2

Solving Sets of Equations

Solving sets of linear equations is the most frequently used numerical procedure when real-world situations are modeled. Linear equations are the basis for mathematical models of economics, weather prediction, heat and mass transfer, statistical analysis, and a myriad of other applications. The methods for solving ordinary and partial-differential equations depend on them. In this book, almost every chapter uses the algorithms that we discuss here.

It is almost impossible to discuss systems of more than two or three equations without using matrices and vectors, so we cover some of the concepts of these at the start. Other aspects of the characteristics of a matrix are described in later chapters.

Contents of This Chapter

It shows when a matrix cannot be inverted and tells of situations where no unique solution exists to a system of equations.

2.4 Ill-Conditioned Systems

Explores systems for which getting the solution with accuracy is very difficult. A number, the condition number, is a measure of such difficulty; a property of a matrix, called its norm, is used to compute its condition number. A way to improve an inaccurate solution is described.

2.5 Iterative Methods

This section describes how a linear system can be solved in an entirely different way, by beginning with an initial estimate of the solution and performing computations that eventually arrive at the correct solution. It tells how the convergence can be accelerated. An iterative method is particularly important in solving systems that have few nonzero coefficients.

2.6 Parallel Processing

Tells how parallel computing can be applied to the solution of linear systems. An algorithm is developed that allows a significant reduction in processing time.

2.1 Matrices and Vectors

When a system of equations has more than two or three equations, it is difficult to discuss them without using *matrices* and *vectors*. While you may already know something about them, it is important that we review this topic in some detail.

A *matrix* is a rectangular array of numbers in which not only the value of the number is important but also its position in the array. The size of the matrix is described by the number of its rows and columns. A matrix of n rows and m columns is said to be $n \times m$. The elements of the matrix are generally enclosed in brackets, and double-subscripting is the common way of indexing the elements. The first subscript also denotes the row, and the second denotes the column in which the element occurs. Capital letters are used to refer to matrices. For example,

$$A = \begin{bmatrix} a_{11} & a_{12} \ldots a_{1m} \\ a_{21} & a_{22} \ldots a_{2m} \\ \vdots & \\ a_{n1} & a_{n2} \ldots a_{nm} \end{bmatrix} = [a_{ij}], \quad i = 1, 2, \ldots, n, \quad j = 1, 2, \ldots, m.$$

Enclosing the general element a_{ij} in brackets is another way of representing matrix A, as just shown. Sometimes we will enclose the name of the matrix in brackets, $[A]$, to emphasize that A is a matrix.

Two matrices of the same size may be added or subtracted. The sum of

$$A = [a_{ij}] \qquad \text{and} \qquad B = [b_{ij}]$$

is the matrix whose elements are the sum of the corresponding elements of A and B:

$$\boxed{C = A + B = [a_{ij} + b_{ij}] = [c_{ij}].}$$

Similarly, we get the *difference* of two equal-sized matrices by subtracting corresponding elements. If two matrices are not equal in size, they cannot be added or subtracted. Two matrices are equal if and only if each element of one is the same as the corresponding element of the other. Obviously, equal matrices must be of the same size. Some examples will help make this clear.

If

$$A = \begin{bmatrix} 4 & 7 & -5 \\ -4 & 2 & 12 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 1 & 5 & 4 \\ 2 & -6 & 3 \end{bmatrix},$$

we say that A is 2×3 because it has two rows and three columns. B is also 2×3. Their sum C is also 2×3:

$$C = A + B = \begin{bmatrix} 5 & 12 & -1 \\ -2 & -4 & 15 \end{bmatrix}.$$

The difference D of A and B is

$$D = A - B = \begin{bmatrix} 3 & 2 & -9 \\ -6 & 8 & 9 \end{bmatrix}.$$

Multiplication of two matrices is defined as follows, when A is $n \times m$ and B is $m \times r$:

$$[a_{ij}] * [b_{ij}] = [c_{ij}]$$
$$= \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1m}b_{m1}) \ldots (a_{11}b_{1r} + \cdots + a_{1m}b_{mr}) \\ (a_{21}b_{11} + a_{22}b_{21} + \cdots + a_{2m}b_{m1}) \ldots (a_{21}b_{1r} + \cdots + a_{2m}b_{mr}) \\ \vdots \\ (a_{n1}b_{11} + a_{n2}b_{21} + \cdots + a_{nm}b_{m1}) \ldots (a_{n1}b_{1r} + \cdots + a_{nm}b_{mr}) \end{bmatrix},$$

$$\boxed{c_{ij} = \sum_{k=1}^{m} a_{ik}b_{kj}, \qquad i = 1, 2, \ldots, n, \qquad j = 1, 2, \ldots, r.}$$

It is simplest to select the proper elements if we count across the rows of A with the left hand while counting down the columns of B with the right. Unless the number of columns of A equals the number of rows of B (so the counting comes out even), the matrices cannot be multiplied. Hence, if A is $n \times m$, B must have m rows or else they are said to be "nonconformable for multiplication" and their product is undefined. In general $AB \neq BA$, so the order of factors must be preserved in matrix multiplication.

If a matrix is multiplied by a scalar (a pure number), the product is a matrix, each element of which is the scalar times the original element. We can write

$$\boxed{\text{If } kA = C, \qquad c_{ij} = ka_{ij}.}$$

A matrix with only one column, $n \times 1$ in size, is termed a *column vector,* and one of only one row, $1 \times m$ in size, is called a *row vector.* When the unqualified term *vector* is used, it nearly always means a *column* vector. Frequently the elements of vectors are only singly subscripted.

Some examples of matrix multiplication follow.

$$\text{Suppose } A = \begin{bmatrix} 3 & 7 & 1 \\ -2 & 1 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & -2 \\ 0 & 3 \\ 1 & -1 \end{bmatrix}, \quad x = \begin{bmatrix} -3 \\ 1 \\ 4 \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}.$$

$$A * B = \begin{bmatrix} 16 & 14 \\ -13 & 10 \end{bmatrix}; \quad B * A = \begin{bmatrix} 19 & 33 & 11 \\ -6 & 3 & -9 \\ 5 & 6 & 4 \end{bmatrix};$$

$$A * x = \begin{bmatrix} 2 \\ -5 \end{bmatrix}; \quad A * y = \begin{bmatrix} 3y_1 + 7y_2 + y_3 \\ -2y_1 + y_2 - 3y_3 \end{bmatrix}.$$

Because A is 2×3 and B is 3×2, they are conformable for multiplication and their product is 2×2. When we form the product of $B * A$, it is 3×3. Observe that not only is $AB \neq BA$; AB and BA are not even the same size. The product of A and the vector x (a 3×1 matrix) is another vector, one with two components. Similarly, Ay has two components. We cannot multiply B times x or B times y; they are nonconformable.

The product of the scalar number 2 and A is

$$2A = \begin{bmatrix} 6 & 14 & 2 \\ -4 & 2 & -6 \end{bmatrix}.$$

Because a vector is just a special case of a matrix, a column vector can be multiplied by a matrix, as long as they are conformable in that the number of columns of the matrix equals the number of elements (rows) in the vector. The product in this case will be another column vector. The size of a product of two matrices, the first $m \times n$ and the second $n \times r$, is $m \times r$. An $m \times n$ matrix times an $n \times 1$ vector gives an $m \times 1$ product.

The general relation for $Ax = b$ is

$$\boxed{b_i = \sum_{k=1}^{\text{No. of cols.}} a_{ik}x_k, \qquad i = 1, 2, \ldots, \text{No. of rows.}}$$

This definition of matrix multiplication permits us to write the set of linear equations

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

much more simply in matrix notation, as $\boxed{Ax = b,}$ where

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

For example,

$$\begin{bmatrix} 3 & 2 & 4 \\ 1 & -2 & 0 \\ -1 & 3 & 2 \end{bmatrix} * x = \begin{bmatrix} 14 \\ -7 \\ 2 \end{bmatrix}$$

is the same as the set of equations

$$\begin{aligned}
3x_1 + 2x_2 + 4x_3 &= 14,\\
x_1 - 2x_2 \qquad &= -7,\\
-x_1 + 3x_2 + 2x_3 &= 2.
\end{aligned}$$

Two vectors, each with the same number of components, may be added or subtracted. Two vectors are equal if each component of one equals the corresponding component of the other.

A very important special case is the multiplication of two vectors. The first must be a row vector if the second is a column vector, and each must have the same number of components. For example,

$$[1 \quad 3 \quad -2] * \begin{bmatrix} 4 \\ -1 \\ 3 \end{bmatrix} = [-5]$$

gives a "matrix" of one row and one column. The result is a pure number, a scalar. This product is called the *scalar product* of the vectors, also called the *inner product*.

If we reverse the order of multiplication of these two vectors, we obtain

$$\begin{bmatrix} 4 \\ -1 \\ 3 \end{bmatrix} * [1 \quad 3 \quad -2] = \begin{bmatrix} 4 & 12 & -8 \\ -1 & -3 & 2 \\ 3 & 9 & -6 \end{bmatrix}.$$

This product is called the *outer product.* Although not as well known as the inner product, the outer product is very important in nonlinear optimization problems.*

A vector whose length is one is called a *unit vector.*† A vector that has all of its elements equal to zero is the *zero vector.* If all elements are zero except one, it is a *unit basis vector.* There are three distinct unit basis vectors of order-3:

$$\begin{bmatrix}1\\0\\0\end{bmatrix}, \quad \begin{bmatrix}0\\1\\0\end{bmatrix}, \quad \text{and} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

Some Special Matrices and Their Properties

Square matrices are particularly important when a system of equations is to be solved. Square matrices have some special properties.

