x_1 + 0x_2 + 750/7x_3 = 300/7 \end{array}\right] \tag{1.77}$$

This process is continued until all the coefficients below the major diagonal are eliminated. In the present example with three equations, this process is now complete, and Eq. (1.77) is the final result. This is the process of elimination.

At this point, the last equation contains only one unknown, x_3 in the present example, which can be solved for. Using that result, the next to last equation can be solved

for x_2. Using the results for x_3 and x_2, the first equation can be solved for x_1. This is the back substitution process. Thus,

$$x_3 = 300/750 = 0.40 \tag{1.78a}$$

$$x_2 = [25 - (-25)(0.40)]/35 = 1.00 \tag{1.78b}$$

$$x_1 = [20 - (-20)(1.00) - (-20)(0.40)/80 = 0.60 \tag{1.78c}$$

The extension of the elimination procedure to n equations is straightforward.

1.3.2.4. Simple Elimination

The elimination procedure illustrated in Example 1.6 involves manipulation of the coefficient matrix **A** and the nonhomogeneous vector **b**. Components of the **x** vector are fixed in their locations in the set of equations. As long as the columns are not interchanged, column j corresponds to x_j. Consequently, the x_j notation does not need to be carried throughout the operations. Only the numerical elements of **A** and **b** need to be considered. Thus, the elimination procedure can be simplified by augmenting the **A** matrix with the **b** vector and performing the row operations on the elements of the augmented **A** matrix to accomplish the elimination process, then performing the back substitution process to determine the solution vector. This simplified elimination procedure is illustrated in Example 1.7.

Example 1.7. Simple elimination.

Let's rework Example 1.6 using simple elimination. From Example 1.6, the **A** matrix augmented by the **b** vector is

$$[\mathbf{A} \mid \mathbf{b}] = \begin{bmatrix} 80 & -20 & -20 \mid 20 \\ -20 & 40 & -20 \mid 20 \\ -20 & -20 & 130 \mid 20 \end{bmatrix} \tag{1.79}$$

Performing the row operations to accomplish the elimination process yields:

$$\begin{bmatrix} 80 & -20 & -20 \mid 20 \\ -20 & 40 & -20 \mid 20 \\ -20 & -20 & 130 \mid 20 \end{bmatrix} \begin{matrix} \\ R_2 - (-20/80)R_1 \\ R_3 - (-20/80)R_1 \end{matrix} \tag{1.80}$$

$$\begin{bmatrix} 80 & -20 & -20 \mid 20 \\ 0 & 35 & -25 \mid 25 \\ 0 & -25 & 125 \mid 25 \end{bmatrix} R_3 - (-25/35)R_2 \tag{1.81}$$

$$\begin{bmatrix} 80 & -20 & -20 \mid 20 \\ 0 & 35 & -25 \mid 25 \\ 0 & 0 & 750/7 \mid 300/7 \end{bmatrix} \rightarrow \begin{matrix} x_1 = [20 - (-20)(1.00) - (-20)(0.40)]/80 \\ \quad = 0.60 \\ x_2 = [25 - (-25)(0.4)]/35 = 1.00 \\ x_3 = 300/750 = 0.40 \end{matrix} \tag{1.82}$$

The back substitution step is presented beside the triangularized augmented **A** matrix.

1.3.2.5. Multiple b Vectors

If more than one **b** vector is to be considered, the **A** matrix is simply augmented by all of the **b** vectors simultaneously. The elimination process is then applied to the multiply augmented **A** matrix. Back substitution is then applied one column at a time to the modified **b** vectors. A more versatile procedure based on matrix factorization is presented in Section 1.4.

Example 1.8. Simple elimination for multiple b vectors.

Consider the system of equations presented in Example 1.7 with two **b** vectors, $\mathbf{b}_1^T = [20 \quad 20 \quad 20]$ and $\mathbf{b}_2^T = [20 \quad 10 \quad 20]$. The doubly augmented **A** matrix is

$$[\mathbf{A} \mid \mathbf{b}_1 \quad \mathbf{b}_2] = \left[\begin{array}{rrr|c|c} 80 & -20 & -20 & 20 & 20 \\ -20 & 40 & -20 & 20 & 10 \\ -20 & -20 & 130 & 20 & 20 \end{array}\right] \tag{1.83}$$

Performing the elimination process yields

$$\left[\begin{array}{rrr|rr} 80 & -20 & -20 & 20 & 20 \\ 0 & 35 & -25 & 25 & 15 \\ 0 & 0 & 750/7 & 300/7 & 250/7 \end{array}\right] \tag{1.84}$$

Performing the back substitution process one column at a time yields

$$\mathbf{x}_1 = \begin{bmatrix} 0.60 \\ 1.00 \\ 0.40 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_2 = \begin{bmatrix} 1/2 \\ 2/3 \\ 1/3 \end{bmatrix} \tag{1.85}$$

1.3.2.6. Pivoting

The element on the major diagonal is called the *pivot* element. The elimination procedure described so far fails immediately if the first pivot element a_{11} is zero. The procedure also fails if any subsequent pivot element $a_{i,i}$ is zero. Even though there may be no zeros on the major diagonal in the original matrix, the elimination process may create zeros on the major diagonal. The simple elimination procedure described so far must be modified to avoid zeros on the major diagonal. This result can be accomplished by rearranging the equations, by interchanging equations (rows) or variables (columns), before each elimination step to put the element of largest magnitude on the diagonal. This process is called *pivoting*. Interchanging both rows and columns is called *full pivoting*. Full pivoting is quite complicated, and thus it is rarely used. Interchanging only rows is called *partial pivoting*. Only partial pivoting is considered in this book.

Pivoting eliminates zeros in the pivot element locations during the elimination process. Pivoting also reduces round-off errors, since the pivot element is a divisor during the elimination process, and division by large numbers introduces smaller round-off errors than division by small numbers. When the procedure is repeated, round-off errors can compound. This problem becomes more severe as the number of equations is increased.

Example 1.9. Elimination with pivoting to avoid zero pivot elements.

Use simple elimination with partial pivoting to solve the following system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} 0 & 2 & 1 \\ 4 & 1 & -1 \\ -2 & 3 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \\ 5 \end{bmatrix} \tag{1.86}$$

Let's apply the elimination procedure by augmenting **A** with **b**. The first pivot element is zero, so pivoting is required. The largest number (in magnitude) in the first column under the pivot element occurs in the second row. Thus, interchanging the first and second rows and evaluating the elimination multipliers yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 2 & 1 & 5 \\ -2 & 3 & -3 & 5 \end{array}\right] \begin{matrix} \\ R_2 - (0/4)R_1 \\ R_3 - (-2/4)R_1 \end{matrix} \tag{1.87}$$

Performing the elimination operations yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 2 & 1 & 5 \\ 0 & 7/2 & -7/2 & 7/2 \end{array}\right] \tag{1.88}$$

Although the pivot element in the second row is not zero, it is not the largest element in the second column underneath the pivot element. Thus, pivoting is called for again. Note that pivoting is based only on the rows below the pivot element. The rows above the pivot element have already been through the elimination process. Using one of the rows above the pivot element would destroy the elimination already accomplished. Interchanging the second and third rows and evaluating the elimination multiplier yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 7/2 & -7/2 & 7/2 \\ 0 & 2 & 1 & 5 \end{array}\right] \begin{matrix} \\ \\ R_3 - (4/7)R_2 \end{matrix} \tag{1.89}$$

Performing the elimination operation yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 7/2 & -7/2 & 7/2 \\ 0 & 0 & 3 & 3 \end{array}\right] \begin{matrix} x_1 = -1 \\ \rightarrow\ x_2 = 2 \\ x_3 = 1 \end{matrix} \tag{1.90}$$

The back substitution results are presented beside the triangularized augmented **A** matrix.

1.3.2.7. Scaling

The elimination process described so far can incur significant round-off errors when the magnitudes of the pivot elements are smaller than the magnitudes of the other elements in the equations containing the pivot elements. In such cases, scaling is employed to select the pivot elements. After pivoting, elimination is applied to the original equations. Scaling is employed only to select the pivot elements.

Scaled pivoting is implemented as follows. Before elimination is applied to the first column, all of the elements in the first column are scaled (i.e., normalized) by the largest elements in the corresponding rows. Pivoting is implemented based on the scaled elements

in the first column, and elimination is applied to obtain zero elements in the first column below the pivot element. Before elimination is applied to the second column, all of the elements from 2 to n in column 2 are scaled, pivoting is implemented, and elimination is applied to obtain zero elements in column 2 below the pivot element. The procedure is applied to the remaining rows 3 to $n-1$. Back substitution is then applied to obtain $\mathbf{x}$.

Example 1.10. Elimination with scaled pivoting to reduce round-off errors.

Let's investigate the advantage of scaling by solving the following linear system:

$$\begin{bmatrix} 3 & 2 & 105 \\ 2 & -3 & 103 \\ 1 & 1 & 3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 104 \\ 98 \\ 3 \end{bmatrix} \tag{1.91}$$

which has the exact solution $x_1 = -1.0$, $x_2 = 1.0$, and $x_3 = 1.0$. To accentuate the effects of round-off, carry only three significant figures in the calculations. For the first column, pivoting does not appear to be required. Thus, the augmented $\mathbf{A}$ matrix and the first set of row operations are given by

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 2 & -3 & 103 & 98 \\ 1 & 1 & 3 & 3 \end{array}\right] \begin{array}{l} \\ R_2 - (0.667)R_1 \\ R_3 - (0.333)R_1 \end{array} \tag{1.92}$$

which gives

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 0 & -4.33 & 33.0 & 28.6 \\ 0 & 0.334 & -32.0 & -31.6 \end{array}\right] \begin{array}{l} \\ \\ R_3 - (-0.0771)R_2 \end{array} \tag{1.93}$$

Pivoting is not required for the second column. Performing the elimination indicated in Eq. (1.93) yields the triangularized matrix

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 0 & -4.33 & 33.0 & 28.9 \\ 0 & 0 & -29.5 & -29.4 \end{array}\right] \tag{1.94}$$

Performing back substitution yields $x_3 = 0.997$, $x_2 = 0.924$, and $x_1 = -0.844$, which does not agree very well with the exact solution $x_3 = 1.0$, $x_2 = 1.0$, and $x_1 = -1.0$. Round-off errors due to the three-digit precision have polluted the solution.

The effects of round-off can be reduced by scaling the equations before pivoting. Since scaling itself introduces round-off, it should be used only to determine if pivoting is required. All calculations should be made with the original unscaled equations.

Let's rework the problem using scaling to determine if pivoting is required. The first step in the elimination procedure eliminates all the elements in the first column under element a_{11}. Before performing that step, let's scale all the elements in column 1 by the largest element in each row. The result is

$$\mathbf{a}_1 = \begin{bmatrix} 3/105 \\ 2/103 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 0.0286 \\ 0.0194 \\ 0.3333 \end{bmatrix} \tag{1.95}$$

where the notation $\mathbf{a}_1$ denotes the column vector consisting of the scaled elements from the first column of matrix $\mathbf{A}$. The third element of $\mathbf{a}_1$ is the largest element in $\mathbf{a}_1$, which

indicates that rows 1 and 3 of matrix A should be interchanged. Thus, Eq. (1.91), with the elimination multipliers indicated, becomes

$$\left[\begin{array}{rrr|r} 1 & 1 & 3 & 3 \\ 2 & -3 & 103 & 98 \\ 3 & 2 & 105 & 104 \end{array}\right] \begin{array}{l} \\ R_2 - (2/1)R_1 \\ R_3 - (3/1)R_1 \end{array} \tag{1.96}$$

Performing the elimination and indicating the next elimination multiplier yields

$$\left[\begin{array}{rrr|r} 1 & 1 & 3 & 3 \\ 0 & -5 & 97 & 92 \\ 0 & -1 & 96 & 95 \end{array}\right] \begin{array}{l} \\ \\ R_3 - (1/5)R_2 \end{array} \tag{1.97}$$

Scaling the second and third elements of column 2 gives

$$\mathbf{a}_2 = \begin{bmatrix} - \\ -5/97 \\ -1/96 \end{bmatrix} = \begin{bmatrix} - \\ -0.0516 \\ -0.0104 \end{bmatrix} \tag{1.98}$$

Consequently, pivoting is not indicated. Performing the elimination indicated in Eq. (1.97) yields

$$\left[\begin{array}{rrr|r} 1.0 & 1.0 & 3.0 & 3.0 \\ 0.0 & -5.0 & 97.0 & 92.0 \\ 0.0 & 0.0 & 76.6 & 76.6 \end{array}\right] \tag{1.99}$$

Solving Eq. (1.99) by back substitution yields $x_1 = 1.00$, $x_2 = 1.00$, and $x_3 = -1.00$, which is the exact solution. Thus, scaling to determine the pivot element has eliminated the round-off error in this simple example.

1.3.3. Gauss Elimination

The elimination procedure described in the previous section, including scaled pivoting, is commonly called *Gauss elimination*. It is the most important and most useful direct elimination method for solving systems of linear algebraic equations. The Gauss-Jordan method, the matrix inverse method, the LU factorization method, and the Thomas algorithm are all modifications or extensions of the Gauss elimination method. Pivoting is an essential element of Gauss elimination. In cases where all of the elements of the coefficient matrix A are the same order of magnitude, scaling is not necessary. However, pivoting to avoid zero pivot elements is always required. Scaled pivoting to decrease round-off errors, while very desirable in general, can be omitted at some risk to the accuracy of the solution. When performing Gauss elimination by hand, decisions about pivoting can be made on a case by case basis. When writing a general-purpose computer program to apply Gauss elimination to arbitrary systems of equations, however, scaled pivoting is an absolute necessity. Example 1.10 illustrates the complete Gauss elimination algorithm.

When solving large systems of linear algebraic equations on a computer, the pivoting step is generally implemented by simply keeping track of the order of the rows as they are interchanged without actually interchanging rows, a time-consuming and unnecessary operation. This is accomplished by using an order vector **o** whose elements denote the order in which the rows of the coefficient matrix A and the right-hand-side

vector **b** are to be processed. When a row interchange is required, instead of actually interchanging the two rows of elements, the corresponding elements of the order vector are interchanged. The rows of the **A** matrix and the **b** vector are processed in the order indicated by the order vector **o** during both the elimination step and the back substitution step.