The elements on the *main diagonal* are those from the upper-left corner to the lower-right corner. These are commonly referred to just as the *diagonal elements;* most often, just the word *diagonal* is used. If all elements except those on the diagonal are zero, the matrix is called a *diagonal matrix.*

If the nonzero elements of a diagonal matrix all are equal to one, the matrix is called the *identity matrix of order n* where n equals the number of row and columns. The usual symbol for this identity matrix is I_n and it has properties similar to unity. For example, the order-4 identity matrix is

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} = I_4.$$

The subscript is omitted when the order is clear from the context.

An important property of an identity matrix, I, is that for any $n \times n$ matrix, A, it is always true that

$$I * A = A * I = A.$$

If two rows of an identity matrix are interchanged, it is called a *transposition matrix.* (We also get a transposition matrix by interchanging two columns.) The name is appropriate because, if transposition matrix P_1 is multiplied with a square matrix of the same size, A, the product $P_1 * A$ will be the A matrix but with the same two rows interchanged. Here is an example:

$$P_1 = \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix}, \quad A = \begin{bmatrix}9&6&2&13\\4&2&8&11\\0&7&1&9\\3&2&6&8\end{bmatrix}, \quad P_1 * A = \begin{bmatrix}9&6&2&13\\3&2&6&8\\0&7&1&9\\4&2&8&11\end{bmatrix}.$$

*Another important product of three-component vectors is the *vector product,* also known as the *cross product.*

† The length of a vector is the square root of the sum of the squares of its components, an extension of the idea of the length of a two-component vector drawn from the origin.

However, if the two matrices are multiplied in reverse order, $A * P_1$, the result will be matrix A but with the columns of A interchanged. You should check this for yourself with the example matrices.

A *permutation matrix* is obtained by multiplying several transposition matrices.

A square matrix is called a *symmetric matrix* when the pairs of elements in similar positions across the diagonal are equal. Here is an example:

$$\begin{bmatrix} 1 & x & y \\ x & 2 & z \\ y & z & 3 \end{bmatrix}$$

The transpose of a matrix is the matrix obtained by writing the rows as columns or by writing the columns as rows. (A matrix does not have to be square to have a transpose.) The symbol for the transpose of matrix A is A^T. Example 2.1 illustrates.

EXAMPLE 2.1

$$A = \begin{bmatrix} 3 & -1 & 4 \\ 0 & 2 & -3 \\ 1 & 1 & 2 \end{bmatrix}; \quad A^T = \begin{bmatrix} 3 & 0 & 1 \\ -1 & 2 & 1 \\ 4 & -3 & 2 \end{bmatrix}.$$

It should be clear that $A = A^T$ if A is symmetric, and that for any matrix, the transpose of the transpose, $(A^T)^T$, is just A itself. It is also true, though not so obvious, that

$$(A * B)^T = B^T * A^T.$$

When a matrix is square, a quantity called its *trace* is defined. The trace of a square matrix is the sum of the elements on its main diagonal. For example, the traces of the previous matrices are

$$\text{tr}(A) = \text{tr}(A^T) = 3 + 2 + 2 = 7.$$

It should be obvious that the trace remains the same if a square matrix is transposed.

If all the elements above the diagonal are zero, a matrix is called *lower-triangular;* it is called *upper-triangular* when all the elements below the diagonal are zero. For example, these order-3 matrices are lower- and upper-triangular:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 6 & 0 \\ -2 & 1 & -4 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & -3 & 3 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Triangular matrices are of special importance, as will become apparent later in this chapter and in several other chapters.

Tridiagonal matrices are those that have nonzero elements only on the diagonal and in the positions adjacent to the diagonal; they will be of special importance in certain partial-differential equations. An example of a tridiagonal matrix is

$$\begin{bmatrix} -4 & 2 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 2 & -4 \end{bmatrix}.$$

For a tridiagonal matrix, only the nonzero values need to be recorded, and that means that the $n \times n$ matrix can be compressed into a matrix of 3 columns and n rows. For this example, we can write the matrix as

$$\begin{bmatrix} x & -4 & 2 \\ 1 & -4 & 1 \\ 1 & -4 & 1 \\ 1 & -4 & 1 \\ 2 & -4 & x \end{bmatrix}.$$

(The x entries are not normally used; they might be entered as zeros.)

In some important applied problems, only a few of the elements are nonzero. Such a matrix is termed a *sparse matrix* and procedures that take advantage of this sparseness are of value.

Examples of Operations with Matrices

Here are some examples of matrix operations:

$$3 * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 9 & 12 \end{bmatrix}.$$

$$\begin{bmatrix} 1 & 3 & 2 \\ -1 & 0 & 4 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 2 \\ 4 & 1 & -3 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 4 \\ 3 & 1 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} 2 & 1 \\ 0 & -4 \\ 7 & 2 \end{bmatrix} - \begin{bmatrix} 3 & -2 \\ 4 & 1 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} -1 & 3 \\ -4 & -5 \\ 7 & 4 \end{bmatrix}.$$

$$\begin{bmatrix} 2 & 0 & -1 \\ 3 & 2 & 6 \end{bmatrix} * \begin{bmatrix} -1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 7 \end{bmatrix}, \text{ but } \begin{bmatrix} 6 & -1 \\ 3 & -2 \end{bmatrix} * \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix} \text{ is undefined.}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} * \begin{bmatrix} 0 & 3 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -3 & 6 \\ 1 & 5 \end{bmatrix}, \text{ but } \begin{bmatrix} 0 & 3 \\ -1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 3 \\ 2 & -1 \end{bmatrix} = \begin{bmatrix} 6 & -3 \\ 1 & -4 \end{bmatrix}.$$

Division of matrices is not defined, but we will discuss the *inverse* of a matrix later in this chapter.

The *determinant* of a square matrix is a number. For a 2×2 matrix, the determinant is computed by subtracting the product of the elements on the minor diagonal (from upper right to lower left) from the product of terms on the major diagonal. For example,

$$A = \begin{bmatrix} -5 & 3 \\ 7 & 2 \end{bmatrix}, \qquad \det(A) = (-5)(2) - (7)(3) = -31;$$

$\det(A)$ is the usual notation for the determinant of A. Sometimes the determinant is symbolized by writing the elements of the matrix between vertical lines (similar to representing the absolute value of a number).

For a 3×3 matrix, you may have learned a crisscross way of forming products of terms (we call it the "spaghetti rule") that probably should be forgotten, for it applies only to the special case of a 3×3 matrix; it won't work for larger systems. The general rule that applies in all cases is to expand in terms of the minors of some row or column. The *minor* of any term is the matrix of lower order formed by striking out the row and column in which the term is found. The determinant is found by adding the product of each term in any row or column by the determinant of its minor, with signs alternating + and −. We expand each of the determinants of the minor until we reach 2×2 matrices. For example,

$$\text{Given } A = \begin{bmatrix} 3 & 0 & -1 & 2 \\ 4 & 1 & 3 & -2 \\ 0 & 2 & -1 & 3 \\ 1 & 0 & 1 & 4 \end{bmatrix},$$

$$\det(A) = 3\begin{vmatrix} 1 & 3 & -2 \\ 2 & -1 & 3 \\ 0 & 1 & 4 \end{vmatrix} - 0\begin{vmatrix} 4 & 3 & -2 \\ 0 & -1 & 3 \\ 1 & 1 & 4 \end{vmatrix}$$

$$+ (-1)\begin{vmatrix} 4 & 1 & -2 \\ 0 & 2 & 3 \\ 1 & 0 & 4 \end{vmatrix} - 2\begin{vmatrix} 4 & 1 & 3 \\ 0 & 2 & -1 \\ 1 & 0 & 1 \end{vmatrix}$$

$$= 3\left\{(1)\begin{vmatrix} -1 & 3 \\ 1 & 4 \end{vmatrix} - (3)\begin{vmatrix} 2 & 3 \\ 0 & 4 \end{vmatrix} + (-2)\begin{vmatrix} 2 & -1 \\ 0 & 1 \end{vmatrix}\right\}$$

$$- 0 + (-1)\left\{(4)\begin{vmatrix} 2 & 3 \\ 0 & 4 \end{vmatrix} - (1)\begin{vmatrix} 0 & 3 \\ 1 & 4 \end{vmatrix} + (-2)\begin{vmatrix} 0 & 2 \\ 1 & 0 \end{vmatrix}\right\}$$

$$- 2\left\{(4)\begin{vmatrix} 2 & -1 \\ 0 & 1 \end{vmatrix} - (1)\begin{vmatrix} 0 & -1 \\ 1 & 1 \end{vmatrix} + (3)\begin{vmatrix} 0 & 2 \\ 1 & 0 \end{vmatrix}\right\}$$

$$= 3\{(1)(-7) - (3)(8) + (-2)(2)\} - 0 + (-1)\{(4)(8) - (1)(-3) + (-2)(-2)\}$$
$$- 2\{(4)(2) - (1)(1) + (3)(-2)\}$$
$$= 3(-7 - 24 - 4) - 0 + (-1)(32 + 3 + 4) - 2(8 - 1 - 6)$$
$$= 3(-35) - 0 + (-1)(39) - 2(1) = -146.$$

In computing the determinant, the expansion can be about the elements of any row or column. To get the signs, give the first term a plus sign if the sum of its column number and row number is even; give it a minus if the sum is odd, with alternating signs thereafter. (For example, in expanding about the elements of the third row we begin with a plus; the first element a_{31} has $3 + 1 = 4$, an even number.) Judicious selection of rows and columns with many zeros can hasten the process, but this method of calculating determinants is a lot of work if the matrix is of large size. Methods that triangularize a matrix, as described in Section 2.2, are much better ways to get the determinant.

If a matrix, B, is triangular (either upper or lower), its determinant is just the product of the diagonal elements: $\det(B) = \Pi B_{ii}$, $i = 1, \ldots, n$. It is easy to show this if the determinant of the triangular matrix is expanded by minors. The following example illustrates this:

$$\det\begin{bmatrix} 4 & 0 & 0 \\ 6 & -2 & 0 \\ 1 & -3 & 5 \end{bmatrix} = 4 * \det\begin{bmatrix} -2 & 0 \\ -3 & 5 \end{bmatrix} + 0 + 0$$
$$= 4 * (-2) * |5| = 4 * (-2) * 5 = -40.$$

Determinants can be used to obtain the *characteristic polynomial* and the *eigenvalues* of a matrix, which are the roots of that polynomial. In Chapter 6, you will see that these are important in solving certain differential equations. The Greek symbol λ is commonly used to represent an eigenvalue. (Eigenvalue is a German word, the corresponding English term is *characteristic value,* but it is less frequently used.)

The two terms, eigenvalue and characteristic polynomial are interrelated: For matrix A, $P_A(\lambda) = \det(A - \lambda I)$.

For example, if

$$A = \begin{bmatrix} 1 & 3 \\ 4 & 5 \end{bmatrix},$$

then

$$\begin{aligned} P_A(\lambda) = |A - \lambda I| &= \det\begin{bmatrix} (1-\lambda) & 3 \\ 4 & (5-\lambda) \end{bmatrix} \\ &= (1-\lambda)(5-\lambda) - 12 \\ &= \lambda^2 - 6\lambda - 7. \end{aligned}$$

(The characteristic polynomial is always of degree n if A is $n \times n$.) If we set the characteristic polynomial to zero and solve for the roots, we get the eigenvalues of A. For this example, these are $\lambda_1 = 7$, $\lambda_2 = -1$, or, in more symbolic mathematical notation,

$$\Lambda(A) = \{7, \quad -1\}.$$

We also mention the notion of an *eigenvector* corresponding to an eigenvalue. The eigenvector is a nonzero vector w such that

$$Aw = \lambda w, \qquad \text{that is,} \qquad (A - \lambda I)w = 0. \tag{2.1}$$

In the current example, the eigenvectors are

$$w_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad w_2 = \begin{bmatrix} -3 \\ 2 \end{bmatrix}.$$

We leave it as an exercise to show that these eigenvectors satisfy Eq. (2.1).

Observe that the trace of A is equal to the sum of the eigenvalues: $\text{tr}(A) = 1 + 5 = \lambda_1 + \lambda_2 = 7 + (-1) = 6$. This is true for any matrix: The sum of its eigenvalues equals its trace.

For now, we limit the finding of eigenvalues and eigenvectors to small matrices because getting these through the characteristic polynomial is not recommended for matrices larger than 4×4. In Chapter 6 we examine other, more efficient ways to get these important quantities.

If a matrix is triangular, its eigenvalues are equal to the diagonal elements. This follows from the fact that its determinant is just the product of the diagonal elements and its characteristic polynomial is the product of the terms $(a_{i,i} - \lambda)$ with i going from 1 to n, the number of rows of the matrix. This simple example illustrates for a 3×3 matrix, A:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}, \ \det(A - \lambda I) = \det \begin{bmatrix} (1-\lambda) & 2 & 3 \\ 0 & (4-\lambda) & 5 \\ 0 & 0 & (5-\lambda) \end{bmatrix}$$
$$= (1-\lambda)(4-\lambda)(6-\lambda),$$

whose roots are clearly 1, 4, and 6. It does not matter if the matrix is upper- or lower-triangular.

Using Computer Algebra Systems

MATLAB can do matrix operations. We first define two matrices and a vector, A, B, and v:

```
EDU>> A = [4 1 -2; 5 1 3; 4 0 -1]
A =
  4   1   -2
  5   1    3
  4   0   -1
EDU>> B = [3 3 1; -2 1 5; 2 2 0]
B =
   3   3   1
  -2   1   5
   2   2   0
EDU>> v = [-2 3 1]
v =
  -2   3   1
```

Now we so some operations:

```
EDU>> 3*A
ans =
    12   3   -6
    15   3    9
    12   0   -3
```

```
EDU>> A + B
ans =
    7    4    -1
    3    2     8
    6    2    -1
EDU>> B - A
ans =
    -1    2    3
    -7    0    2
    -2    2    1
EDU>> A*B
ans =
     6     9     9
    19    22    10
    10    10     4
EDU>> B*v
??? Error using ==> *
Inner matrix dimensions must agree.
```

We can't multiply by the row vector, but we could with a column vector—so we transpose the vector:

```
EDU>> vt = v'
vt =
   -2
    3
    1
EDU>> B*Vt
ans =
    4
   12
    2
```

and now it works. Here are some other operations:

```
EDU>> det(A)
ans =
   21
EDU>> v*vt
ans =
   14
EDU>> vt*v
ans =
     4    -6    -2
    -6     9     3
    -2     3     1
```

```
EDU>> trace(A)
ans =
  4
```

We can get the characteristic polynomial:

```
EDU>> poly(A)
ans =
  1.0000    -4.0000    2.0000    -21.0000
```

where the coefficients are given. This represents

$$x^4 - 4x^3 + 2x - 21.$$

Using Maple

Maple and MATLAB are interrelated and MATLAB commands can be invoked in Maple and vice versa, but Maple can do matrix manipulations on its own. Here are a few—Maple's commands are somewhat different. Most need to be preceded by `with(linalg)`.

```
matadd (A, B)     does A + B
multiply (A, B) does A * B
  (but evalm ( . . . ) is more versatile, does all arithmetic
    operations with matrices, vectors, and scalars.)
trace (A)         gets the trace
transpose (A)     transposes
det (A)           gets the determinant
```

2.2 Elimination Methods

We now discuss numerical methods that are used to solve a set of linear equations. The term *linear equation* means an equation in several variables where all of the variables occur to the first power and there are no terms involving transcendental functions such as sine and cosine.

It used to be that students were taught to use Cramer's rule, in which a system can be solved through the use of determinants. However, Cramer's rule is inefficient and is almost impossible to use if there are more than two or three equations. As we have said, most applied problems are modeled with large systems and we present methods that work well with them. Even so, we use small systems as illustrations.

Suppose we have a system of equations that is of a special form, an *upper-triangular system,* such as

$$\begin{aligned} 5x_1 + 3x_2 - 2x_3 &= -3, \\ 6x_2 + x_3 &= -1, \\ 2x_3 &= 10. \end{aligned}$$

Whenever a system has this special form, its solution is very easy to obtain. From the third equation we see that $x_3 = 5$. Substituting this value into the second equation quickly gives $x_2 = -1$. Then substituting both values into the first equation reveals that $x_1 = 2$; now we have the solution: $x_1 = 2, x_2 = -1, x_3 = 5$.*

The first objective of the *elimination method* is to change the matrix of coefficients so that it is upper triangular. Consider this example of three equations:

$$\begin{aligned} 4x_1 - 2x_2 + x_3 &= 15, \\ -3x_1 - x_2 + 4x_3 &= 8, \\ x_1 - x_2 + 3x_3 &= 13. \end{aligned} \tag{2.2}$$

Multiplying the first equation by 3 and the second by 4 and adding these will eliminate x_1 from the second equation. Similarly, multiplying the first by -1 and the third by 4 and adding eliminates x_1 from the third equation. (We prefer to multiply by the negatives and add, to avoid making mistakes when subtracting quantities of unlike sign.) The result is

$$\begin{aligned} 4x_1 - 2x_2 + x_3 &= 15, \\ -10x_2 + 19x_3 &= 77, \\ -2x_2 + 11x_3 &= 37. \end{aligned}$$

We now eliminate x_2 from the third equation by multiplying the second by 2 and the third by -10 and adding to get

$$\begin{aligned} 4x_1 - 2x_2 + x_3 &= 15, \\ -10x_2 + 19x_3 &= 77, \\ -72x_3 &= -216. \end{aligned}$$

Now we have a triangular system and the solution is readily obtained; obviously $x_3 = 3$ from the third equation, and *back-substitution* into the second equation gives $x_2 = -2$. We continue with back-substitution by substituting both x_2 and x_3 into the first equation to get $x_1 = 2$.

The essence of any elimination method is to reduce the coefficient matrix to a triangular matrix and then use back-substitution to get the solution.