As an example, consider the second part of Example 1.10. The order vector has the initial value $\mathbf{o}^T = [1 \quad 2 \quad 3]$. After scaling, rows 1 and 3 are to be interchanged. Instead of actually interchanging these rows as done in Example 1.10, the corresponding elements of the order vector are changed to yield $\mathbf{o}^T = [3 \quad 2 \quad 1]$. The first elimination step then uses the third row to eliminate x_1 from the second and first rows. Pivoting is not required for the second elimination step, so the order vector is unchanged, and the second row is used to eliminate x_2 from the first row. Back substitution is then performed in the reverse order of the order vector, **o**, that is, in the order 1, 2, 3. This procedure saves computer time for large systems of equations, but at the expense of a slightly more complicated program.

The number of multiplications and divisions required for Gauss elimination is approximately $N = (n^3/3 - n/3)$ for matrix **A** and n^2 for each **b**. For $n = 10$, $N = 430$, and for $n = 100$, $N = 343{,}300$. This is a considerable reduction compared to Cramer's rule.

The Gauss elimination procedure, in a format suitable for programming on a computer, is summarized as follows:

1. Define the $n \times n$ coefficient matrix **A**, the $n \times 1$ column vector **b**, and the $n \times 1$ order vector **o**.
2. Starting with column 1, scale column k ($k = 1, 2, \ldots, n-1$) and search for the element of largest magnitude in column k and pivot (interchange rows) to put that coefficient into the $a_{k,k}$ pivot position. This step is actually accomplished by interchanging the corresponding elements of the $n \times 1$ order vector **o**.
3. For column k ($k = 1, 2, \ldots, n-1$), apply the elimination procedure to rows i ($i = k+1, k+2, \ldots, n$) to create zeros in column k below the pivot element, $a_{k,k}$. Do not actually calculate the zeros in column k. In fact, storing the elimination multipliers, $\text{em} = (a_{i,k}/a_{k,k})$, in place of the eliminated elements, $a_{i,k}$, creates the Doolittle LU factorization presented in Section 1.4. Thus,

$$a_{i,j} = a_{i,j} - \left(\frac{a_{i,k}}{a_{k,k}}\right) a_{k,j} \qquad (i,j = k+1, k+2, \ldots, n) \tag{1.100a}$$

$$b_i = b_i - \left(\frac{a_{i,k}}{a_{k,k}}\right) b_k \qquad (i = k+1, k+2, \ldots, n) \tag{1.100b}$$

   After step 3 is applied to all k columns, ($k = 1, 2, \ldots, n-1$), the original **A** matrix is upper triangular.
4. Solve for **x** using back substitution. If more than one **b** vector is present, solve for the corresponding **x** vectors one at a time. Thus,

$$x_n = \frac{b_n}{a_{n,n}} \tag{1.101a}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} x_j}{a_{i,i}} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.101b}$$

1.3.4. Gauss-Jordan Elimination

Gauss-Jordan elimination is a variation of Gauss elimination in which the elements above the major diagonal are eliminated (made zero) as well as the elements below the major diagonal. The **A** matrix is transformed to a diagonal matrix. The rows are usually scaled to yield unity diagonal elements, which transforms the **A** matrix to the identity matrix, **I**. The transformed **b** vector is then the solution vector **x**. Gauss-Jordan elimination can be used for single or multiple b vectors.

The number of multiplications and divisions for Gauss-Jordan elimination is approximately $N = (n^3/2 - n/2) + n^2$, which is approximately 50 percent larger than for Gauss elimination. Consequently, Gauss elimination is preferred.

Example 1.11. Gauss-Jordan elimination.

Let's rework Example 1.7 using simple Gauss-Jordan elimination, that is, elimination without pivoting. The augmented **A** matrix is [see Eq. (1.79)]

$$\left[\begin{array}{rrr|r} 80 & -20 & -20 & 20 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array}\right] R_1/80 \tag{1.102}$$

Scaling row 1 to give $a_{11} = 1$ gives

$$\left[\begin{array}{rrr|r} 1 & -1/4 & -1/4 & 1/4 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array}\right] \begin{array}{l} \\ R_2 - (-20)R_1 \\ R_3 - (-20)R_1 \end{array} \tag{1.103}$$

Applying elimination below row 1 yields

$$\left[\begin{array}{rrr|r} 1 & -1/4 & -1/4 & 1/4 \\ 0 & 35 & -25 & 25 \\ 0 & -25 & 125 & 25 \end{array}\right] R_2/35 \tag{1.104}$$

Scaling row 2 to give $a_{22} = 1$ gives

$$\left[\begin{array}{rrr|r} 1 & -1/4 & -1/4 & 1/4 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & -25 & 125 & 25 \end{array}\right] \begin{array}{l} R_1 - (-1/4)R_2 \\ \\ R_3 - (-25)R_2 \end{array} \tag{1.105}$$

Applying elimination both above and below row 2 yields

$$\left[\begin{array}{rrr|r} 1 & 0 & -3/7 & 3/7 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & 0 & 750/7 & 300/7 \end{array}\right] R_3/(750/7) \tag{1.106}$$

Scaling row 3 to give $a_{33} = 1$ gives

$$\left[\begin{array}{rrr|r} 1 & 0 & -3/7 & 3/7 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & 0 & 1 & 215 \end{array}\right] \begin{array}{l} R_1 - (-3/7)R_3 \\ R_2 - (-5/7)R_3 \\ \\ \end{array} \tag{1.107}$$

Applying elimination above row 3 completes the process.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.60 \\ 0 & 1 & 0 & 1.00 \\ 0 & 0 & 1 & 0.40 \end{array}\right] \tag{1.108}$$

The **A** matrix has been transformed to the identity matrix **I** and the **b** vector has been transformed to the solution vector, **x**. Thus, $\mathbf{x}^T = [0.60 \quad 1.00 \quad 0.40]$.

The inverse of a square matrix **A** is the matrix $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Gauss-Jordan elimination can be used to evaluate the inverse of matrix **A** by augmenting **A** with the identity matrix **I** and applying the Gauss-Jordan algorithm. The transformed **A** matrix is the identity matrix **I**, and the transformed identity matrix is the matrix inverse, $\mathbf{A}^{-1}$. Thus, applying Gauss-Jordan elimination yields

$$\boxed{[\mathbf{A} \mid \mathbf{I}] \rightarrow [\mathbf{I} \mid \mathbf{A}^{-1}]} \tag{1.109}$$

The Gauss-Jordan elimination procedure, in a format suitable for programming on a computer, can be developed to solve Eq. (1.109) by modifying the Gauss elimination procedure presented in Section 1.3.C. Step 1 is changed to augment the $n \times n$ **A** matrix with the $n \times n$ identity matrix, **I**. Steps 2 and 3 of the procedure are the same. Before performing Step 3, the pivot element is scaled to unity by dividing all elements in the row by the pivot element. Step 3 is expanded to perform elimination above the pivot element as well as below the pivot element. At the conclusion of step 3, the **A** matrix has been transformed to the identity matrix, **I**, and the original identity matrix, **I**, has been transformed to the matrix inverse, $\mathbf{A}^{-1}$.

Example 1.12. Matrix inverse by Gauss-Jordan elimination.

Let's evaluate the inverse of matrix **A** presented in Example 1.7. First, augment matrix **A** with the identity matrix, **I**. Thus,

$$[\mathbf{A} \mid \mathbf{I}] = \left[\begin{array}{ccc|ccc} 80 & -20 & -20 & 1 & 0 & 0 \\ -20 & 40 & -20 & 0 & 1 & 0 \\ -20 & -20 & 130 & 0 & 0 & 1 \end{array}\right] \tag{1.110}$$

Performing Gauss-Jordan elimination transforms Eq. (1.110) to

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 2/125 & 1/100 & 1/250 \\ 0 & 1 & 0 & 1/100 & 1/30 & 1/150 \\ 0 & 0 & 1 & 1/250 & 1/150 & 7/750 \end{array}\right] \tag{1.111}$$

from which

$$\mathbf{A}^{-1} = \left[\begin{array}{ccc} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{array}\right] = \left[\begin{array}{ccc} 0.016000 & 0.010000 & 0.004000 \\ 0.010000 & 0.033333 & 0.006667 \\ 0.004000 & 0.006667 & 0.009333 \end{array}\right] \tag{1.112}$$

Multiplying **A** times $\mathbf{A}^{-1}$ yields the identity matrix **I**, thus verifying the computations.

1.3.5. The Matrix Inverse Method

Systems of linear algebraic equations can be solved using the matrix inverse, $\mathbf{A}^{-1}$. Consider the general system of linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \tag{1.113}$$

Multiplying Eq. (1.113) by $\mathbf{A}^{-1}$ yields

$$\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{Ix} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{1.114}$$

from which

$$\boxed{\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}} \tag{1.115}$$

Thus, when the matrix inverse $\mathbf{A}^{-1}$ of the coefficient matrix $\mathbf{A}$ is known, the solution vector $\mathbf{x}$ is simply the product of the matrix inverse $\mathbf{A}^{-1}$ and the right-hand-side vector $\mathbf{b}$. Not all matrices have inverses. Singular matrices, that is, matrices whose determinant is zero, do not have inverses. The corresponding system of equations does not have a unique solution.

Example 1.13. The matrix inverse method.

Let's solve the linear system considered in Example 1.7 using the matrix inverse method. The matrix inverse $\mathbf{A}^{-1}$ of the coefficient matrix $\mathbf{A}$ for that linear system is evaluated in Example 1.12. Multiplying $\mathbf{A}^{-1}$ by the vector $\mathbf{b}$ from Example 1.7 gives

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} \tag{1.116}$$

Performing the matrix multiplication yields

$$x_1 = (2/125)(20) + (1/100)(20) + (1/250)(20) = 0.60 \tag{1.117a}$$
$$x_2 = (1/100)(20) + (1/30)(20) + (1/150)(20) = 1.00 \tag{1.117b}$$
$$x_3 = (1/250)(20) + (1/150)(20) + (7/750)(20) = 0.04 \tag{1.117c}$$

Thus, $x^T = [0.60 \quad 1.00 \quad 0.40]$.

1.3.6. Determinants

The evaluation of determinants by the cofactor method is discussed in Section 1.2.4 and illustrated in Example 1.4. Approximately $N = (n-1)n!$ multiplications are required to evaluate the determinant of an $n \times n$ matrix by the cofactor method. For $n = 10$, $N = 32{,}659{,}000$. Evaluation of the determinants of large matrices by the cofactor method is prohibitively expensive, if not impossible. Fortunately, determinants can be evaluated much more efficiently by a variation of the elimination method.

First, consider the matrix **A** expressed in upper triangular form:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} \tag{1.118}$$

Expanding the determinant of **A** by cofactors down the first column gives a_{11} times the $(n-1)\times(n-1)$ determinant having a_{22} as its first element in its first column, with the remaining elements in its first column all zero. Expanding that determinant by cofactors down its first column yields a_{22} times the $(n-2)\times(n-2)$ determinant having a_{33} as its first element in its first column with the remaining elements in its first column all zero. Continuing in this manner yields the result that the determinant of an upper triangular matrix (or a lower triangular matrix) is simply the product of the elements on the major diagonal. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \prod_{i=1}^{n} a_{i,i} \tag{1.119}$$

where the $\prod$ notation denotes the product of the $a_{i,i}$. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = a_{11}a_{22}\cdots a_{nn} \tag{1.120}$$

This result suggests the use of elimination to triangularize a general square matrix, then to evaluate its determinant using Eq. (1.119). This procedure works exactly as stated if no pivoting is used. When pivoting is used, the value of the determinant is changed, but in a predictable manner, so elimination can also be used with pivoting to evaluate determinants. The row operations must be modified as follows to use elimination for the evaluation of determinants.

1. Multiplying a row by a constant multiplies the determinant by that constant.
2. Interchanging any two rows changes the sign of the determinant. Thus, an even number of row interchanges does not change the sign of the determinant, whereas an odd number of row interchanges does change the sign of the determinant.
3. Any row may be added to the multiple of any other row without changing the value of the determinant.

The modified elimination method based on the above row operations is an efficient way to evaluate the determinant of a matrix. The number of multiplications required is approximately $N = n^3 + n^2 - n$, which is orders and orders of magnitude less effort than the $N = (n-1)n!$ multiplications required by the cofactor method.

Example 1.14. Evaluation of a 3×3 determinant by the elimination method.

Let's rework Example 1.4 using the elimination method. Recall Eq. (1.53):

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \tag{1.121}$$

From Example 1.7, after Gauss elimination, matrix **A** becomes

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \tag{1.122}$$

There are no row interchanges or multiplications of the matrix by scalars in this example. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = (80)(35)(750/7) = 300{,}000 \tag{1.123}$$

1.4 LU FACTORIZATION

Matrices (like scalars) can be *factored* into the product of two other matrices in an infinite number of ways. Thus,

$$\mathbf{A} = \mathbf{BC} \tag{1.124}$$

When **B** and **C** are lower triangular and upper triangular matrices, respectively, Eq. (1.124) becomes

$$\mathbf{A} = \mathbf{LU} \tag{1.125}$$

Specifying the diagonal elements of either **L** or **U** makes the factoring unique. The procedure based on unity elements on the major diagonal of **L** is called the *Doolittle method*. The procedure based on unity elements on the major diagonal of **U** is called the *Crout method*.

Matrix factoring can be used to reduce the work involved in Gauss elimination when multiple unknown **b** vectors are to be considered. In the Doolittle LU method, this is accomplished by defining the elimination multipliers, em, determined in the elimination step of Gauss elimination as the elements of the **L** matrix. The **U** matrix is defined as the upper triangular matrix determined by the elimination step of Gauss elimination. In this manner, multiple **b** vectors can be processed through the elimination step using the **L** matrix and through the back substitution step using the elements of the **U** matrix.