We now present the same problem, solved in exactly the same way, in matrix notation:

$$\begin{bmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 15 \\ 8 \\ 13 \end{bmatrix}.$$

The arithmetic operations that we have performed affect only the coefficients and the right-hand-side terms, so we work with the matrix of coefficients *augmented* with the

* A system that is *lower-triangular* is equally easy to solve. We then do *forward-substitution* rather than *back-substitution*.

right-hand-side vector:

$$A|b = \left[\begin{array}{rrr:r} 4 & -2 & 1 & 15 \\ -3 & -1 & 4 & 8 \\ 1 & -1 & 3 & 13 \end{array}\right]$$

(The dashed line is usually omitted.)

We perform elementary row transformations* to convert A to *upper-triangular* form:

$$\begin{bmatrix} 4 & -2 & 1 & 15 \\ -3 & -1 & 4 & 8 \\ 1 & -1 & 3 & 13 \end{bmatrix}, \quad \begin{array}{r} \\ 3R_1 + 4R_2 \rightarrow \\ (-1)R_1 + 4R_3 \rightarrow \end{array} \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -10 & 19 & 77 \\ 0 & -2 & 11 & 37 \end{bmatrix},$$

$$\begin{array}{r} \\ \\ 2R_2 - 10R_3 \rightarrow \end{array} \begin{bmatrix} 4 & -2 & 1 & 15 \\ 0 & -10 & 19 & 77 \\ 0 & 0 & -72 & -216 \end{bmatrix}. \qquad (2.3)$$

The steps here are to add 3 times the first row to 4 times the second row and to add -1 times the first row to 4 times the third row. The next and final phase (in order to get a triangular system) is to add 2 times the second row to -10 times the third row.

The array in Eq. (2.3) represents the equations

$$\begin{aligned} 4x_1 - 2x_2 + x_3 &= 15, \\ -10x_2 + 19x_3 &= 77, \\ -72x_3 &= -216. \end{aligned} \qquad (2.4)$$

The back-substitution step can be performed quite mechanically by solving the equations of Eq. (2.4) in reverse order. That is,

$$\begin{aligned} x_3 &= -216/(-72) = 3, \\ x_2 &= (77 - 3(19))/(-10) = -2, \\ x_1 &= (15 - 1(3) - (-2)(-2))/4 = 2. \end{aligned}$$

Thinking of the procedure in terms of matrix operations, we transform the augmented coefficient matrix by elementary row operations until a triangular matrix is created on the left. After back-substitution, the x-vector stands as the rightmost column.†

These operations, which do not change the relationships represented by a set of equations, can be applied to an augmented matrix, because this is only a different notation for the equations. (We need to add one proviso: Because round-off error is related to the magnitude of the values when we express them in fixed-word-length computer

* Elementary row operations are arithmetic operations that obviously are valid rearrangements of a set of equations: (1) Any equation can be multiplied by a constant; (2) the order of the equations can be changed; (3) any equation can be replaced by its sum with another of the equations.

† Making the matrix triangular by row operations is not a way to get its eigenvalues; the row operations change them.

representations, some of our operations may have an effect on the accuracy of the computed solution.)

An alternative to converting the system to upper-triangular is to make it lower-triangular. For example, if we have this set of equations (or an augmented matrix):

$$\begin{aligned} 6x_1 \qquad\qquad &= 18 \\ -2x_1 + 5x_2 \qquad &= 2 \\ 3x_1 - 4x_2 + 3x_3 &= 11 \end{aligned} \qquad \text{or} \qquad \begin{bmatrix} 6 & 0 & 0 \\ -2 & 5 & 0 \\ 3 & -4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 2 \\ 11 \end{bmatrix}.$$

Here, we would solve for the variables in this order: x_1, then x_2, and finally x_3, with the same number of computations as in the case of a lower-triangular system. Both the lower- and upper-triangular systems play an important part in the development of algorithms in the following sections, because these systems require fewer multiplications/divisions than the general system. We shall also show that we can often write a general matrix A as the product LU, a lower-triangular matrix times an upper-triangular matrix.

Note that there exists the possibility that the set of equations has no solution, or that the prior procedure will fail to find it. During the triangularization step, if a zero is encountered on the diagonal, we cannot use that row to eliminate coefficients below that zero element. However, in that case, we can continue by interchanging rows and eventually achieve an upper-triangular matrix of coefficients. The real stumbling block is finding a zero on the diagonal after we have triangularized. If that occurs, the back-substitution fails, for we cannot divide by zero. It also means that the determinant is zero: There is no solution.

It is worthwhile to explain in more detail what we mean by the *elementary row operations* that we have used here, and to see why they can be used in solving a linear system. There are three of these operations:

1. We may multiply any row of the augmented coefficient matrix by a constant.
2. We can add a multiple of one row to a multiple of any other row.
3. We can interchange the order of any two rows (this was not used earlier).

The validity of these row operations is intuitively obvious if we think of them applied to a set of linear equations. Certainly, multiplying one equation through by a constant does not change the truth of the equality. Adding equal quantities to both sides of an equality results in an equality, and this is the equivalent of the second transformation. Obviously, the order of the set is arbitrary, so rule 3 is valid.

Gaussian Elimination

The procedure just described has a major problem. While it may be satisfactory for hand computations with small systems, it is inadequate for a large system. Observe that the transformed coefficients can become very large as we convert to a triangular system. The method that is called *Gaussian elimination* avoids this by subtracting a_{i1}/a_{11} times the first equation from the ith equation to make the transformed numbers in the first column equal to zero. We do similarly for the rest of the columns.

We must always guard against dividing by zero. Observe that zeros may be created in the diagonal positions even if they are not present in the original matrix of coefficients. A useful strategy to avoid (if possible) such zero divisors is to rearrange the equations so as to put the coefficient of largest magnitude on the diagonal at each step. This is called *pivoting*. Complete pivoting may require both row and column interchanges. This is not frequently done. Partial pivoting, which places a coefficient of larger magnitude on the diagonal by row interchanges only, will guarantee a nonzero divisor if there is a solution to the set of equations, and will have the added advantage of giving improved arithmetic precision. The diagonal elements that result are called *pivot elements*. (When there are large differences in magnitude of coefficients in one equation compared to the other equations, we may need to *scale* the values; we consider this later.)

We repeat the example of the previous section, incorporating these ideas and carrying four significant digits in our work. We begin with the augmented matrix.

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ -3 & -1 & 4 & 8 \\ 1 & -1 & 3 & 13 \end{array}\right]$$

$$\begin{array}{r} \\ R_2 - (-\tfrac{3}{4})R_1 \rightarrow \\ R_3 - (\tfrac{1}{4})R_1 \rightarrow \end{array} \left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -2.5 & 4.75 & 19.25 \\ 0 & -0.5 & 2.75 & 9.25 \end{array}\right]$$

$$\begin{array}{r} \\ \\ R_3 - (-0.5/-2.5)R_2 \rightarrow \end{array} \left[\begin{array}{ccc|c} 4 & -2 & 1 & 15 \\ 0 & -2.5 & 4.75 & 19.25 \\ 0 & 0.0 & 1.80 & 5.40 \end{array}\right]$$

[The notation used here means to subtract $(-3/4)$ times row 1 from row 2 and to subtract $(1/4)$ times row 1 from row 3 in reducing in the first column; to subtract $(-0.5/-2.5)$ times row 2 from row 3 in the third column.]

The method we have just illustrated is called *Gaussian elimination*. (In this example, no pivoting was required to make the largest coefficients be on the diagonal.) Back-substitution, as presented with Eq. (2.4), gives us, as before, $x_3 = 3$, $x_2 = -2$, $x_1 = 2$. We have come up with the exact answer to this problem. Often it will turn out that we shall obtain answers that are just close approximations to the exact answer because of round-off error. When there are many equations, the effects of round-off (the term is applied to the error due to chopping as well as when rounding is used) may cause large effects. In certain cases, the coefficients are such that the results are particularly sensitive to round off; such systems are called *ill-conditioned*.

In the example just presented, the zeros below the main diagonal show that we have reduced the problem [Eq. (2.3)] to solving an upper-triangular system of equations as in Eqs. (2.4). However, at each stage, if we had stored the ratio of coefficients in place of zero (we show these in parentheses), our final form would have been

$$\left[\begin{array}{ccc:c} 4 & -2 & 1 & 15 \\ (-0.75) & -2.5 & 4.75 & 19.25 \\ (0.25) & (0.20) & 1.80 & 5.4 \end{array}\right].$$

Then, in addition to solving the problem as we have done, we find that the original matrix

$$A = \begin{bmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{bmatrix}$$

can be written as the product:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -0.75 & 1 & 0 \\ 0.25 & 0.20 & 1 \end{bmatrix}}_{L} * \underbrace{\begin{bmatrix} 4 & -2 & 1 \\ 0 & -2.5 & 4.75 \\ 0 & 0 & 1.80 \end{bmatrix}}_{U}. \qquad \textbf{(2.5)}$$

This procedure is called a *LU decomposition* of A. In this case,

$$A = L * U,$$

where L is *lower-triangular* and U is *upper-triangular.* As we shall see in the next example, usually $L * U = A'$, where A' is just a permutation of the rows of A due to row interchange from pivoting.

Finally, because the determinant of two matrices, $B * C$, is the product of each of the determinants, for this example we have

$$\det(L * U) = \det(L) * \det(U) = \det(U),$$

because L is triangular and has only ones on its diagonal so that $\det(L) = 1$. Thus, for the example given in Eq. (2.5), we have

$$\det(A) = \det(U) = (4) * (-2.5) * (1.8) = -18,$$

because U is upper-triangular and its determinant is just the product of the diagonal elements.