Consider the linear system, $\mathbf{Ax} = \mathbf{b}$. Let **A** be factored into the product **LU**, as illustrated in Eq. (1.125). The linear system becomes

$$\mathbf{LUx} = \mathbf{b} \tag{1.126}$$

Multiplying Eq. (1.126) by $\mathbf{L}^{-1}$ gives

$$\mathbf{L}^{-1}\mathbf{LUx} = \mathbf{IUx} = \mathbf{Ux} = \mathbf{L}^{-1}\mathbf{b} \tag{1.127}$$

The last two terms in Eq. (1.127) give

$$\mathbf{Ux} = \mathbf{L}^{-1}\mathbf{b} \tag{1.128}$$

Define the vector $\mathbf{b}'$ as follows:

$$\mathbf{b}' = \mathbf{L}^{-1}\mathbf{b} \tag{1.129}$$

Multiplying Eq. (1.129) by **L** gives

$$\mathbf{Lb}' = \mathbf{LL}^{-1}\mathbf{b} = \mathbf{Ib} = \mathbf{b} \tag{1.130}$$

Equating the first and last terms in Eq. (1.130) yields

$$\boxed{\mathbf{Lb}' = \mathbf{b}} \tag{1.131}$$

Substituting Eq. (1.129) into Eq. (1.128) yields

$$\boxed{\mathbf{Ux} = \mathbf{b}'} \tag{1.132}$$

Equation (1.131) is used to transform the **b** vector into the **b**′ vector, and Eq. (1.132) is used to determine the solution vector **x**. Since Eq. (1.131) is lower triangular, forward substitution (analogous to back substitution presented earlier) is used to solve for **b**′. Since Eq. (1.132) is upper triangular, back substitution is used to solve for **x**.

In the *Doolittle LU method*, the **U** matrix is the upper triangular matrix obtained by Gauss elimination. The **L** matrix is the lower triangular matrix containing the elimination multipliers, em, obtained in the Gauss elimination process as the elements below the diagonal, with unity elements on the major diagonal. Equation (1.131) applies the steps performed in the triangularization of **A** to **U** to the **b** vector to transform **b** to **b**′. Equation (1.132) is simply the back substitution step of the Gauss elimination method. Consequently, once **L** and **U** have been determined, any **b** vector can be considered at any later time, and the corresponding solution vector **x** can be obtained simply by solving Eqs. (1.131) and (1.132), in that order. The number of multiplicative operations required for each **b** vector is n^2.

Example 1.15. The Doolittle LU method.

Let's solve Example 1.7 using the Doolittle LU method. The first step is to determine the **L** and **U** matrices. The **U** matrix is simply the upper triangular matrix determined by the Gauss elimination procedure in Example 1.7. The **L** matrix is simply the record of the elimination multipliers, em, used to transform **A** to **U**. These multipliers are the numbers in parentheses in the row operations indicated in Eqs. (1.80) and (1.81) in Example 1.7. Thus, **L** and **U** are given by

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \tag{1.133}$$

Consider the first **b** vector from Example 1.8: $\mathbf{b}_1^T = [20 \quad 20 \quad 20]$. Equation (1.131) gives

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b_1' \\ b_2' \\ b_3' \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} \tag{1.134}$$

Performing forward substitution yields

$$b_1' = 20 \tag{1.135a}$$

$$b_2' = 20 - (-1/4)(20) = 25 \tag{1.135b}$$

$$b_3' = 20 - (-1/4)(20) - (-5/7)(25) = 300/7 \tag{1.135c}$$

The $\mathbf{b}'$ vector is simply the transformed $\mathbf{b}$ vector determined in Eq. (1.82). Equation (1.132) gives

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 25 \\ 300/7 \end{bmatrix} \tag{1.136}$$

Performing back substitution yields $x_1^T = [0.60 \quad 1.00 \quad 0.40]$. Repeating the process for $\mathbf{b}_2^T = [20 \quad 10 \quad 20]$ yields

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b_1' \\ b_2' \\ b_3' \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 20 \end{bmatrix} \begin{matrix} b_1' = 20 \\ b_2' = 15 \\ b_3' = 250/7 \end{matrix} \tag{1.137}$$

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 15 \\ 250/7 \end{bmatrix} \begin{matrix} x_1 = 1/2 \\ x_2 = 2/3 \\ x_3 = 1/3 \end{matrix} \tag{1.138}$$

When pivoting is used with **LU** factorization, it is necessary to keep track of the row order, for example, by an order vector **o**. When the rows of **A** are interchanged during the elimination process, the corresponding elements of the order vector **o** are interchanged. When a new **b** vector is considered, it is processed in the order corresponding to the elements of the order vector **o**.

The major advantage of LU factorization methods is their efficiency when multiple unknown **b** vectors must be considered. The number of multiplications and divisions required by the complete Gauss elimination method is $N = (n^3/3 - n/3) + n^2$. The forward substitution step required to solve $\mathbf{Lb}' = \mathbf{b}$ requires $N = n^2/2 - n/2$ multiplicative operations, and the back substitution step required to solve $\mathbf{Ux} = \mathbf{b}'$ requires $N = n^2/2 + n/2$ multiplicative operations. Thus, the total number of multiplicative operations required by LU factorization, after **L** and **U** have been determined, is n^2, which is much less work than required by Gauss elimination, especially for large systems.

The Doolittle LU method, in a format suitable for programming for a computer, is summarized as follows:

1. Perform steps 1, 2, and 3 of the Gauss elimination procedure presented in Section 1.3.3. Store the pivoting information in the order vector **o**. Store the row elimination multipliers, em, in the locations of the eliminated elements. The results of this step are the **L** and **U** matrices.
2. Compute the $\mathbf{b}'$ vector in the order of the elements of the order vector **o** using forward substitution:

$$b_i' = b_i - \sum_{k=1}^{i-1} l_{i,k} b_k' \qquad (i = 2, 3, \ldots, n) \tag{1.139}$$

   where $l_{i,k}$ are the elements of the **L** matrix.
3. Compute the **x** vector using back substitution:

$$x_i = b_i' - \sum_{k=i+1}^{n} u_{i,k} x_k / u_{i,i} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.140}$$

   where $u_{i,k}$ and $u_{i,i}$ are elements of the **U** matrix.

As a final application of LU factorization, it can be used to evaluate the inverse of matrix **A**, that is, $\mathbf{A}^{-1}$. The matrix inverse is calculated in a column by column manner using unit vectors for the right-hand-side vector **b**. Thus, if $\mathbf{b}_1^T = [1 \quad 0 \quad \cdots \quad 0]$, $\mathbf{x}_1$ will be the first column of $\mathbf{A}^{-1}$. The succeeding columns of $\mathbf{A}^{-1}$ are calculated by letting $\mathbf{b}_2^T = [0 \quad 1 \quad \cdots \quad 0]$, $\mathbf{b}_3^T = [0 \quad 0 \quad 1 \quad \cdots \quad 0]$, etc., and $\mathbf{b}_n^T = [0 \quad 0 \quad \cdots \quad 1]$. The number of multiplicative operations for each column is n^2. There are n columns, so the total number of multiplicative operations is n^3. The number of multiplicative operations required to determine **L** and **U** are $(n^3/3 - n/3)$. Thus, the total number of multiplicative operations required is $4n^3/3 - n/3$, which is smaller than the $3n^3/2 - n/2$ operations required by the Gauss-Jordan method.

Example 1.16. Matrix inverse by the Doolittle LU method.

Let's evaluate the inverse of matrix **A** presented in Example 1.7 by the Doolittle LU method:

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \tag{1.141}$$

Evaluate the **L** and **U** matrices by Doolittle LU factorization. Thus,

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \tag{1.142}$$

Let $\mathbf{b}_1^T = [1 \quad 0 \quad 0]$. Then, $\mathbf{L}\mathbf{b}_1' = \mathbf{b}_1$ gives

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b_1' \\ b_2' \\ b_3' \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \mathbf{b}_1' = \begin{bmatrix} 1 \\ 1/4 \\ 3/7 \end{bmatrix} \tag{1.143a}$$

Solve $\mathbf{U}\mathbf{x} = \mathbf{b}_1'$ to determine $\mathbf{x}_1$. Thus,

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/4 \\ 3/7 \end{bmatrix} \rightarrow \mathbf{x}_1 = \begin{bmatrix} 2/125 \\ 1/100 \\ 1/250 \end{bmatrix} \tag{1.143b}$$

where $\mathbf{x}_1$ is the first column of $\mathbf{A}^{-1}$. Letting $\mathbf{b}_2^T = [0 \quad 1 \quad 0]$ gives $\mathbf{x}_2^T = [1/100 \quad 1/30 \quad 1/150]$, and letting $\mathbf{b}_3^T = [0 \quad 0 \quad 1]$ gives $\mathbf{x}_3^T = [1/250 \quad 1/150 \quad 7/750]$. Thus, $\mathbf{A}^{-1}$ is given by

$$\mathbf{A}^{-1} = [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3] = \begin{bmatrix} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{bmatrix}$$

$$= \begin{bmatrix} 0.016000 & 0.010000 & 0.004000 \\ 0.01000 & 0.033333 & 0.006667 \\ 0.004000 & 0.006667 & 0.009333 \end{bmatrix} \tag{1.143c}$$

which is the same result obtained by Gauss-Jordan elimination in Example 1.12.

1.5 TRIDIAGONAL SYSTEMS OF EQUATIONS

When a large system of linear algebraic equations has a special pattern, such as a tridiagonal pattern, it is usually worthwhile to develop special methods for that unique pattern. There are a number of direct elimination methods for solving systems of linear algebraic equations which have special patterns in the coefficient matrix. These methods are generally very efficient in computer time and storage. Such methods should be considered when the coefficient matrix fits the required pattern, and when computer storage and/or execution time are important. One algorithm that deserves special attention is the algorithm for tridiagonal matrices, often referred to as the Thomas (1949) algorithm. Large tridiagonal systems arise naturally in a number of problems, especially in the numerical solution of differential equations by implicit methods. Consequently, the Thomas algorithm has found a large number of applications.

To derive the Thomas algorithm, let's apply the Gauss elimination procedure to a tridiagonal matrix **T**, modifying the procedure to eliminate all unnecessary computations involving zeros. Consider the matrix equation:

$$\boxed{\mathbf{Tx} = \mathbf{b}} \tag{1.144}$$

where **T** is a tridiagonal matrix. Thus,

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \tag{1.145}$$

Since all the elements of column 1 below row 2 are already zero, the only element to be eliminated in row 2 is a_{21}. Thus, replace row 2 by $R_2 - (a_{21}/a_{11})R_1$. Row 2 becomes

$$[0 \quad a_{22} - (a_{21}/a_{11})a_{12} \quad a_{23} \quad 0 \quad 0 \quad \cdots \quad 0 \quad 0 \quad 0] \tag{1.146}$$

Similarly, only a_{32} in column 2 must be eliminated from row 3, only a_{43} in column 3 must be eliminated from row 4, etc. The eliminated element itself does not need to be calculated. In fact, storing the elimination multipliers, em $= (a_{21}/a_{11})$, etc., in place of the eliminated elements allows this procedure to be used as an LU factorization method. Only the diagonal element in each row is affected by the elimination. Elimination in rows 2 to n is accomplished as follows:

$$a_{i,i} = a_{i,i} - (a_{i,i-1}/a_{i-1,i-1})a_{i-1,i} \qquad (i = 2, \ldots, n) \tag{1.147}$$

Thus, the elimination step involves only $2n$ multiplicative operations to place **T** in upper triangular form.

The elements of the **b** vector are also affected by the elimination process. The first element b_1 is unchanged, The second element b_2 becomes

$$b_2 = b_2 - (a_{21}/a_{11})b_1 \tag{1.148}$$

Subsequent elements of the **b** vector are changed in a similar manner. Processing the **b** vector requires only one multiplicative operation, since the elimination multiplier,

em $= (a_{21}/a_{11})$, is already calculated. Thus, the total process of elimination, including the operation on the **b** vector, requires only $3n$ multiplicative operations.

The $n \times n$ tridiagonal matrix **T** can be stored as an $n \times 3$ matrix $\mathbf{A}'$ since there is no need to store the zeros. The first column of matrix $\mathbf{A}'$, elements $a'_{i,1}$, corresponds to the subdiagonal of matrix **T**, elements $a_{i,i-1}$. The second column of matrix $\mathbf{A}'$, elements $a'_{i,2}$, corresponds to the diagonal elements of matrix **T**, elements $a_{i,i}$. The third column of matrix $\mathbf{A}'$, elements $a'_{i,3}$, corresponds to the superdiagonal of matrix **T**, elements $a_{i,i+1}$. The elements $a'_{1,1}$ and $a'_{n,3}$ do not exist. Thus,

$$\mathbf{A}' = \begin{bmatrix} — & a'_{1,2} & a'_{1,3} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} \\ \cdots & \cdots & \cdots \\ a'_{n-1,1} & a'_{n-1,2} & a'_{n-1,3} \\ a'_{n,1} & a'_{n,2} & — \end{bmatrix} \tag{1.149}$$

When the elements of column 1 of matrix $\mathbf{A}'$ are eliminated, that is, the elements $a'_{i,1}$, the elements of column 2 of matrix $\mathbf{A}'$ become

$$a'_{1,2} = a'_{1,2} \tag{1.150a}$$

$$a'_{i,2} = a'_{i,2} - (a'_{i,1}/a'_{i-1,2})a'_{i-1,3} \qquad (i = 2, 3, \ldots, n) \tag{1.150b}$$

The **b** vector is modified as follows:

$$b_1 = b_1 \tag{1.151a}$$

$$b_i = b_i - (a'_{i,1}/a'_{i-1,2})b_{i-1} \qquad (i = 2, 3, \ldots, n) \tag{1.151b}$$

After $a'_{i,2}$ $(i = 2, 3, \ldots, n)$ and **b** are evaluated, the back substitution step is as follows:

$$x_n = b_n/a'_{n,2} \tag{1.152a}$$

$$x_i = (b_i - a'_{i,3}x_{i+1})/a'_{i,2} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.152b}$$

Example 1.17. The Thomas algorithm.

Let's solve the tridiagonal system of equations obtained in Example 8.4, Eq. (8.54). In that example, the finite difference equation

$$T_{i-1} - (2 + \alpha^2\,\Delta x^2)T_i + T_{i+1} = 0 \tag{1.153}$$

is solved for $\alpha = 4.0$ and $\Delta x = 0.125$, for which $(2 + \alpha^2\,\Delta x^2) = 2.25$, for $i = 2, \ldots, 8$, with $T_1 = 0.0$ and $T_9 = 100.0$. Writing Eq. (1.153) in the form of the $n \times 3$ matrix $\mathbf{A}'$ (where the temperatures T_i of Example 8.4 correspond to the elements of the **x** vector) yields

$$\mathbf{A}' = \begin{bmatrix} — & -2.25 & 1.0 \\ 1.0 & -2.25 & 1.0 \\ 1.0 & -2.25 & 1.0 \\ 1.0 & -2.25 & 1.0 \\ 1.0 & -2.25 & 1.0 \\ 1.0 & -2.25 & 1.0 \\ 1.0 & -2.25 & — \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -100.0 \end{bmatrix} \tag{1.154}$$

The major diagonal terms (the center column of the $\mathbf{A}'$ matrix) are transformed according to Eq. (1.150). Thus, $a'_{1,2} = -2.25$ and $a'_{2,2}$ is given by

$$a'_{2,2} = a'_{2,2} - (a'_{2,1}/a'_{1,2})a'_{1,3} = -2.25 - [1.0/(-2.25)](1.0) = -1.805556 \quad (1.155)$$

The remaining elements of column 2 are processed in the same manner. The $\mathbf{A}'$ matrix after elimination is presented in Eq. (1.157), where the elimination multipliers are presented in parentheses in column 1. The $\mathbf{b}$ vector is transformed according to Eq. (1.151). Thus, $b_1 = 0.0$, and

$$b_2 = b_2 - (a'_{2,1}/a'_{1,2})b_1 = 0.0 - [1.0/(-2.25)](0.0) = 0.0 \quad (1.156)$$

The remaining elements of $\mathbf{b}$ are processed in the same manner. The results are presented in Eq. (1.157). For this particular $\mathbf{b}$ vector, where elements b_1 to b_{n-1} are all zero, the $\mathbf{b}$ vector does not change. This is certainly not the case in general. The final result is:

$$\mathbf{A}' = \begin{bmatrix} — & -2.250000 & 1.0 \\ (-0.444444) & -1.805556 & 1.0 \\ (-0.553846) & -1.696154 & 1.0 \\ (-0.589569) & -1.660431 & 1.0 \\ (-0.602253) & -1.647747 & 1.0 \\ (-0.606889) & -1.643111 & 1.0 \\ (-0.608602) & -1.641398 & — \end{bmatrix} \quad \text{and} \quad \mathbf{b}' = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -100.0 \end{bmatrix} \quad (1.157)$$

The solution vector is computed using Eq. (1.152). Thus,

$$x_7 = b_7/a'_{7,2} = (-100)/(-1.641398) = 60.923667 \quad (1.158a)$$

$$x_6 = (b_6 - a'_{6,3}x_7)/a'_{6,2} = [0 - (1.0)(60.923667)]/(-1.643111)$$
$$= 37.078251 \quad (1.158b)$$

Processing the remaining rows yields the solution vector:

$$\mathbf{x} = \begin{bmatrix} 1.966751 \\ 4.425190 \\ 7.989926 \\ 13.552144 \\ 22.502398 \\ 37.078251 \\ 60.923667 \end{bmatrix} \quad (1.158c)$$

Equation (1.158c) is the solution presented in Table 8.9.

Pivoting destroys the tridiagonality of the system of linear algebraic equations, and thus cannot be used with the Thomas algorithm. Most large tridiagonal systems which represent real physical problems are diagonally dominant, so pivoting is not necessary.

The number of multiplicative operations required by the elimination step is $N = 2n - 3$ and the number of multiplicative operations required by the back substitution step is $N = 3n - 2$. Thus, the total number of multiplicative operations is $N = 5n - 4$ for the complete Thomas algorithm. If the $\mathbf{T}$ matrix is constant and multiple $\mathbf{b}$ vectors are to be considered, only the back substitution step is required once the $\mathbf{T}$ matrix has been factored into $\mathbf{L}$ and $\mathbf{U}$ matrices. In that case, $N = 3n - 2$ for subsequent $\mathbf{b}$ vectors. The advantages of the Thomas algorithm are quite apparent when compared with either the Gauss elimination method, for which $N = (n^3/3 - n/3) + n^2$, or the Doolittle LU method, for

which $N = n^2 - n/2$, for each **b** vector after the first one. The Thomas algorithm, in a format suitable for programming for a computer, is summarized as follows:

1. Store the $n \times n$ tridiagonal matrix **T** in the $n \times 3$ matrix **A**′. The right-hand-side vector **b** is an $n \times 1$ column vector.
2. Compute the $a'_{i,2}$ terms from Eq. (1.150). Store the elimination multipliers, $\text{em} = a'_{i,1}/a'_{i-1,2}$, in place of $a'_{i,1}$.
3. Compute the b_i terms from Eq. (1.151).
4. Solve for x_i by back substitution using Eq. (1.152).

An extended form of the Thomas algorithm can be applied to *block tridiagonal matrices*, in which the elements of **T** are partitioned into submatrices having similar patterns. The solution procedure is analogous to that just presented for scalar elements, except that matrix operations are employed on the submatrix elements.

An algorithm similar to the Thomas algorithm can be developed for other special types of systems of linear algebraic equations. For example, a pentadiagonal system of linear algebraic equations is illustrated in Example 8.6.

1.6. PITFALLS OF ELIMINATION METHODS

All nonsingular systems of linear algebraic equations have a solution. In theory, the solution can always be obtained by Gauss elimination. However, there are two major pitfalls in the application of Gauss elimination (or its variations): (a) the presence of round-off errors, and (b) ill-conditioned systems. Those pitfalls are discussed in this section. The effects of round-off can be reduced by a procedure known as iterative improvement, which is presented at the end of this section.

1.6.1. Round-Off Errors

Round-off errors occur when exact infinite precision numbers are approximated by finite precision numbers. In most computers, single precision representation of numbers typically contains about 7 significant digits, double precision representation typically contains about 14 significant digits, and quad precision representation typically contains about 28 significant digits. The effects of round-off errors are illustrated in the following example.

Example 1.18. Effects of round-off errors.

Consider the following system of linear algebraic equations:

$$0.0003x_1 + 3x_2 = 1.0002 \tag{1.159a}$$

$$x_1 + x_2 = 1 \tag{1.159b}$$

Solve Eq. (1.159) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 0.0003 & 3 & 1.0002 \\ 1 & 1 & 1 \end{array}\right] R_2 - R_1/0.0003 \tag{1.160a}$$

$$\left[\begin{array}{cc|c} 0.0003 & 3 & 1.0002 \\ 0 & -9999 & 1 - \dfrac{1.0002}{0.0003} \end{array}\right] \tag{1.160b}$$

Table 1.1. Solution of Eq. (1.162)

Precision	x_2	x_1
3	0.333	3.33
4	0.3332	1.333
5	0.33333	0.70000
6	0.333333	0.670000
7	0.3333333	0.6670000
8	0.33333333	0.66670000

The exact solution of Eq. (1.160) is

$$x_2 = \frac{1 - \dfrac{1.0002}{0.0003}}{-9999} = \frac{\dfrac{0.0003 - 1.0002}{0.0003}}{-9999} = \frac{\dfrac{-0.9999}{0.0003}}{-9999} = \frac{1}{3} \tag{1.161a}$$

$$x_1 = \frac{1.0002 - 3x_2}{0.0003} = \frac{1.0002 - 3(1/3)}{0.0003} = \frac{0.0002}{0.0003} = \frac{2}{3} \tag{1.161b}$$

Let's solve Eq. (1.161) using finite precision arithmetic with two to eight significant figures. Thus,

$$x_2 = \frac{1 - \dfrac{1.0002}{0.0003}}{-9999} \quad \text{and} \quad x_1 = \frac{1.0002 - 3x_2}{0.0003} \tag{1.162}$$

The results are presented in Table 1.1. The algorithm is clearly performing very poorly.

Let's rework the problem by interchanging rows 1 and 2 in Eq. (1.159). Thus,

$$x_1 + x_2 = 1 \tag{1.163a}$$

$$0.0003x_1 + 3x_2 = 1.0002 \tag{1.163b}$$

Solve Eq. (1.163) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 1 & 1 & 1 \\ 0.0003 & 3 & 1.0002 \end{array}\right] R_2 - 0.0003R_1 \tag{1.164a}$$

$$\left[\begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 2.9997 & 0.9999 \end{array}\right] \tag{1.164b}$$

$$x_2 = \frac{0.9999}{2.9997} \quad \text{and} \quad x_1 = 1 - x_2 \tag{1.164c}$$

Let's solve Eq. (1.164c) using finite precision arithmetic. The results are presented in Table 1.2. These results clearly demonstrate the benefits of pivoting.

Table 1.2. Solution of Eq. (1.164c)

Precision	x_2	x_1
3	0.333	0.667
4	0.3333	0.6667
5	0.33333	0.66667

Round-off errors can never be completely eliminated. However, they can be minimized by using high precision arithmetic and pivoting.

1.6.2. System Condition

All well-posed nonsingular numerical problems have an exact solution. In theory, the exact solution can always be obtained using fractions or infinite precision numbers (i.e., an infinite number of significant digits). However, all practical calculations are done with finite precision numbers which necessarily contain round-off errors. The presence of round-off errors alters the solution of the problem.

A *well-conditioned* problem is one in which a small change in any of the elements of the problem causes only a small change in the solution of the problem.

An *ill-conditioned* problem is one in which a small change in any of the elements of the problem causes a large change in the solution of the problem. Since ill-conditioned systems are extremely sensitive to small changes in the elements of the problem, they are also extremely sensitive to round-off errors.

Example 1.19. System condition.

Let's illustrate the behavior of an ill-conditioned system by the following problem:

$$x_1 + x_2 = 2 \tag{1.165a}$$

$$x_1 + 1.0001x_2 = 2.0001 \tag{1.165b}$$

Solve Eq. (1.165) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 1.0001 & 2.0001 \end{array}\right] R_2 - R_1 \tag{1.166a}$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0.0001 & 0.0001 \end{array}\right] \tag{1.166b}$$

Solving Eq. (1.166b) yields $x_2 = 1$ and $x_1 = 1$.

Consider the following slightly modified form of Eq. (1.165) in which a_{22} is changed slightly from 1.0001 to 0.9999:

$$x_1 + x_2 = 2 \tag{1.167a}$$

$$x_1 + 0.9999x_2 = 2.0001 \tag{1.167b}$$

Solving Eq. (1.167) by Gauss elimination gives

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 0.9999 & 2.0001 \end{array}\right] R_2 - R_1 \tag{1.168a}$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & -0.0001 & 0.0001 \end{array}\right] \tag{1.168b}$$

Solving Eq. (1.168b) yields $x_2 = -1$ and $x_1 = 3$, which is greatly different from the solution of Eq. (1.165).

Consider another slightly modified form of Eq. (1.165) in which b_2 is changed slightly from 2.0001 to 2:

$$x_1 + x_2 = 2 \tag{1.169a}$$
$$x_1 + 1.0001x_2 = 2 \tag{1.169b}$$

Solving Eq. (1.169) by Gauss elimination gives

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 1.0001 & 2 \end{array}\right] R_2 - R_1 \tag{1.170a}$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0.0001 & 0 \end{array}\right] \tag{1.170b}$$

Solving Eq. (1.170) yields $x_2 = 0$ and $x_1 = 2$, which is greatly different from the solution of Eq. (1.165).

This problem illustrates that very small changes in any of the elements of **A** or **b** can cause extremely large changes in the solution, **x**. Such a system is ill-conditioned.

With infinite precision arithmetic, ill-conditioning is not a problem. However, with finite precision arithmetic, round-off errors effectively change the elements of **A** and **b** slightly, and if the system is ill-conditioned, large changes (i.e., errors) can occur in the solution. Assuming that scaled pivoting has been performed, the only possible remedy to ill-conditioning is to use higher precision arithmetic.

There are several ways to check a matrix **A** for ill-conditioning. If the magnitude of the determinant of the matrix is small, the matrix may be ill-conditioned. However, this is not a foolproof test. The inverse matrix $\mathbf{A}^{-1}$ can be calculated, and $\mathbf{A}\mathbf{A}^{-1}$ can be computed and compared to **I**. Similary, $(\mathbf{A}^{-1})^{-1}$ can be computed and compared to **A**. A close comparison in either case suggests that matrix **A** is well-conditioned. A poor comparison suggests that the matrix is ill-conditioned. Some of the elements of **A** and/or **b** can be changed slightly, and the solution repeated. If a drastically different solution is obtained, the matrix is probably ill-conditioned. None of these approaches is foolproof, and none give a quantitative measure of ill-conditioning. The surest way to detect ill-conditioning is to evaluate the condition number of the matrix, as discussed in the next subsection.

1.6.3. Norms and the Condition Number

The problems associated with an ill-conditioned system of linear algebraic equations are illustrated in the previous discussion. In the following discussion, ill-conditioning is quantified by the *condition number* of a matrix, which is defined in terms of the *norms* of the matrix and its inverse. Norms and the condition number are discussed in this section.

1.6.3.1. Norms

The measure of the magnitude of **A**, **x**, or **b** is called its *norm* and denoted by $\|\mathbf{A}\|$, $\|\mathbf{x}\|$, and $\|\mathbf{b}\|$, respectively. Norms have the following properties:

$$\|\mathbf{A}\| > 0 \tag{1.171a}$$

$$\|\mathbf{A}\| = 0 \quad \text{only if } \mathbf{A} = \mathbf{0} \tag{1.171b}$$

$$\|k\mathbf{A}\| = |k|\|\mathbf{A}\| \tag{1.171c}$$

$$\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\| \tag{1.171d}$$

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\| \tag{1.171e}$$

The norm of a scalar is its absolute value. Thus, $\|k\| = |k|$. There are several definitions of the norm of a vector. Thus,

$$\|\mathbf{x}\|_1 = \sum |x_i| \qquad \text{Sum of magnitudes} \tag{1.172a}$$

$$\|\mathbf{x}\|_2 = \|\mathbf{x}\|_e = \left(\sum x_i^2\right)^{1/2} \qquad \text{Euclidean norm} \tag{1.172b}$$

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| \qquad \text{Maximum magnitude norm} \tag{1.172c}$$

The *Euclidean* norm is the length of the vector in n-space.