From this example, we see that Gaussian elimination does the following:

1. It solves the system of equations.
2. It computes the determinant of a matrix very efficiently.
3. It can provide us with the LU decomposition of the matrix of coefficients, in the sense that the product of the two matrices, $L * U$, may give us a permutation of the rows of the original matrix.

With regard to item 2, when there are row interchanges,

$$\det(A) = (-1)^m * u_{11} * \cdots * u_{nn},$$

where the exponent m represents the number of row interchanges.

We summarize the operations of Gaussian elimination in a form that will facilitate the writing of a computer program. Note that in the actual implementation of the algorithm, the L and U matrices are actually stored in the space of the original matrix A.

Gaussian Elimination

To solve a system of n linear equations: $Ax = b$.

```
For $j = 1$ To $(n - 1)$
    pvt = $|a[j, j]|$
    pivot $[j] = j$
    ipvt_temp $= j$

    For $i = j + 1$ To $n$        (Find pivot row)
        IF $|a[i, j]| >$ pvt   Then
            pvt = $|a[i, j]|$
            ipvt_temp $= i$
        End IF
    End For $i$

    (Switch rows if necessary)
    IF pivot $[j]$ < > ipvt_temp
        [switch_rows(rows $j$ and ipvt_temp)]

    For $i = j + 1$ to $n$        (Store multipliers)
        $a[i, j] = a[i, j]/a[j, j]$
    End For $i$

    (Create zeros below the main diagonal)
    For $i = j + 1$ To $n$
        For $k = j + 1$ To $n$
            $a[i, k] = a[i, k] - a[i, j] * a[j, k]$
        End For $k$
        $b[i] = b[i] - a[i, j] * b[j]$
    End For $i$

End For $j$;

(Back Substitution Part)
$x[n] = b[n]/a[n, n]$
For $j = n - 1$ Down To 1
    $x[j] = b[j]$
    For $k = n$ Down To $j + 1$
        $x[j] = x[j] - x[k] * a[j, k]$
    End For $k$
    $x[j] = x[j]/a[j, j]$

End For $j$.
```

Interchanging rows in a large matrix can be expensive; there is a better way. We keep track of the order of the rows in an *order vector* and, when a row interchange is indicated, we only interchange the corresponding elements in the order vector. This vector then tells which rows are to be worked on. Using an order vector saves computer time because only two numbers of this vector are interchanged; we do not have to switch all the elements of the two rows. However, we do not do this here in order to keep our explanations simple. You will later see an example that uses an order vector.

The algorithm for Gaussian elimination will be clarified by an additional numerical example. Solve the following system of equations using Gaussian elimination. In addition, compute the determinant of the coefficient matrix and the *LU* decomposition of this matrix.

Given the system of equations, solve

$$\begin{aligned} 2x_2 \qquad\quad + x_4 &= 0, \\ 2x_1 + 2x_2 + 3x_3 + 2x_4 &= -2, \\ 4x_1 - 3x_2 \qquad\quad + x_4 &= -7, \\ 6x_1 + x_2 - 6x_3 - 5x_4 &= 6. \end{aligned} \tag{2.6}$$

The augmented coefficient matrix is

$$\begin{bmatrix} 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 3 & 2 & -2 \\ 4 & -3 & 0 & 1 & -7 \\ 6 & 1 & -6 & -5 & 6 \end{bmatrix}. \tag{2.7}$$

We cannot permit a zero in the a_{11} position because that element is the pivot in reducing the first column. We could interchange the first row with any of the other rows to avoid a zero divisor, but interchanging the first and fourth rows is our best choice. This gives

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ 2 & 2 & 3 & -2 & -2 \\ 4 & -3 & 0 & 1 & -7 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix}. \tag{2.8}$$

We make all the elements in the first column zero by subtracting the appropriate multiple of row one:

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ 0 & 1.6667 & 5 & 3.6667 & -4 \\ 0 & -3.6667 & 4 & 4.3333 & -11 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix}. \tag{2.9}$$

We again interchange before reducing the second column, not because we have a zero divisor, but because we want to preserve accuracy.* Interchanging the second and third rows

* A numerical example that demonstrates the improved accuracy when partial pivoting is used will be found in Section 2.4.

puts the element of largest magnitude on the diagonal. (We could also interchange the fourth column with the second, giving an even larger diagonal element, but we do not do this.) After the interchange, we have

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ 0 & -3.6667 & 4 & 4.3333 & -11 \\ 0 & 1.6667 & 5 & 3.6667 & -4 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix}. \tag{2.10}$$

Now we reduce in the second column

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ 0 & -3.6667 & 4 & 4.3333 & -11 \\ 0 & 0 & 6.8182 & 5.6364 & -9.0001 \\ 0 & 0 & 2.1818 & 3.3636 & -5.9999 \end{bmatrix}.$$

No interchange is indicated in the third column. Reducing, we get

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ 0 & -3.6667 & 4 & 4.3333 & -11 \\ 0 & 0 & 6.8182 & 5.6364 & -9.0001 \\ 0 & 0 & 0 & 1.5600 & -3.1199 \end{bmatrix}. \tag{2.11}$$

Back-substitution gives

$$\boxed{\begin{aligned} x_4 &= \frac{-3.1199}{1.5600} = -1.9999, \\ x_3 &= \frac{-9.0001 - 5.6364(-1.9999)}{6.8182} = 0.33325, \\ x_2 &= \frac{-11 - 4.3333(-1.9999) - 4(0.33325)}{-3.6667} = 1.0000, \\ x_1 &= \frac{6 - (-5)(-1.9999) - (-6)(0.33325) - (1)(1.0000)}{6} = -0.50000. \end{aligned}}$$

The correct answers are -2, $\frac{1}{3}$, 1, and $-\frac{1}{2}$ for x_4, x_3, x_2, and x_1. In this calculation we have carried five significant figures and rounded each calculation. Even so, we do not have five-digit accuracy in the answers. The discrepancy is due to round off. The question of the accuracy of the computed solution to a set of equations is a most important one, and at several points in the following discussion we will discuss how to minimize the effects of round off and avoid conditions that can cause round-off errors to be magnified.

In this example, if we had replaced the zeros below the main diagonal with the ratio of coefficients at each step, the resulting augmented matrix would be

$$\begin{bmatrix} 6 & 1 & -6 & -5 & 6 \\ (0.66667) & -3.6667 & 4 & 4.3333 & -11 \\ (0.33333) & (-0.45454) & 6.8182 & 5.6364 & -9.0001 \\ (0.0) & (-0.54545) & (0.32) & 1.5600 & -3.1199 \end{bmatrix}. \qquad \textbf{(2.12)}$$

This gives a *LU* decomposition as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.66667 & 1 & 0 & 0 \\ 0.33333 & -0.45454 & 1 & 0 \\ 0.0 & -0.54545 & 0.32 & 1 \end{bmatrix} * \begin{bmatrix} 6 & 1 & -6 & -5 \\ 0 & -3.6667 & 4 & 4.3333 \\ 0 & 0 & 6.8182 & 5.6364 \\ 0 & 0 & 0 & 1.5600 \end{bmatrix}. \qquad \textbf{(2.13)}$$

It should be noted that the product of the matrices in Eq. (2.13) produces a permutation of the original matrix, call it A', where

$$A' = \begin{bmatrix} 6 & 1 & -6 & -5 \\ 4 & -3 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 0 & 2 & 0 & 1 \end{bmatrix},$$

because rows 1 and 4 were interchanged in Eq. (2.8) and rows 2 and 3 in Eq. (2.10). The determinant of the original matrix of coefficients—the first four columns of Eq. (2.7)—can be easily computed from Eq. (2.11) or Eq. (2.12) according to the formula

$$\det(A) = (-1)^2 * (6) * (-3.6667) * (6.8182) * (1.5600) = -234.0028,$$

which is close to the exact solution: -234.* The exponent 2 is required, because there were two row interchanges in solving this system. To summarize, you should note that the Gaussian elimination method applied to Eq. (2.6) produces the following:

1. The solution to the four equations.
2. The determinant of the coefficient matrix

$$\begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 4 & -3 & 0 & 1 \\ 6 & 1 & -6 & -5 \end{bmatrix}.$$

* The difference is because the computed values have been rounded to four decimal places.

3. *A LU* decomposition of the matrix, A', which is just the original matrix, A, after we have interchanged its rows in the process.

MATLAB can get the matrices of Eq. (2.13) with its `lu` command:

```
EDU>> A = [0 2 0 1 0; 2 2 3 2 -2; 4 -3 0 1 -7; 6 1 -6 -5 6]
A =
   0     2     0     1     0
   2     2     3     2    -2
   4    -3     0     1    -7
   6     1    -6    -5     6
EDU>> [L,U,P] = lu(A)
L =
   1.0000      0          0         0
   0.6667      1.0000     0         0
   0.3333     -0.4545     1.0000    0
   0          -0.5455     0.3200    1.0000
U =
   6.0000      1.0000     -6.0000     -5.0000      6.0000
   0          -3.6667      4.0000      4.3333    -11.0000
   0           0           6.8182      5.6364     -9.0000
   0           0           0           1.5600     -3.1200
P =
   0     0     0     1
   0     0     1     0
   0     1     0     0
   1     0     0     0
```

In this, matrix P is the permutation matrix that was used to put the largest magnitude coefficient in the pivot position. Observe that MATLAB got a more accurate solution.

We really desire the solution to $Ax = b$. MATLAB gets this with a simple command. We define A and b (trailing semicolons suppress the outputs); the apostrophe on b gets the transpose:

```
EDU>> A = [0 2 0 1; 2 2 3 2; 4 -3 0 1; 6 1 -6 -5];
EDU>> b = [0 2 -7 6]';
EDU>> A\b
ans =
   -0.5000
    1.0000
    0.3333
   -2.0000
```

Again, we obtained a more accurate solution.

Operational Count

The efficiency of a numerical procedure is ordinarily measured by counting the number of arithmetic operations that are required. In the past, only multiplications and divisions were counted because they used to take much longer to perform than additions and subtractions. In today's computers using math coprocessors, all four of these take about the same time, so we should count them all.

In a system of n equations with just one right-hand side, we compute the number of operations as follows. The augmented matrix is $n \times n + 1$ in size.

To reduce the elements below the diagonal in column 1, we first compute $(n - 1)$ multiplying factors [takes $(n - 1)$ divides]. We multiply each of these by all the elements in row 1 except the first element [takes (n) multiplies)] and subtract these products from the n elements in each of the $n - 1$ rows below row 1, ignoring the first elements because these are known to become zero [takes $n * (n - 1)$ multiplies and the same number of subtracts]. In summary:

$$\text{Divides} = (n - 1),$$
$$\text{Multiplies} = n * (n - 1),$$
$$\text{Subtracts} = n * (n - 1).$$

In the other columns, we do similarly except each succeeding column has one fewer element. So, we have for column i:

$$\text{Divides: } (n - i),$$
$$\text{Multiplies: } (n - i + 1) * (n - i),$$
$$\text{Subtracts: } (n - i + 1)(n - i).$$

We add these quantities together for the reduction in columns 1 through $n - 1$ to get:

$$\text{Divides} = \sum_{i=1}^{n-1} (n - i) = \sum_{i=1}^{n-1} i = n^2/2 - n/2,$$

$$\text{Multiplies} = \sum_{i=1}^{n-1} (n - i + 1)(n - i) = \sum_{i=1}^{n-1} i(i + 1) = n^3/3 - n/3.$$

Subtracts are the same as multiplies $= n^3/3 - n/3$. If we add these together, we get, for the triangularization part, $2n^3/3 + n^2/2 - 7n/6$ total operations. In terms of the order relation discussed in Chapter 0, this is $O(n^3/3)$. We still need to do the back-substitutions. A little reflection shows that this requires

$$\text{Multiplies} = \sum_{i=1}^{n-1} i = n^2/2 - n/2,$$

$$\text{Subtracts} = \text{same as number of multiplies},$$

$$\text{Divides} = n,$$

so the back substitution requires a total of n^2 operations.

If we add the operations needed for the entire solution of a system of n equations, we get:

$$2n^3/3 + 3n^2/2 - 7n/6.$$

Multiple Right-Hand Sides

Gaussian elimination can readily work with more than one right-hand-side vector. We just append the additional column vectors within the augmented matrix and apply the reduction steps to these new columns exactly as we do to the first column. If row interchanges are made, the entire augmented matrix is included.

The Gauss-Jordan Method

There are many variants to the Gaussian elimination scheme. The back-substitution step can be performed by eliminating the elements above the diagonal after the triangularization has been finished, using elementary row operations and proceeding upward from the last row. This technique is equivalent to the procedures described in the following example. The diagonal elements may all be made ones as a first step before creating zeros in their column; this performs the divisions of the back-substitution phase at an earlier time.

One variant that is sometimes used is the *Gauss–Jordan* scheme. In it, the elements above the diagonal are made zero at the *same time* that zeros are created below the diagonal. Usually, the diagonal elements are made ones at the same time that the reduction is performed; this transforms the coefficient matrix into the identity matrix. When this has been accomplished, the column of right-hand sides has been transformed into the solution vector. Pivoting is normally employed to preserve arithmetic accuracy.

The previous example, solved by the Gauss–Jordan method, gives this succession of calculations. The original augmented matrix is

$$\begin{bmatrix} 0 & 2 & 0 & 1 & 0 \\ 2 & 2 & 3 & 2 & -2 \\ 4 & -3 & 0 & 1 & -7 \\ 6 & 1 & -6 & -5 & 6 \end{bmatrix}.$$

Interchanging rows 1 and 4, dividing the new first row by 6, and reducing the first column gives

$$\begin{bmatrix} 1 & 0.1667 & -1 & -0.8333 & 1 \\ 0 & 1.6667 & 5 & 3.3667 & -4 \\ 0 & -3.6667 & 4 & 4.3334 & -11 \\ 0 & 2 & 0 & 1 & 0 \end{bmatrix}.$$

Interchanging rows 2 and 3, dividing the new second row by −3.6667, and reducing the second column (operating above the diagonal as well as below) gives

$$\begin{bmatrix} 1 & 0 & -0.8182 & -0.6364 & 0.5 \\ 0 & 1 & -1.0909 & -1.1818 & 3 \\ 0 & 0 & 6.8182 & 5.6364 & -9 \\ 0 & 0 & 2.1818 & 3.3636 & -6 \end{bmatrix}.$$

No interchanges now are required. We divide the third row by 6.8182 and zero the other elements in the third column:

$$\begin{bmatrix} 1 & 0 & 0 & 0.04 & -0.58 \\ 0 & 1 & 0 & -0.280 & 1.56 \\ 0 & 0 & 1 & 0 & -1.32 \\ 0 & 0 & 0 & 1.5599 & -3.12 \end{bmatrix}.$$

We complete by dividing the fourth row by 1.5599 and create zeros above:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 0 & 1.0001 \\ 0 & 0 & 1 & 0 & 0.3333 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix}.$$

The fourth column is now the solution. It differs slightly from that obtained with Gaussian elimination; round-off errors have been entered in a different way.

While the Gauss–Jordan method might seem to require the same effort as Gaussian elimination, it really requires almost 50% more operations. As an exercise, you should show that it takes $(n^2 - n)/2$ divides, $(n^3 - n)/2$ multiplies, and $(n^3 - n)/2$ subtracts for a total of $n^3 + n^2 - 2n$ altogether. It is $O(n^3)$, compared to $O(2n^3/3)$.

Scaled Partial Pivoting*

There are times when the partial pivoting procedure is inadequate. When some rows have coefficients that are very large in comparison to those in other rows, partial pivoting may not give a correct solution. The answer to this problem is *scaling*, which means that we adjust the coefficients to make the largest in each row of the same magnitude.

Coefficients may differ in magnitude for several reasons. It might be caused by relations where the quantities are in widely different units: microvolts versus kilovolts, seconds versus years, for example. It could be due to inherently large numbers in just one equation. Here is a simple example to show how partial pivoting may not be enough:

$$\text{Given: } A = \begin{bmatrix} 3 & 2 & 100 \\ -1 & 3 & 100 \\ 1 & 2 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 105 \\ 102 \\ 2 \end{bmatrix}.$$

whose correct answer obviously is $x = [1.00, 1.00, 1.00]^T$.

* Sometimes this is called *virtual scaling*.

If we solve this using partial pivoting but with only three digits of precision to emphasize round-off error, we get this augmented matrix after triangularization:

$$\begin{bmatrix} 3 & 2 & 100 & 105 \\ 0 & 3.67 & 133 & 135 \\ 0 & 0 & -82.4 & -82.6 \end{bmatrix},$$

from which the solution vector is readily found to be the erroneous value of $[0.939, 1.09, 1.00]^T$.

The trouble here is that, while no pivoting appears to be needed during the solution, the coefficients in the third equation are much smaller than those in the other two. We can do scaled partial pivoting by first dividing each equation by its coefficient of largest magnitude. Doing so with the original equations gives (still using just three digits):

$$\begin{bmatrix} 0.0300 & 0.01 & 1.00 & 1.05 \\ -0.0100 & 0.03 & 1.00 & 1.02 \\ 0.500 & 1.00 & -0.500 & 1.00 \end{bmatrix}.$$

We now see that we should interchange row 1 with row 3 before we begin the reduction.

We could now solve this scaled set of equations but there is a better way that uses the original equations, eliminating the rounding off that may occur in obtaining the scaled equations. This method begins by computing a *scaling vector* whose elements are the elements in each row of largest magnitude. Calling the scaling vector S, we have for this example:

$$S = [100, 100, 2].$$

Before reducing the first column, we divide each element by the corresponding element of S to get $R = [0.0300, -0.0300, 0.500]$ in which the largest element is the third. This shows that the third equation should be the pivot and that the third equation should be interchanged with the first. (As you will see below, we do not have to actually interchange equations.) In preparation for further reduction steps, we do interchange the elements of S to get:

$$S' = [2, 100, 100].$$

The reduced matrix after reducing in the first column (still using only three digits of precision) is

$$\begin{bmatrix} 1 & 2 & -1 & 2 \\ 0 & 5 & 99 & 104 \\ 0 & -4 & 103 & 99 \end{bmatrix}.$$

We now are ready to reduce in column 2. We divide the elements of this column by the elements of S to get $R = [1, 0.0500, -0.0400]$. We ignore the first element of R, and see that no interchange is needed. Doing the reduction we get this final matrix:

$$\begin{bmatrix} 1 & 2 & -1 & 2 \\ 0 & 5 & 99 & 104 \\ 0 & 0 & 182 & 182 \end{bmatrix},$$

and back substitution gives the correct answer: $x = [1.00, 1.00, 1.00]$.