In a similar manner, there are several definitions of the norm of a matrix. Thus,

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}| \qquad \text{Maximum column sum} \tag{1.173a}$$

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}| \qquad \text{Maximum row sum} \tag{1.173b}$$

$$\|\mathbf{A}\|_2 = \min \lambda_i \quad \text{(eigenvalue)} \qquad \text{Spectral norm} \tag{1.173c}$$

$$\|\mathbf{A}\|_e = \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2\right)^{1/2} \qquad \text{Euclidean norm} \tag{1.173d}$$

1.6.3.2. Condition Number

The *condition number* of a system is a measure of the sensitivity of the system to small changes in any of its elements. Consider a system of linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \tag{1.174}$$

For Eq. (1.174),

$$\|\mathbf{b}\| \leq \|\mathbf{A}\|\|\mathbf{x}\| \tag{1.175}$$

Consider a slightly modified form of Eq. (1.174) in which **b** is altered by $\delta\mathbf{b}$, which causes a change in the solution $\delta\mathbf{x}$. Thus,

$$\mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \tag{1.176}$$

Subtracting Eq. (1.174) from Eq. (1.176) gives

$$\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{b} \tag{1.177}$$

Solving Eq. (1.177) for $\delta\mathbf{x}$ gives

$$\delta\mathbf{x} = \mathbf{A}^{-1}\,\delta\mathbf{b} \tag{1.178}$$

For Eq. (1.178),

$$\|\delta \mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \|\delta \mathbf{b}\| \tag{1.179}$$

Multiplying the left-hand and right-hand sides of Eqs. (1.175) and (1.179) gives

$$\|\mathbf{b}\| \|\delta \mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \|\mathbf{A}^{-1}\| \|\delta \mathbf{b}\| \tag{1.180}$$

Dividing Eqs. (1.180) by $\|\mathbf{b}\| \|\mathbf{x}\|$ yields

$$\boxed{\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|} = C(\mathbf{A}) \frac{\|\delta \mathbf{b}\|}{\|\mathbf{b}\|}} \tag{1.181}$$

where $C(\mathbf{A})$ is the *condition number* of matrix $\mathbf{A}$:

$$\boxed{C(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|} \tag{1.182}$$

Equation (1.182) determines the sensitivity of the solution, $\|\delta \mathbf{x}\|/\|\mathbf{x}\|$, to changes in the vector $\mathbf{b}$, $\|\delta \mathbf{b}\|/\|\mathbf{b}\|$. The sensitivity is determined directly by the value of the condition number $C(\mathbf{A})$. Small values of $C(\mathbf{A})$, of the order of unity, show a small sensitivity of the solution to changes in $\mathbf{b}$. Such a problem is well-conditioned. Large values of $C(\mathbf{A})$ show a large sensitivity of the solution to changes in $\mathbf{b}$. Such a problem is ill-conditioned.

It can be shown by a similar analysis that perturbing the matrix $\mathbf{A}$ instead of the vector $\mathbf{b}$ gives

$$\boxed{\frac{\|\delta \mathbf{x}\|}{\|\mathbf{x} + \delta \mathbf{x}\|} \leq C(\mathbf{A}) \frac{\|\delta \mathbf{A}\|}{\|\mathbf{A}\|}} \tag{1.183}$$

The use of the condition number is illustrated in Example 1.20.

Example 1.20. Norms and condition numbers.

Consider the coefficient matrix of Eq. (1.159):

$$\mathbf{A} = \begin{bmatrix} 0.0003 & 3 \\ 1 & 1 \end{bmatrix} \tag{1.184}$$

The Euclidian norm of matrix $\mathbf{A}$ is

$$\|\mathbf{A}\|_e = [(0.0003)^2 + 3^2 + 1^2 + 1^2]^{1/2} = 3.3166 \tag{1.185}$$

The inverse of matrix $\mathbf{A}$ is

$$\mathbf{A}^{-1} = \begin{bmatrix} -\dfrac{1}{(0.0003)9{,}999} & \dfrac{10{,}000}{9{,}999} \\ \dfrac{1}{(0.0003)9{,}999} & -\dfrac{1}{9{,}999} \end{bmatrix} \tag{1.186}$$

The Euclidian norm of $\mathbf{A}^{-1}$ is $\|\mathbf{A}^{-1}\|_e = 1.1057$. Thus, the condition number of matrix $\mathbf{A}$ is

$$C(\mathbf{A}) = \|\mathbf{A}\|_e \|\mathbf{A}^{-1}\|_e = 3.3166(1.1057) = 3.6672 \tag{1.187}$$

This relatively small condition number shows that matrix **A** is well-conditioned. As shown in Section 1.6.1, Eq. (1.159) is sensitive to the precision of the arithmetic (i.e., round-off effects), even though it is well-conditioned. This is a precision problem, not a condition problem.

Consider the coefficient matrix of Eq. (1.165):

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \tag{1.188}$$

The Euclidean norm of matrix **A** is $\|\mathbf{A}\|_e = 2.00005$. The inverse of matrix **A** is

$$\mathbf{A}^{-1} = \begin{bmatrix} 10{,}001 & -10{,}000 \\ -10{,}000 & 10{,}000 \end{bmatrix} \tag{1.189}$$

The Euclidean norm of $\mathbf{A}^{-1}$ is $\|\mathbf{A}^{-1}\|_e = 20{,}000.5$. Thus, the condition number of matrix **A** is

$$C(\mathbf{A}) = \|\mathbf{A}\|_e \|\mathbf{A}^{-1}\|_e = (2.00005)20{,}000.5 = 40{,}002.0 \tag{1.190}$$

This large condition number shows that matrix **A** is ill-conditioned.

1.6.4. Iterative Improvement

In all direct elimination methods, the effects of round-off propagate as the solution progresses through the system of equations. The accumulated effect of round-off is *round-off error* in the computed values. Round-off errors in any calculation can be decreased by using higher precision (i.e., more significant digits) arithmetic. Round-off errors in direct elimination methods of solving systems of linear algebraic equations are minimized by using scaled pivoting. Further reduction in round-off errors can be achieved by a procedure known as *iterative improvement.*

Consider a system of linear algebraic equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1.191}$$

Solving Eq. (1.191) by a direct elimination method yields $\tilde{\mathbf{x}}$, where $\tilde{\mathbf{x}}$ differs from the exact solution $\mathbf{x}$ by the error $\delta\mathbf{x}$, where $\tilde{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$. Substituting $\tilde{\mathbf{x}}$ into Eq. (1.191) gives

$$\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{A}\,\delta\mathbf{x} = \mathbf{b} + \delta\mathbf{b} \tag{1.192}$$

From the first and last terms in Eq. (1.192), $\delta\mathbf{b}$, is given by

$$\delta\mathbf{b} = \mathbf{A}\tilde{\mathbf{x}} - \mathbf{b} \tag{1.193}$$

Subtracting $\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \mathbf{A}\,\delta\mathbf{x}$ into Eq. (1.193) gives a system of linear algebraic equations for $\delta\mathbf{x}$. Thus,

$$\boxed{\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{b}} \tag{1.194}$$

Equation (1.194) can be solved for $\delta\mathbf{x}$, which can be added to $\tilde{\mathbf{x}}$ to give an improved approximation to $\mathbf{x}$. The procedure can be repeated (i.e., iterated) if necessary. A convergence check on the value of $\delta\mathbf{x}$ can be used to determine if the procedure should be repeated. If the procedure is iterated, LU factorization should be used to reduce the

computational effort since matrix **A** is constant. Equation (1.194) should be solved with higher precision than the precision used in the solution of Eq. (1.191).

1.7 ITERATIVE METHODS

For many large systems of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, the coefficient matrix **A** is extremely sparse. That is, most of the elements of **A** are zero. If the matrix is diagonally dominant [see Eq. (1.15)], it is generally more efficient to solve such systems of linear algebraic equations by iterative methods than by direct elimination methods. Three iterative methods are presented in this section: *Jacobi iteration*, *Gauss-Seidel iteration*, and *successive-over-relaxation (SOR)*.

Iterative methods begin by assuming an initial solution vector $\mathbf{x}^{(0)}$. The initial solution vector is used to generate an improved solution vector $\mathbf{x}^{(1)}$ based on some strategy for reducing the difference between $\mathbf{x}^{(0)}$ and the actual solution vector **x**. This procedure is repeated (i.e., iterated) to convergence. The procedure is convergent if each iteration produces approximations to the solution vector that approach the exact solution vector as the number of iterations increases.

Iterative methods do not converge for all sets of equations, nor for all possible arrangements of a particular set of equations. *Diagonal dominance* is a sufficient condition for convergence of Jacobi iteration, Gauss-Seidel iteration, and SOR, for any initial solution vector. Diagonal dominance is defined by Eq. (1.15). Some systems that are not diagonally dominant can be rearranged (i.e., by row interchanges) to make them diagonally dominant. Some systems that are not diagonally dominant may converge for certain initial solution vectors, but convergence is not assured. Iterative methods should not be used for systems of linear algebraic equations that cannot be made diagonally dominant.

When repeated application of an iterative method produces insignificant changes in the solution vector, the procedure should be terminated. In other words, the algorithm is repeated (iterated) until some specified convergence criterion is achieved. Convergence is achieved when some measure of the relative or absolute change in the solution vector is less than a specified convergence criterion. The number of iterations required to achieve convergence depends on:

1. The dominance of the diagonal coefficients. As the diagonal dominance increases, the number of iterations required to satisfy the convergence criterion decreases.
2. The method of iteration used.
3. The initial solution vector.
4. The convergence criterion specified.

1.7.1. The Jacobi Iteration Method

Consider the general system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, written in index notation:

$$\sum_{j=1}^{n} a_{i,j}x_j = b_i \qquad (i = 1, 2, \ldots, n) \tag{1.195}$$

In Jacobi iteration, each equation of the system is solved for the component of the solution vector associated with the diagonal element, that is, x_i. Thus,

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j - \sum_{j=i+1}^{n} a_{i,j}x_j\right) \qquad (i = 1, 2, \ldots, n) \tag{1.196}$$

An initial solution vector $\mathbf{x}^{(0)}$ is chosen. The superscript in parentheses denotes the iteration number, with zero denoting the initial solution vector. The initial solution vector $\mathbf{x}^{(0)}$ is substituted into Eq. (1.196) to yield the first improved solution vector $\mathbf{x}^{(1)}$. Thus,

$$x_i^{(1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(0)} - \sum_{j=i+1}^{n} a_{i,j}x_j^{(0)}\right) \qquad (i = 1, 2, \ldots, n) \tag{1.197}$$

This procedure is repeated (i.e., iterated) until some convergence criterion is satisfied. The Jacobi algorithm for the general iteration step (k) is:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j}x_j^{(k)}\right) \qquad (i = 1, 2, \ldots, n) \tag{1.198}$$

An equivalent, but more convenient, form of Eq. (1.198) can be obtained by adding and subtracting $x_i^{(k)}$ from the right-hand side of Eq. (1.198) to yield

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{i,i}}\left(b_i - \sum_{j=1}^{n} a_{i,j}x_j^{(k)}\right) \qquad (i = 1, 2, \ldots, n) \tag{1.199}$$

Equation (1.199) is generally written in the form

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \qquad (i = 1, 2, \ldots, n) \tag{1.200a}$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{n} a_{i,j}x_j^{(k)} \qquad (i = 1, 2, \ldots, n) \tag{1.200b}$$

where the term $R_i^{(k)}$ is called the *residual* of equation i. The residuals $R_i^{(k)}$ are simply the net values of the equations evaluated for the approximate solution vector $\mathbf{x}^{(k)}$.

The Jacobi method is sometimes called the method of simultaneous iteration because all values of x_i are iterated simultaneously. That is, all values of $x_i^{(k+1)}$ depend only on the values of $x_i^{(k)}$. The order of processing the equations is immaterial.

Example 1.21. The Jacobi iteration method.

To illustrate the Jacobi iteration method, let's solve the following system of linear algebraic equations:

$$\begin{bmatrix} 4 & -1 & 0 & 1 & 0 \\ -1 & 4 & -1 & 0 & 1 \\ 0 & -1 & 4 & -1 & 0 \\ 1 & 0 & -1 & 4 & -1 \\ 0 & 1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \tag{1.201}$$

Table 1.3. Solution by the Jacobi Iteration Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	25.000000	25.000000	25.000000	25.000000	25.000000
2	25.000000	31.250000	37.500000	31.250000	25.000000
3	25.000000	34.375000	40.625000	34.375000	25.000000
4	25.000000	35.156250	42.187500	35.156250	25.000000
5	25.000000	35.546875	42.578125	35.546875	25.000000
...	.....	.....	.....	.....	.....
16	25.000000	35.714284	42.857140	35.714284	25.000000
17	25.000000	35.714285	42.857142	35.714285	25.000000
18	25.000000	35.714285	42.857143	35.714285	25.000000

Equation (1.201), when expanded, becomes

$$4x_1 - x_2 + x_4 = 100 \tag{1.202.1}$$

$$-x_1 + 4x_2 - x_3 + x_5 = 100 \tag{1.202.2}$$

$$-x_2 + 4x_3 - x_4 = 100 \tag{1.202.3}$$

$$x_1 - x_3 + 4x_4 - x_5 = 100 \tag{1.202.4}$$

$$x_2 - x_4 + 4x_5 = 100 \tag{1.202.5}$$

Equation (1.202) can be rearranged to yield expressions for the residuals, R_i. Thus,

$$R_1 = 100 - 4x_1 + x_2 - x_4 \tag{1.203.1}$$

$$R_2 = 100 + x_1 - 4x_2 + x_3 - x_5 \tag{1.203.2}$$

$$R_3 = 100 + x_2 - 4x_3 + x_4 \tag{1.203.3}$$

$$R_4 = 100 - x_1 + x_3 - 4x_4 + x_5 \tag{1.203.4}$$

$$R_5 = 100 - x_2 + x_4 - 4x_5 \tag{1.203.5}$$

To initiate the solution, let $\mathbf{x}^{(0)T} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$. Substituting these values into Eq. (1.203) gives $R_i^{(0)} = 100.0$ $(i = 1, \ldots, 5)$. Substituting these values into Eq. (1.200a) gives $x_1^{(1)} = x_2^{(1)} = x_3^{(1)} = x_4^{(1)} = x_5^{(1)} = 25.0$. The procedure is then repeated with these values to obtain $\mathbf{x}^{(2)}$, etc.