Using an Order Vector

We now give an example of using an order vector to avoid the actual interchange of rows. Our system in augmented matrix form is

$$\begin{bmatrix} 4 & -3 & 0 & -7 \\ 2 & 2 & 3 & -2 \\ 6 & 1 & -6 & 6 \end{bmatrix}.$$

Initially, we set the order vector to [1, 2, 3]. Looking at column 1, we see that $A(3, 1)$ should be the pivot; we exchange elements in the order vector to get [3, 2, 1]. In the reduction of column 1, we use row 3 as the pivot row to get

$$\begin{bmatrix} (0.6667) & -3.667 & 4 & -11 \\ (0.3333) & 1.667 & 5 & -4 \\ 6 & 1 & -6 & 6 \end{bmatrix}.$$

From here, we ignore row 3. We see that $A(1, 2)$ should be the next pivot. We exchange elements in the order vector to get [3, 1, 2] so as to use row 1 as the next pivot row. Reducing column 2, we then get

$$\begin{bmatrix} (0.6667) & -3.667 & 4 & -11 \\ (0.3333) & (-0.4545) & 6.8182 & -9 \\ 6 & 1 & -6 & 6 \end{bmatrix},$$

and "back-substituting" from the final set of equations in the order given by the final order vector: first 2, then 1, then 3, gives the solution: $x_2 = 1.5600$, $x_1 = -0.5800$, $x_3 = -1.3200$.

Using the *LU* Matrix for Multiple Right-Hand Sides

Many physical situations are modeled with a large set of linear equations: an example is determining the internal temperatures in a nuclear reactor, and knowing the maximum temperature is critical. The equations will depend on the geometry and certain external factors that will determine the right-hand sides. If we want the solution for many different values of these right-hand sides, it is inefficient to solve the system from the start with each one of the right-hand-side values—using the *LU* equivalent of the coefficient matrix is preferred.

Of course, getting the solutions for a problem with several right-hand sides can be done in ordinary Gaussian elimination by appending the several right-hand vectors to the coefficient matrix. However, when these vectors are not known in advance, we might think we would have to start from the beginning.

We can use the *LU* equivalent of the coefficient matrix to avoid this if we solve the system in two steps. Suppose we have solved the system $Ax = b$ by Gaussian elimination—we now know the *LU* equivalent of A: $A = L * U$. Consider now that we want to solve $Ax = b$ with some new b-vector. We can write

$$Ax = b = L * U * x = b.$$

The product of U and x is a vector, call it y. Now, we can solve for y from $Ly = b$ and this is readily done because L is lower-triangular and we get y by "forward-substitution." Call the solution $y = b'$.

Going back to the original $LUx = b$, we see that, from $Ux = y = b'$, we can get x from $Ux = b'$, which is again readily done by back-substitution (U is upper-triangular). The operational count for either forward- or back-substitution is exactly n^2 operations, so solving $Ax = b$ will take only $2n^2$ operations if the LU equivalent of A is already known, which is significantly less than the $2n^3/3 + 3n^2/2 - 7n/6$ operations required to solve $Ax = b$ directly.

What if we had reordered the rows of matrix A by pivoting? This is no problem if we save the order vector that tells the final order of the rows of the matrix. We then use this to rearrange the elements of the b-vector, or perhaps use it to select the proper elements during the forward- and back-substitutions.

AN EXAMPLE

Solve $Ax = b$, where we already have its L and U matrices:

$$L = \begin{bmatrix} 1.0000 & 0 & 0 & 0 \\ 0.6667 & 1.0000 & 0 & 0 \\ 0.3333 & -0.4545 & 1.0000 & 0 \\ 0 & -0.5455 & 0.3200 & 1.0000 \end{bmatrix},$$

$$U = \begin{bmatrix} 6.0000 & 1.0000 & -6.0000 & -5.0000 \\ 0 & -3.6667 & 4.0000 & 4.3333 \\ 0 & 0 & 6.8182 & 5.6364 \\ 0 & 0 & 0 & 1.5600 \end{bmatrix}.$$

Suppose that the b-vector is $[6, -7, -2, 0]^T$. We first get $y = Ux$ from $Ly = b$ by forward-substitution:

$$y = [6, -11, -9, -3.12]^T,$$

and use it to compute x from $Ux = y$:

$$x = [-0.5, 1, 0.3333, -2]^T.$$

[This is the same system as Eq. (2.6) but with the equations reordered so pivoting is not needed.]

Now, if we want the solution with a different b-vector:

$$bb = [1 \;\; 4 \;\; -3 \;\; 1]^T,$$

we just do $Ly = bb$ to get

$$y = [1, 3.3333, -1.8182, 3.4]^T,$$

and then use this y in $Ux = y$ to find the new x:

$$x = [0.0128, -0.5897, -2.0684, 2.1795]^T.$$

Tridiagonal Systems

There are some applied problems where the coefficient matrix is of a special form. Most of the coefficients are zero; only those on the diagonal and adjacent to the diagonal are non-zero. Such a matrix is called *tridiagonal*. Here is an example from Chapter 6:

$$\begin{bmatrix} \_2.70192 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & \_14.0385 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & -100.0 \end{bmatrix}.$$

When this system is solved by Gaussian elimination, the zeros do not enter into the solution; only the non-zero coefficients are used. That means that there is no need to store the zeros; we can compress the coefficients into an array of three columns and put the right-hand-side terms into a fourth column. The number of arithmetic operations is reduced significantly.

Here is an algorithm that carries out the solution of the problem:

Gaussian Elimination for a Tridiagonal System

```
Given the $n \times 4$ matrix that has the right-hand side as its fourth column,
  ($LU$ decomposition phase)

  For $i = 2$ To $n$
     $A[i, 1] = A[i, 1]/A[i - 1, 2]$
     $A[i, 2] = A[i, 2] - A[i, 1] * A[i - 1, 3]$
     $A[i, 4] = A[i, 4] - A[i, 1] * A[i - 1, 4]$

  End For $i$

  (Back-substitution)

  $A[n, 4] = A[n, 4]/A[n, 2]$
  For $i = (n - 1)$ Down To 1
     $A[i, 4] = (A[i, 4] - A[i, 3] * A[i + 1, 4])/A[i, 2]$µ
  End For $i$
```

2.3 The Inverse of a Matrix and Matrix Pathology

Division by a matrix is not defined but the equivalent is obtained from the *inverse* of the matrix. If the product of two square matrices, $A * B$, equals the identity matrix, I, B is said to be the inverse of A (and A is the inverse of B). The usual notation for the inverse of matrix A is A^{-1}. We have said that matrices do not commute on multiplication but inverses are an exception: $A * A^{-1} = A^{-1} * A$.

One way to find the inverse of matrix A is to employ the minors of its determinant but this is not efficient. The better way is to use an elimination method. We augment the A matrix with the identity matrix of the same size and solve. The solution is A^{-1}. This is equivalent to solving the system with n right-hand sides, each column being one of the n unit basis vectors in turn. Here is an example:

EXAMPLE 2.2 Given matrix A, find its inverse. First use the Gauss–Jordan method with exact arithmetic.

$$A = \begin{bmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix}.$$

Augment A with the identity matrix and then reduce:

$$\begin{bmatrix} 1 & -1 & 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 3 & -5 & -3 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\xrightarrow{(1)} \begin{bmatrix} 1 & -1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -5 & 0 & 1 & -3 \end{bmatrix} \xrightarrow{(2)} \begin{bmatrix} 1 & -1 & 0 & 1 & \frac{2}{5} & -\frac{6}{5} \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{2}{5} & -\frac{1}{5} \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix}.$$

We confirm the fact that we have found the inverse by multiplication:

$$\begin{bmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix} \begin{bmatrix} 0 & \frac{2}{5} & -\frac{1}{5} \\ -1 & 0 & 1 \\ 0 & -\frac{1}{5} & \frac{3}{5} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

It is more efficient to use Gaussian elimination. We show only the final triangular matrix; we used pivoting:

(1) Interchange the third and second rows before eliminating from the third row.

(2) Divide the third row by -5 before eliminating from the first row.

$$\begin{bmatrix} 1 & -1 & 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 0 & 1 & 0 & 1 & 0 \\ (0.333) & -1 & 1.667 & 1 & -0.333 & 0 \\ (0.333) & (0) & 1.667 & 0 & -0.333 & 1 \end{bmatrix}.$$

After doing the back-substitutions, we get

$$\boxed{\begin{bmatrix} 3 & 0 & 1 & 0 & 0.4 & -0.2 \\ (0.333) & -1 & 1.667 & -1 & 0 & 1 \\ (0.333) & (0) & 1.667 & 0 & -0.2 & 0.6 \end{bmatrix}.}$$

If we have the inverse of a matrix, we can use it to solve a set of equations, $Ax = b$, because multiplying by A^{-1} gives the answer:

$$\boxed{\begin{aligned} A^{-1}Ax &= A^{-1}b, \\ x &= A^{-1}b. \end{aligned}}$$

This would seem like a good way to solve equations, and there are many references to it. But this is not the best way to solve a system—getting the LU equivalent of A first and then using the L and U to solve $Ax = b$ requires only two back-substitutions and that requires exactly the same work as multiplying the vector b by a matrix. Finding A^{-1} means solving a system with three right-hand sides.