The first and subsequent iterations are summarized in Table 1.3. Due to the symmetry of the coefficient matrix **A** and the symmetry of the **b** vector, $x_1 = x_5$ and $x_2 = x_4$. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 18 iterations.

1.7.2. Accuracy and Convergence of Iterative Methods

All nonsingular systems of linear algebraic equations have an exact solution. In principle, when solved by direct methods, the exact solution can be obtained. However, all real calculations are performed with finite precision numbers, so round-off errors pollute the

solution. Round-off errors can be minimized by pivoting, but even the most careful calculations are subject to the round-off characteristics of the computing device (i.e., hand computation, hand calculator, personal computer, work station, or mainframe computer).

Iterative methods are less susceptable to round-off errors than direct elimination methods for three reasons: (a) The system of equations is diagonally dominant, (b) the system of equations is typically sparse, and (c) each iteration through the system of equations is independent of the round-off errors of the previous iteration.

When solved by iterative methods, the exact solution of a system of linear algebraic equations is approached asymptotically as the number of iterations increases. When the number of iterations increases without bound, the numerical solution yields the exact solution within the round-off limit of the computing device. Such solutions are said to be correct to machine accuracy. In most practical solutions, machine accuracy is not required. Thus, the iterative process should be terminated when some type of accuracy criterion (or criteria) has been satisfied. In iterative methods, the term *accuracy* refers to the number of significant figures obtained in the calculations, and the term *convergence* refers to the point in the iterative process when the desired accuracy is obtained.

1.7.2.1. Accuracy

The *accuracy* of any approximate method is measured in terms of the *error* of the method. There are two ways to specify error: *absolute error* and *relative error.* Absolute error is defined as

$$\text{Absolute error} = \text{approximate value} - \text{exact value} \tag{1.204}$$

and relative error is defined as

$$\text{Relative error} = \frac{\text{absolute error}}{\text{exact value}} \tag{1.205}$$

Relative error can be stated directly or as a percentage.

Consider an iterative calculation for which the desired absolute error is ± 0.001. If the exact solution is 100.000, then the approximate value is 100.000 ± 0.001, which has five significant digits. However, if the exact solution is 0.001000, then the approximate value is 0.001000 ± 0.001, which has no significant digits. This example illustrates the danger of using absolute error as an accuracy criterion. When the magnitude of the exact solution is known, an absolute accuracy criterion can be specified to yield a specified number of significant digits in the approximate solution. Otherwise, a relative accuracy criterion is preferable.

Consider an iterative calculation for which the desired relative error is ± 0.00001. If the exact solution is 100.000, then the absolute error must be $100.000 \times (\pm 0.00001) = \pm 0.001$ to satisfy the relative error criterion. This yields five significant digits in the approximate value. If the exact solution is 0.001000, then the absolute error must be $0.001000 \times (\pm 0.00001) = \pm 0.00000001$ to satisfy the relative error criterion. This yields five significant digits in the approximate solution. A relative error criterion yields the same number of significant figures in the approximate value, regardless of the magnitude of the exact solution.

1.7.2.2. Convergence

Convergence of an iterative procedure is achieved when the desired accuracy criterion (or criteria) is satisfied. Convergence criteria can be specified in terms of absolute error or relative error. Since the exact solution is unknown, the error at any step in the iterative

process is based on the change in the quantity being calculated from one step to the next. Thus, for the iterative solution of a system of linear algebraic equations, the error, $\Delta x_i = x_i^{(k+1)} - x_i^{\text{exact}}$, is approximated by $x_i^{(k+1)} - x_i^{(k)}$. The error can also be specified by the magnitudes of the residuals R_i. When the exact answer (or the exact answer to machine accuracy) is obtained, the residuals are all zero. At each step in the iterative procedure, some of the residuals may be near zero while others are still quite large. Therefore, care is needed to ensure that the desired accuracy of the complete system of equations is achieved.

Let ε be the magnitude of the convergence tolerance. Several convergence criteria are possible. For an absolute error criterion, the following choices are possible:

$$|(\Delta x_i)_{\max}| \le \varepsilon \qquad \sum_{i=1}^{n} |\Delta x_i| \le \varepsilon \qquad \text{or} \qquad \left[\sum_{i=1}^{n} (\Delta x_i)^2\right]^{1/2} \le \varepsilon \tag{1.206}$$

For a relative error criterion, the following choices are possible:

$$\left|\frac{(\Delta x_i)_{\max}}{x_i}\right| \le \varepsilon \qquad \sum_{i=1}^{n} \left|\frac{\Delta x_i}{x_i}\right| \le \varepsilon \qquad \text{or} \qquad \left[\sum_{i=1}^{n} \left(\frac{\Delta x_i}{x_i}\right)^2\right]^{1/2} \le \varepsilon \tag{1.207}$$

The concepts of accuracy and convergence discussed in this section apply to all iterative procedures, not just the iterative solution of a system of linear algebraic equations. They are relevant to the solution of eigenvalue problems (Chapter 2), to the solution of nonlinear equations (Chapter 3), etc.

1.7.3. The Gauss-Seidel Iteration Method

In the Jacobi method, all values of $\mathbf{x}^{(k+1)}$ are based on $\mathbf{x}^{(k)}$. The Gauss-Seidel method is similar to the Jacobi method, except that the most recently computed values of all x_i are used in all computations. In brief, as better values of x_i are obtained, use them immediately. Like the Jacobi method, the Gauss-Seidel method requires diagonal dominance to ensure convergence. The Gauss-Seidel algorithm is obtained from the Jacobi algorithm, Eq. (1.198), by using $x_j^{(k+1)}$ values in the summation from $j = 1$ to $i - 1$ (assuming the sweeps through the equations proceed from $i = 1$ to n). Thus,

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)} \right) \qquad (i = 1, 2, \ldots, n) \tag{1.208}$$

Equation (1.208) can be written in terms of the residuals R_i by adding and subtracting $x_i^{(k)}$ from the right-hand side of the equation and rearranging to yield

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \qquad (i = 1, 2, \ldots, n) \tag{1.209}$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^{n} a_{i,j} x_j^{(k)} \qquad (i = 1, 2, \ldots, n) \tag{1.210}$$

The Gauss-Seidel method is sometimes called the method of successive iteration because the most recent values of all x_i are used in all the calculations. Gauss-Seidel iteration generally converges faster than Jacobi iteration.

Table 1.4. Solution by the Gauss-Seidel Iteration Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	
1	25.000000	31.250000	32.812500	26.953125	23.925781
2	26.074219	33.740234	40.173340	34.506226	25.191498
3	24.808502	34.947586	42.363453	35.686612	25.184757
4	24.815243	35.498485	42.796274	35.791447	25.073240
5	24.926760	35.662448	42.863474	35.752489	25.022510
...	.....	.....	.....	.....	.....
13	25.000002	35.714287	42.857142	35.714285	25.999999
14	25.000001	35.714286	42.857143	35.714285	25.000000
15	25.000000	35.714286	42.857143	35.714286	25.000000

Example 1.22. The Gauss-Seidel iteration method.

Let's rework the problem presented in Example 1.21 using Gauss-Seidel iteration. The residuals are given by Eq. (1.210). Substituting the initial solution vector, $\mathbf{x}^{(0)T} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$, into Eq. (1.210.1) gives $R_1^{(0)} = 100.0$. Substituting that result into Eq. (1.209.1) gives $x_1^{(1)} = 25.0$. Substituting $\mathbf{x}^T = [25.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$ into Eq. (1.210.2) gives

$$R_2^{(1)} = (100.0 + 25.0) = 125.0 \tag{1.211a}$$

Substituting this result into Eq. (1.209.2) yields

$$x_2^{(1)} = 0.0 + \frac{125.0}{4} = 31.25 \tag{1.211b}$$

Continuing in this manner yields $R_3^{(1)} = 131.250$, $x_3^{(1)} = 32.81250$, $R_4^{(1)} = 107.81250$, $x_4^{(1)} = 26.953125$, $R_5^{(1)} = 95.703125$, and $x_5^{(1)} = 23.925781$.

The first and subsequent iterations are summarized in Table 1.4. The intermediate iterates are no longer symmetrical as they were in Example 1.21. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 15 iterations, which is three less than required by the Jacobi method in Example 1.21.

1.7.4. The Successive-Over-Relaxation (SOR) Method

Iterative methods are frequently referred to as relaxation methods, since the iterative procedure can be viewed as relaxing $\mathbf{x}^{(0)}$ to the exact value $\mathbf{x}$. Historically, the method of relaxation, or just the term relaxation, refers to a specific procedure attributed to Southwell (1940). *Southwell's relaxation method* embodies two procedures for accelerating the convergence of the basic iteration scheme. First, the relaxation order is determined by visually searching for the residual of greatest magnitude, $|R_i|_{\max}$, and then relaxing the corresponding equation by calculating a new value of x_i so that $(R_i)_{\max} = 0.0$. This changes the other residuals that depend on x_i. As the other residuals are relaxed, the value of R_i moves away from zero. The procedure is applied repetitively until all the residuals satisfy the convergence criterion (or criteria).

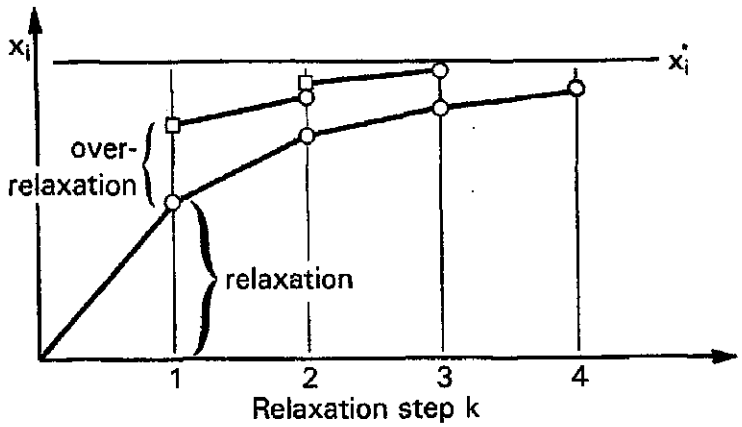

Figure 1.4 Over-relaxation.

Southwell observed that in many cases the changes in x_i from iteration to iteration were always in the same directions. Consequently, over-correcting (i.e., over-relaxing) the values of x_i by the right amount accelerates convergence. This procedure is illustrated in Figure 1.4.

Southwell's method is quite efficient for hand calculation. However, the search for the largest residual is inefficient for computer application, since it can take almost as much computer time to search for the largest residual as it does to make a complete pass through the iteration procedure. On the other hand, the over-relaxation concept is easy to implement on the computer and is very effective in accelerating the convergence rate of the Gauss-Seidel method.

The Gauss-Seidel method can be modified to include over-relaxation simply by multiplying the residual $R_i^{(k)}$ in Eq. (1.209), by the over-relaxation factor, ω. Thus, the *successive-over-relaxation method* is given by

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{R_i^{(k)}}{a_{i,i}} \qquad (i = 1, 2, \ldots, n) \tag{1.212}$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^{n} a_{i,j} x_j^{(k)} \qquad (i = 1, 2, \ldots, n) \tag{1.213}$$

When $\omega = 1.0$, Eq. (1.212) yields the Gauss-Seidel method. When $1.0 < \omega < 2.0$, the system of equations is over-relaxed. Over-relaxation is appropriate for systems of linear algebraic equations. When $\omega < 1.0$, the system of equations is under-relaxed. Under-relaxation is appropriate when the Gauss-Seidel algorithm causes the solution vector to overshoot and move farther away from the exact solution. This behavior is generally associated with the iterative solution of systems of nonlinear algebraic equations. The iterative method diverges if $\omega \geq 2.0$. The relaxation factor does not change the final solution since it multiplies the residual R_i, which is zero when the final solution is reached. The major difficulty with the over-relaxation method is the determination of the best value for the over-relaxation factor, ω. Unfortunately, there is not a good general method for determining the optimum over-relaxation factor, ω_{opt}.

The optimum value of the over-relaxation factor ω_{opt} depends on the size of the system of equations (i.e., the number of equations) and the nature of the equations (i.e., the

strength of the diagonal dominance, the structure of the coefficient matrix, etc.). As a general rule, larger values of ω_{opt} are associated with larger systems of equations. In Section 9.6, Eqs. (9.51) and (9.52), a procedure is described for estimating ω_{opt} for the system of equations obtained when solving the Laplace equation in a rectangular domain with Dirichlet boundary conditions. In general, one must resort to numerical experimentation to determine ω_{opt}. In spite of this inconvenience, it is almost always worthwhile to search for a near optimum value of ω if a system of equations is to be solved many times. In some problems, the computation time can be reduced by factors as large as 10 to 50. For serious calculations with a large number of equations, the potential is too great to ignore.

Example 1.23. The SOR method.

To illustrate the SOR method, let's rework the problem presented in Example 1.22 using $\omega = 1.10$. The residuals are given by Eq. (1.213). Substituting the initial solution vector, $\mathbf{x}^{(0)T} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$, into Eq. (1.213.1) gives $R_1^{(0)} = 100.0$. Substituting that value into Eq. (1.212.1) with $\omega = 1.10$ gives

$$x_1^{(1)} = 0.0 + 1.10\frac{100.0}{4} = 27.500000 \tag{1.214a}$$

Substituting $\mathbf{x}^T = [27.50 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0]$ into Eq. (1.213.2) gives

$$R_2^{(0)} = (100.0 + 27.50) = 127.50 \tag{1.214b}$$

Substituting this result into Eq. (1.212.2) gives

$$x_2^{(1)} = 0.0 + 1.10\frac{127.50}{4} = 35.062500 \tag{1.214c}$$

Continuing in this manner yields the results presented in Table 1.5.

The first and subsequent iterations are summarized in Table 1.5. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 13 iterations, which is 5 less than required by the Jacobi method and 2 less than required by the Gauss-Seidel method. The value of over-relaxation is modest in this example. Its value becomes more significant as the number of equations increases.

Table 1.5. Solution by the SOR Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	27.500000	35.062500	37.142188	30.151602	26.149503
2	26.100497	34.194375	41.480925	35.905571	25.355629
3	24.419371	35.230346	42.914285	35.968342	25.167386
4	24.855114	35.692519	42.915308	35.790750	25.010375
5	24.987475	35.726188	42.875627	35.717992	24.996719
...	.....	.....	.....	.....	.....
11	24.999996	35.714285	42.857145	35.714287	25.000000
12	25.000000	35.714286	42.857143	35.714286	25.000000
13	25.000000	35.714286	42.857143	35.714286	25.000000

Table 1.6. Number of Iterations k as a Function of ω

ω	k	ω	k	ω	k
1.00	15	1.06	13	1.12	13
1.01	14	1.07	13	1.13	13
1.02	14	1.08	13	1.14	13
1.03	14	1.09	13	1.15	14
1.04	14	1.10	13		
1.05	13	1.11	13		

The optimum value of ω can be determined by experimentation. If a problem is to be worked only once, that procedure is not worthwhile. However, if a problem is to be worked many times with the same **A** matrix for many different **b** vectors, then a search for ω_{opt} may be worthwhile. Table 1.6 presents the results of such a search for the problem considered in Example 1.23. For this problem, $1.05 \leq \omega \leq 1.14$ yields the most efficient solution. Much more dramatic results are obtained for large systems of equations.

1.8. PROGRAMS

Four FORTRAN subroutines for solving systems of linear algebraic equations are presented in this section:

1. Simple Gauss elimination
2. Doolittle LU factorization
3. The Thomas algorithm
4. Successive-over-relaxation (SOR)

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

1.8.1. Simple Gauss Elimination

The elimination step of simple Gauss elimination is based on Eq. (1.100). For each column k ($k = 1, 2, \ldots, n-1$),

$$a_{i,j} = a_{i,j} - (a_{i,k}/a_{k,k})a_{k,j} \qquad (i,j = k+1, k+2, \ldots, n) \tag{1.215a}$$

$$b_i = b_i - (a_{i,k}/a_{k,k})b_k \qquad (i = k+1, k+2, \ldots, n) \tag{1.215b}$$

The back substitution step is based on Eq. (1.101):

$$x_n = b_n/a_{n,n} \tag{1.216a}$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j}x_j}{a_{i,i}} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.216b}$$

A FORTRAN subroutine, *subroutine gauss*, for solving these equations, without pivoting, is presented below. Note that the eliminated elements from matrix **A** have been replaced by the elimination multipliers, em, so *subroutine gauss* actually evaluates the **L** and **U** matrices needed for Doolittle LU factorization, which is presented in Section 1.8.2. *Program main* defines the data set and prints it, calls *subroutine gauss* to implement the solution, and prints the solution.

Program 1.1. Simple Gauss elimination program.

```
      program main
c     main program to illustrate linear equation solvers
c     ndim  array dimension, ndim = 6 in this example
c     n     number of equations, n
c     a     coefficient matrix, A(i,j)
c     b     right-hand side vector, b(i)
c     x     solution vector, x(i)
      dimension a(6,6),b(6),x(6)
      data ndim,n / 6, 3 /
      data (a(i,1),i=1,3) /  80.0, -20.0, -20.0 /
      data (a(i,2),i=1,3) / -20.0,  40.0, -20.0 /
      data (a(i,3),i=1,3) / -20.0, -20.0, 130.0 /
      data (b(i),i=1,3) / 20.0, 20.0, 20.0 /
      write (6,1000)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,n),b(i)
      end do
      call gauss (ndim,n,a,b,x)
      write (6,1020)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,n),b(i),x(i)
      end do
      stop
 1000 format (' Simple Gauss elimination'/' '/' A and b'/' ')
 1010 format (i2,7f12.6)
 1020 format (' '/' A, b, and x after elimination'/' ')
      end

      subroutine gauss (ndim,n,a,b,x)
c     simple gauss elimination
      dimension a(ndim,ndim),b(ndim),x(ndim)
c     forward elimination
      do k=1,n-1
         do i=k+1,n
            em=a(i,k)/a(k,k)
            a(i,k)=em
            b(i)=b(i)-em*b(k)
            do j=k+1,n
               a(i,j)=a(i,j)-em*a(k,j)
            end do
         end do
      end do
```

```
c     back substitution
      x(n)=b(n)/a(n,n)
      do i=n-1,1,-1
         x(i)=b(i)
         do j=n,i+1,-1
            x(i)=x(i)-a(i,j)*x(j)
         end do
         x(i)=x(i)/a(i,i)
      end do
      return
      end
```

The data set used to illustrate *subroutine gauss* is taken from Example 1.7. The output generated by the simple Gauss elimination program is presented below.

Output 1.1. Solution by simple Gauss elimination.

```
Simple Gauss elimination

A and b

1   80.000000  -20.000000  -20.000000   20.000000
2  -20.000000   40.000000  -20.000000   20.000000
3  -20.000000  -20.000000  130.000000   20.000000

A, b, and x after elimination

1   80.000000  -20.000000  -20.000000   20.000000    0.600000
2   -0.250000   35.000000  -25.000000   25.000000    1.000000
3   -0.250000   -0.714286  107.142857   42.857143    0.400000
```

1.8.2. Doolittle LU Factorization

Doolittle LU factorization is based on the LU factorization implicit in Gauss elimination. *Subroutine gauss* presented in Section 1.8.1 is modified to evaluate the **L** and **U** matrices simply by removing the line evaluating $b(i)$ from the first group of statements and entirely deleting the second group of statements, which evaluates the back substitution step. The modified subroutine is named *subroutine lufactor*.

A second subroutine, *subroutine solve*, based on steps 2 and 3 in the description of the Doolittle LU factorization method in Section 1.4, is required to process the **b** vector to the **b**′ vector and to process the **b**′ vector to the **x** vector. These steps are given by Eqs. (1.139) and (1.140):

$$b'_i = b_i - \sum_{k=1}^{i-1} l_{i,k} b'_k \qquad (i = 2, 3, \ldots, n) \tag{1.217a}$$

$$x_i = b'_i - \sum_{k=i+1}^{n} u_{i,k} x_k / u_{i,i} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.217b}$$

FORTRAN subroutines for implementing Doolittle LU factorization are presented below. *Program main* defines the data set and prints it, calls *subroutine lufactor* to evaluate the **L** and **U** matrices, calls *subroutine solve* to implement the solution for a specified **b**

vector, and prints the solution. *Program main* below shows only the statements which are different from the statements in *program main* in Section 1.8.1.

Program 1.2. Doolittle LU factorization program.

```
      program main
c     main program to illustrate linear equation solvers
c     bp    b' vector, bp(i)
      dimension a(6,6),b(6),bp(6),x(6)
      call lufactor (ndim,n,a)
      write (6,1020)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,n)
      end do
      call solve (ndim,n,a,b,bp,x)
      write (6,1030)
      do i=1,n
         write (6,1010) i,b(i),bp(i),x(i)
      end do
      stop
 1000 format (' Doolittle LU factorization'/' '/' A and b'/' ')
 1010 format (i2,7f12.6)
 1020 format (' '/' L and U stored in A'/' ')
 1030 format (' '/' b, bprime, and x vectors'/' ')
      end

      subroutine lufactor (ndim,n,a)
c     Doolittle LU factorization, stores L and U in A
      dimension a(ndim,ndim)
      do k=1,n-1
         do i=k+1,n
            em=a(i,k)/a(k,k)
            a(i,k)=em
            do j=k+1,n
               a(i,j)=a(i,j)-em*a(k,j)
            end do
         end do
      end do
      return
      end

      subroutine solve (ndim,n,a,b,bp,x)
c     processes b to b' and b' to x
      dimension a(ndim,ndim),b(ndim),bp(ndim),x(ndim)
c     forward elimination step to calculate b'
      bp(1)=b(1)
      do i=2,n
         bp(i)=b(i)
         do j=1,i-1
            bp(i)=bp(i)-a(i,j)*bp(j)
         end do
      end do
```

```
c     back substitution step to calculate x
      x(n)=bp(n)/a(n,n)
      do i=n-1,1,-1
         x(i)=bp(i)
         do j=n,i+1,-1
             x(i)=x(i)-a(i,j)*x(j)
         end do
         x(i)=x(i)/a(i,i)
      end do
      return
      end
```

The data set used to illustrate *subroutines lufactor* and *solve* is taken from Example 1.15. The output generated by the Doolittle LU factorization program is presented below.

Output 1.2. Solution by Doolittle LU factorization.

```
Doolittle LU factorization

A and b

1   80.000000  -20.000000  -20.000000
2  -20.000000   40.000000  -20.000000
3  -20.000000  -20.000000  130.000000

L and U matrices stored in A matrix

1   80.000000  -20.000000  -20.000000
2   -0.250000   35.000000  -25.000000
3   -0.250000   -0.714286  107.142857

b, bprime, and x vectors

1   20.000000   20.000000    0.600000
2   20.000000   25.000000    1.000000
3   20.000000   42.857143    0.400000
```

1.8.3. The Thomas Algorithm

The elimination step of the Thomas algorithm is based on Eqs. (1.150) and (1.151):

$$a'_{1,2} = a'_{1,2} \tag{1.218a}$$

$$a'_{i,2} = a'_{i,2} - (a'_{i,1}/a'_{i-1,2})a'_{i-1,3} \qquad (i = 2, 3, \ldots, n) \tag{1.218b}$$

$$b_1 = b_1 \tag{1.218c}$$

$$b_i = b_i - (a'_{i,1}/a'_{i-1,2})b_{i-1} \qquad (i = 2, 3, \ldots, n) \tag{1.218d}$$

The back substitution step is based on Eq. (1.152):

$$x_n = b_n/a'_{n,2} \tag{1.219a}$$

$$x_i = (b_i - a'_{i,3}x_{i+1})/a'_{i,2} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.219b}$$

A FORTRAN subroutine, *subroutine thomas*, for solving these equations is presented below. Note that the eliminated elements from matrix A have been replaced by the elimination multipliers, em, so *subroutine thomas* actually evaluates the L and U matrices needed for Doolittle LU factorization. *Program main* defines the data set and prints it, calls *subroutine thomas* to implement the solution, and prints the solution.

Program 1.3. The Thomas algorithm program.

```
      program main
c     main program to illustrate linear equation solvers
c     ndim  array dimension, ndim = 9 in this example
c     n     number of equations, n
c     a     coefficient matrix, A(i,3)
c     b     right-hand side vector, b(i)
c     x     solution vector, x(i)
      dimension a(9,3),b(9),x(9)
      data ndim,n / 9, 7 /
      data (a(1,j),j=1,3) /  0.0, -2.25, 1.0 /
      data (a(2,j),j=1,3) /  1.0, -2.25, 1.0 /
      data (a(3,j),j=1,3) /  1.0, -2.25, 1.0 /
      data (a(4,j),j=1,3) /  1.0, -2.25, 1.0 /
      data (a(5,j),j=1,3) /  1.0, -2.25, 1.0 /
      data (a(6,j),j=1,3) /  1.0, -2.25, 1.0 /
      data (a(7,j),j=1,3) /  1.0, -2.25, 0.0 /
      data (b(i),i=1,7) / 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -100.0 /
      write (6,1000)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,3),b(i)
      end do
      call thomas (ndim,n,a,b,x)
      write (6,1020)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,3),b(i),x(i)
      end do
      stop
 1000 format (' The Thomas algorithm'/' '/' A and b'/' ')
 1010 format (i2,6f12.6)
 1020 format (' '/' A, b, and x after elimination'/' ')
      end

      subroutine thomas (ndim,n,a,b,x)
c     the Thomas algorithm for a tridiagonal system
      dimension a(ndim,3),b(ndim),x(ndim)
c     forward elimination
      do i=2,n
         em=a(i,1)/a(i-1,2)
         a(i,1)=em
         a(i,2)=a(i,2)-em*a(i-1,3)
         b(i)=b(i)-a(i,1)*b(i-1)
      end do
```

```
c     back substitution
      x(n)=b(n)/a(n,2)
      do i=n-1,1,-1
         x(i)=(b(i)-a(i,3)*x(i+1))/a(i,2)
      end do
      return
      end
```

The data set used to illustrate *subroutine thomas* is taken from Example 1.17. The output generated by the Thomas algorithm program is presented below.

Output 1.3. Solution by the Thomas algorithm.

```
The Thomas algorithm

A and b

1     0.000000   -2.250000    1.000000     0.000000
2     1.000000   -2.250000    1.000000     0.000000
3     1.000000   -2.250000    1.000000     0.000000
4     1.000000   -2.250000    1.000000     0.000000
5     1.000000   -2.250000    1.000000     0.000000
6     1.000000   -2.250000    1.000000     0.000000
7     1.000000   -2.250000    0.000000  -100.000000

A, b, and x after elimination

1     0.000000   -2.250000    1.000000     0.000000     1.966751
2    -0.444444   -1.805556    1.000000     0.000000     4.425190
3    -0.553846   -1.696154    1.000000     0.000000     7.989926
4    -0.589569   -1.660431    1.000000     0.000000    13.552144
5    -0.602253   -1.647747    1.000000     0.000000    22.502398
6    -0.606889   -1.643111    1.000000     0.000000    37.078251
7    -0.608602   -1.641398    0.000000  -100.000000    60.923667
```

1.8.4. Successive-Over-Relaxation (SOR)

Successive-over-relaxation (SOR) is based on Eqs. (1.212) and (1.213):

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{R_i^{(k)}}{a_{i,i}} \qquad (i = 1, 2, \ldots, n) \tag{1.220a}$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^{n} a_{i,j} x_j^{(k)} \qquad (i = 1, 2, \ldots, n) \tag{1.220b}$$

A FORTRAN subroutine, *subroutine sor*, for solving these equations is presented below. *Program main* defines the data set and prints it, calls *subroutine sor* to implement the solution, and prints the solution. Input variable *iw* is a flag for output of intermediate results. When $iw = 0$, no intermediate results are output. When $iw = 1$, intermediate results are output.

Program 1.4. Successive-over-relaxation (SOR) program.

```
      program main
c     main program to illustrate linear equation solvers
c     ndim  array dimension, ndim = 6 in this example
c     n     number of equations, n
c     a     coefficient matrix, A(i,j)
c     b     right-hand side vector, b(i)
c     x     solution vector, x(i)
c     iter  number of iterations allowed
c     tol   convergence tolerance
c     omega over-relaxation factor
c     iw    flag for intermediate output: 0 no, 1 yes
      dimension a(6,6),b(6),x(6)
      data ndim,n,iter,tol,omega,iw / 6,5,25,0.000001,1.0,1 /
      data (a(i,1),i=1,5) /  4.0, -1.0,  0.0,  1.0,  0.0 /
      data (a(i,2),i=1,5) / -1.0,  4.0, -1.0,  0.0,  1.0 /
      data (a(i,3),i=1,5) /  0.0, -1.0,  4.0, -1.0,  0.0 /
      data (a(i,4),i=1,5) /  1.0,  0.0, -1.0,  4.0, -1.0 /
      data (a(i,5),i=1,5) /  0.0,  1.0,  0.0, -1.0,  4.0 /
      data (b(i),i=1,5) / 100.0, 100.0, 100.0, 100.0, 100.0 /
      data (x(i),i=1,5) /  0.0,  0.0,  0.0,  0.0,  0.0 /
      write (6,1000)
      do i=1,n
         write (6,1010) i,(a(i,j),j=1,n),b(i)
      end do
      write (6,1020)
      it=0
      write (6,1010) it,(x(i),i=1,n)
      call sor (ndim,n,a,b,x,iter,tol,omega,iw,it)
      if (iw.eq.0) write (6,1010) it,(x(i),i=1,n)
      stop
 1000 format (' SOR iteration'/' '/' A and b'/' ')
 1010 format (i2,7f12.6)
 1020 format (' '/' i                x(1) to x(n)'/' ')
      end

      subroutine sor (ndim,n,a,b,x,iter,tol,omega,iw,it)
c     sor iteration
      dimension a(ndim,ndim),b(ndim),x(ndim)
      do it=1,iter
         dxmax=0.0
         do i=1,n
            residual=b(i)
            do j=1,n
               residual=residual-a(i,j)*x(j)
            end do
            if (abs(residual).gt.dxmax) dxmax=abs(residual)
         x(i)=x(i)+omega*residual/a(i,i)
         end do
```

```
        if (iw.eq.1) write (6,1000) it,(x(i),i=1,n)
        if (dxmax.lt.tol) return
      end do
      write (6,1010)
      return
1000  format (i2,7f12.6)
1010  format (' '/' Solution failed to converge'/' ')
      end
```

The data set used to illustrate *subroutine sor* is taken from Example 1.23. The output generated by the SOR program is presented below.

Output 1.4. Solution by successive-over-relaxation (SOR).

```
 SOR iteration

 A and b

 1    4.000000  -1.000000   0.000000   1.000000   0.000000 100.000000
 2   -1.000000   4.000000  -1.000000   0.000000   1.000000 100.000000
 3    0.000000  -1.000000   4.000000  -1.000000   0.000000 100.000000
 4    1.000000   0.000000  -1.000000   4.000000  -1.000000 100.000000
 5    0.000000   1.000000   0.000000  -1.000000   4.000000 100.000000

 i                 x(1) to x(n)

 0    0.000000   0.000000   0.000000   0.000000   0.000000
 1   25.000000  31.250000  32.812500  26.953125  23.925781
 2   26.074219  33.740234  40.173340  34.506226  25.191498
 3   24.808502  34.947586  42.363453  35.686612  25.184757
 4   24.815243  35.498485  42.796274  35.791447  25.073240
     ..........................................................
15   25.000000  35.714286  42.857143  35.714286  25.000000
16   25.000000  35.714286  42.857143  35.714286  25.000000
```

1.8.5. Packages for Systems of Linear Algebraic Equations

Numerous libraries and software packages are available for solving systems of linear algebraic equations. Many work stations and mainframe computers have such libraries attached to their operating systems. If not, libraries such as IMSL (International Mathematics and Statistics Library) or LINPACK (Argonne National Laboratory) can be added to the operating systems.

Most commercial software packages contain solvers for systems of linear algebraic equations. Some of the more prominent packages are Matlab and Mathcad. The spreadsheet Excel can also be used to solve systems of equations. More sophisticated packages, such as Mathematica, Macsyma, and Maple, also contain linear equation solvers. Finally, the book *Numerical Recipes* (Press et al., 1989) contains several subroutines for solving systems of linear algebraic equations.

8. Calculate the following determinants by the diagonal method, if defined:
   (a) det(**A**) (b) det(**B**) (c) det(**C**) (d) det(**D**) (e) det(**AB**)
   (f) det(**AD**) (g) det(**BA**) (h) det(**DA**) (i) det(**CD**) (j) det($\mathbf{C}^T\mathbf{A}$)
9. Work Problem 8 using the cofactor method.
10. Show that det(**A**) det(**B**) = det(**AB**).
11. Show that det(**A**) det(**D**) = det(**AD**).
12. Show that for the general 2×2 matrices **A** and **B**, det(**A**) det(**B**) = det(**AB**).

Section 1.3. Direct Elimination Methods

Consider the following eight systems of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$:

$$\begin{aligned} -2x_1 + 3x_2 + x_3 &= 9 \\ 3x_1 + 4x_2 - 5x_3 &= 0 \\ x_1 - 2x_2 + x_3 &= -4 \end{aligned} \quad \text{(A)} \qquad \begin{bmatrix} 1 & 1 & 3 \\ 5 & 3 & 1 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ -1 \end{bmatrix} \quad \text{(B)}$$

$$\begin{aligned} x_1 + 3x_2 + 2x_3 - x_4 &= 9 \\ 4x_1 + 2x_2 + 5x_3 + x_4 &= 27 \\ 3x_1 - 3x_2 + 2x_3 + 4x_4 &= 19 \\ -x_1 + 2x_2 - 3x_3 + 5x_4 &= 14 \end{aligned} \quad \text{(C)} \qquad \begin{bmatrix} 3 & 1 & -1 & 3 \\ 2 & 1 & -2 & 0 \\ 0 & 3 & 2 & -2 \\ 1 & 1 & 1 & 5 \end{bmatrix} [x_i] = \begin{bmatrix} 4 \\ -1 \\ 4 \\ -2 \end{bmatrix} \quad \text{(D)}$$

$$\begin{bmatrix} 1 & -2 & 1 \\ 2 & 1 & 2 \\ -1 & 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix} \quad \text{(E)} \qquad \begin{bmatrix} 2 & 3 & 5 \\ 3 & 1 & -2 \\ 1 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ -3 \end{bmatrix} \quad \text{(F)}$$

$$\begin{bmatrix} 2 & -2 & 2 & 1 \\ 2 & -4 & 1 & 3 \\ -1 & 3 & -4 & 2 \\ 2 & 4 & 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \\ -14 \\ 1 \end{bmatrix} \quad \text{(G)} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 3 & 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -4 \\ 1 \end{bmatrix} \quad \text{(H)}$$

Cramer's Rule

13. Solve Eq. (A) by Cramer's rule.
14. Solve Eq. (B) by Cramer's rule.
15. Solve Eq. (C) by Cramer's rule.
16. Solve Eq. (D) by Cramer's rule.
17. Solve Eq. (E) by Cramer's rule.
18. Solve Eq. (F) by Cramer's rule.
19. Solve Eq. (G) by Cramer's rule.
20. Solve Eq. (H) by Cramer's rule.

Gauss Elimination

21. Solve Eq. (A) by Gauss elimination without pivoting.
22. Solve Eq. (B) by Gauss elimination without pivoting.
23. Solve Eq. (C) by Gauss elimination without pivoting.
24. Solve Eq. (D) by Gauss elimination without pivoting.
25. Solve Eq. (E) by Gauss elimination without pivoting.
26. Solve Eq. (F) by Gauss elimination without pivoting.
27. Solve Eq. (G) by Gauss elimination without pivoting.
28. Solve Eq. (H) by Gauss elimination without pivoting.

Gauss-Jordan Elimination

29. Solve Eq. (A) by Gauss-Jordan elimination.
30. Solve Eq. (B) by Gauss-Jordan elimination.
31. Solve Eq. (C) by Gauss-Jordan elimination.
32. Solve Eq. (D) by Gauss-Jordan elimination.
33. Solve Eq. (E) by Gauss-Jordan elimination.
34. Solve Eq. (F) by Gauss-Jordan elimination.
35. Solve Eq. (G) by Gauss-Jordan elimination.
36. Solve Eq. (H) by Gauss-Jordan elimination.

The Matrix Inverse Method

37. Solve Eq. (A) using the matrix inverse method.
38. Solve Eq. (B) using the matrix inverse method.
39. Solve Eq. (C) using the matrix inverse method.
40. Solve Eq. (D) using the matrix inverse method.
41. Solve Eq. (E) using the matrix inverse method.
42. Solve Eq. (F) using the matrix inverse method.
43. Solve Eq. (G) using the matrix inverse method.
44. Solve Eq. (H) using the matrix inverse method.

Section 1.4. LU Factorization

45. Solve Eq. (A) by the Doolittle LU factorization method.
46. Solve Eq. (B) by the Doolittle LU factorization method.
47. Solve Eq. (C) by the Doolittle LU factorization method.
48. Solve Eq. (D) by the Doolittle LU factorization method.
49. Solve Eq. (E) by the Doolittle LU factorization method.
50. Solve Eq. (F) by the Doolittle LU factorization method.
51. Solve Eq. (G) by the Doolittle LU factorization method.
52. Solve Eq. (H) by the Doolittle LU factorization method.

Section 1.5. Tridiagonal Systems of Equations

Consider the following tridiagonal systems of linear algebraic equations:

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 11 \end{bmatrix} \text{ (I)} \qquad \begin{bmatrix} 3 & 2 & 0 & 0 \\ 2 & 3 & 2 & 0 \\ 0 & 2 & 3 & 2 \\ 0 & 0 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 17 \\ 14 \\ 7 \end{bmatrix} \text{ (J)}$$

$$\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ -7 \\ -1 \end{bmatrix} \text{ (K)} \qquad \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 0 \\ 8 \end{bmatrix} \text{ (L)}$$

$$\begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix} [x_i] = \begin{bmatrix} 150 \\ 200 \\ 150 \\ 100 \end{bmatrix} \text{ (M)} \qquad \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 0 \\ 8 \end{bmatrix} \text{ (N)}$$

53. Solve Eq. (I) by the Thomas algorithm.
54. Solve Eq. (J) by the Thomas algorithm.
55. Solve Eq. (K) by the Thomas algorithm.
56. Solve Eq. (L) by the Thomas algorithm.
57. Solve Eq. (M) by the Thomas algorithm.
58. Solve Eq. (N) by the Thomas algorithm.

Section 1.7. Iterative Methods

Solve the following problems by iterative methods. Let $\mathbf{x}^{(0)T} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0]$. For hand calculations, make at least five iterations. For computer solutions, iterate until six digits after the decimal place converges.

Jacobi Iteration

59. Solve Eq. (I) by Jacobi iteration.
60. Solve Eq. (K) by Jacobi iteration.
61. Solve Eq. (L) by Jacobi iteration.
62. Solve Eq. (M) by Jacobi iteration.
63. Solve Eq. (N) by Jacobi iteration.

Gauss-Seidel Iteration

64. Solve Eq. (I) by Gauss-Seidel iteration.
65. Solve Eq. (K) by Gauss-Seidel iteration.
66. Solve Eq. (L) by Gauss-Seidel iteration.
67. Solve Eq. (M) by Gauss-Seidel iteration.
68. Solve Eq. (N) by Gauss-Seidel iteration.

Successive Over-Relaxation

69. Solve Eq. (I) by the SOR method with $\omega = 1.27$.
70. Solve Eq. (K) by the SOR method with $\omega = 1.27$.
71. Solve Eq. (L) by the SOR method with $\omega = 1.27$.
72. Solve Eq. (M) by the SOR method with $\omega = 1.05$.
73. Solve Eq. (N) by the SOR method with $\omega = 1.25$.
74. Solve Eq. (I) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
75. Solve Eq. (K) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
76. Solve Eq. (L) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
77. Solve Eq. (M) by the SOR method for $1.00 \leq \omega = 1.10$ with $\Delta\omega = 0.01$.
78. Solve Eq. (N) by the SOR method for $1.25 \leq \omega = 1.35$ with $\Delta\omega = 0.01$.

Section 1.8. Programs

79. Implement the simple Gauss elimination program presented in Section 1.8.1. Check out the program using the given data.
80. Solve any of Eqs. (A) to (H) using the Gauss elimination program.
81. Implement the Doolittle LU factorization program presented in Section 1.8.2. Check out the program using the given data.
82. Solve any of Eqs. (A) to (H) using the Doolittle LU factorization program.
83. Implement the Thomas algorithm program presented in Section 1.8.3. Check out the program using the given data.
84. Solve any of Eqs. (I) to (N) using the Thomas algorithm program.
85. Implement the SOR program presented in Section 1.8.4. Check out the program using the given data.
86. Solve any of Eqs. (I) and (K) to (N) using the SOR program.