The real importance of the inverse is to develop theory and is essential to understanding many things in applied mathematics. For example, does every square matrix have an inverse? The answer is no, and we look now at when this is true.

Pathological Systems

When a real physical situation is modeled by a set of linear equations, we can anticipate that the set of equations will have a solution that matches the values of the quantities in the physical problem, at least as far as the equations truly do represent it.* Because of round-off errors, the solution vector that is calculated may imperfectly predict the physical quantity, but there is assurance that a solution exists, at least in principle. Consequently, it must always be theoretically possible to avoid divisions by zero when the set of equations has a solution.

An arbitrary set of equations may not have such a guaranteed solution, however. There are several such possible situations, which we term "pathological." In each case, there is *no unique solution* to the set of equations.

* There are certain problems for which values of interest are determined from a set of equations that do not have a unique solution; these are called *eigenvalue* problems and are discussed in Chapter 6.

First, here is an example of a matrix that has no inverse: What is the *LU* equivalent of

$$A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & 4 & -1 \\ -1 & -14 & 11 \end{bmatrix}?$$

MATLAB can find this:

```
EDU>> A = [1 -2 3; 2 4 -1;-1 -14 11]
A =
   1   -2    3
   2    4   -1
  -1  -14   11
EDU>> lu(A)
ans =
   2.0000     4.0000    -1.0000
  -0.5000   -12.0000    10.5000
   0.5000     0.3333     0
```

It is obvious that we cannot ordinarily solve a system $Ax = b$ that has this A matrix, for the zero in element $A(3, 3)$ cannot be used as a divisor in the back-substitution. That means that we cannot solve a system with the identity matrix as the right-hand sides. And that would have to be done to find A^{-1}. What does MATLAB say if we ask it to find the inverse?

```
EDU>> inv(A)
Warning: Matrix is singular to working precision.
ans =
  Inf    Inf    Inf
  Inf    Inf    Inf
  Inf    Inf    Inf
```

and we ask ourselves what the term *singular* means. Actually, the definition of a singular matrix is a matrix that does not have an inverse!

Are there other ways to see if a matrix is singular without trying to triangularize it? Yes, here are five other tests.

1. A singular matrix has a determinant of zero. This follows directly from the above result, where we saw that element $A(3, 3)$ was zero, and $\det(A) = (2)(-12)(0) = 0$.
2. The rank of the matrix is less than n, the number of rows. The rank is not as easy to find, but MATLAB can find it:

   ```
   EDU>> rank(A)
   ans =
     2
   ```

   The rank of a matrix is really determined by the next two properties of a singular matrix.

3. A singular matrix has rows that are linearly dependent vectors. A set of vectors is said to be *linearly dependent* if a weighted sum equals zero without using all weighting factors equal to zero. For matrix A above,

$$-3[1, -2, 3] + 2[2, 4, -1] + 1[-1, -14, 11] = [0, 0, 0].$$

4. A singular matrix has columns that are linearly dependent vectors. For matrix A,

$$-10[1, 2, -1]^T + 7[-2, 4, -14]^T + 8[3, -1, 11]^T = [0, 0, 0]^T.$$

5. A set of equations with this coefficient matrix has no unique solution.

Redundant and Inconsistent Systems

Even though a matrix is singular, it may still have a solution. Consider again the same singular matrix:

$$A = \begin{bmatrix} 1 & -2 & 3 \\ 2 & 4 & -1 \\ -1 & -14 & 11 \end{bmatrix}.$$

Suppose we solve the system $Ax = b$ where the right-hand side is $b = [5, 7, 1]^T$. MATLAB then gives

```
EDU>> Ab = [1 -2 3 5; 2 4 -1 7; -1 -14 11 1]
Ab =
    1      -2      3      5
    2       4     -1      7
   -1     -14     11      1
EDU>> lu(Ab)
ans =
    2.0000      4.0000     -1.0000     7.0000
   -0.5000    -12.0000     10.5000     4.5000
    0.5000      0.3333      0          0
```

and the back-substitution cannot be done. The display suggests that x_3 can have any value. Suppose we set it equal to 0. We can solve the first two equations with that substitution; that gives $[17/4, -3/8, 0]^T$. We get the same result from solving any other combination of two equations.

Suppose we set x_3 to 1 and repeat. This gives $[3, 1/2, 1]^T$, and this is another solution. In fact, any linear combination of these two is a solution! While that may not be a satisfactory answer, we must agree that we have found a solution, actually, an infinity of them. The reason for this is that the system is redundant: The third equation, given by the third row, is just a linear combination of the first two:

$$-3[1, -2, 3, 5] + 2[2, 4, -1, 7] = -1[-1, -14, 11, 1].$$

Of course, this means that any one equation is a linear combination of the other two. What we have here is not truly three linear equations but only two independent ones. The system is called *redundant*.

What if the right-hand vector is different, say, $[5, 7, 2]^T$? Solving with this vector, we find this final array:

$$\begin{array}{cccc} 2.0000 & 4.0000 & -1.0000 & 7.0000 \\ (-0.5000) & -12.0000 & 10.5000 & 5.5000 \\ (0.5000) & (0.3333) & 0 & -0.3333 \end{array}$$

We now say, for this system, that it is *inconsistent;* there is no solution that satisfies the equations. In either case, there is no unique solution to a system with a singular coefficient matrix.

Here is a comparison of singular and nonsingular matrices:

For Singular Matrix A:	For Nonsingular Matrix A:
It has no inverse, A^{-1}	It has an inverse, A^{-1} exists
Its determinant is zero	The determinant is nonzero
There is no unique solution to the system $Ax = b$	There is a unique solution to the system $Ax = b$
Gaussian elimination cannot avoid a zero on the diagonal	Gaussian elimination does not encounter a zero on the diagonal
The rank is less than n	The rank equals n
Rows are linearly dependent	Rows are linearly independent
Columns are linearly dependent	Columns are linearly independent

2.4 Ill-Conditioned Systems

We have seen that a system whose coefficient matrix is singular has no unique solution. What if the matrix is "almost singular"? Here is an example:

$$A = \begin{bmatrix} 3.02 & -1.05 & 2.53 \\ 4.33 & 0.56 & -1.78 \\ -0.83 & -0.54 & 1.47 \end{bmatrix}.$$

The LU equivalent has a very small element in position (3, 3):

$$LU = \begin{bmatrix} 4.33 & 0.56 & -1.78 \\ 0.6975 & -1.4406 & 3.7715 \\ -0.1917 & 0.3003 & -0.0039 \end{bmatrix}.$$

Let's look at the inverse:

$$\text{inv}(A) = \begin{bmatrix} 5.6611 & -7.2732 & -18.5503 \\ 200.5046 & -268.2570 & -669.9143 \\ 76.8511 & -102.6500 & -255.8846 \end{bmatrix},$$

and we see that this has elements very large in comparison to A.

Both of these results suggest that matrix A is nonsingular but is "almost singular." Suppose we solve the system $Ax = b$, with b equal to $[-1.61, 7.23, -3.38]^T$. It is clear, if you do the math, that the solution is $x = 1.0000, 2.0000, -1.0000$.

Now suppose that we make a small change in just the first element of the b-vector: $b = [-1.60, 7.23, -3.38]$. Now solve again; we get $x = [1.0566, 4.0051, -0.2315]$. What a difference! Let us try another small change in the b-vector: $b = [-1.61, 7.22, -3.38]^T$. The solution now is $x = [1.07271, 4.6826, 0.0265]$ which also differs much from our first answer.

A system whose coefficient matrix is nearly singular is called *ill-conditioned.* When a system is ill-conditioned, the solution is very sensitive to changes in the right-hand vector. It is also sensitive to small changes in the coefficients. If $A(1, 1)$ is changed from 3.02 to 3.00 and we solve the system again with the original b-vector, we now find a large change in the solution: $x = [1.1277, 6.5221, 0.7333]^T$. This means that it is also very sensitive to round-off error.

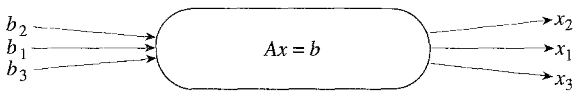
This phenomenon shows up even more pointedly in large systems. But even this system of only two equations shows the effect of near singularity:

$$\begin{bmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2.00 \\ 2.00 \end{bmatrix}.$$

The solution is clearly seen to be $x = 1.00, y = 1.00$.

However, if we make a small change to the b-vector, to $[2.02, 1.98]^T$, the solution now is $x = 2, y = 0$. If we had another slightly different b-vector: $[1.98, 2.02]^T$, we would have $x = 0, y = 2$!

It is helpful to think of the system, $Ax = b$, as a *linear system solver machine.* In this view, we have *inputs* to the machine, the b-vector, and *outputs,* the x-values. For an ill-conditioned system, small changes in the input make large changes in the output.



Even though the three inputs are "close together"—$b_1 = (2, 2)^T$, $b_2 = (2.02, 1.98)^T$, and $b_3 = (1.98, 2.02)^T$—we get very "distant" outputs—$x_1 = (1, 1)^T$, $x_2 = (2, 0)^T$, $x_3 = (0, 2)^T$. This modest example shows the basic idea of an ill-conditioned system: *For small changes in the input, we get large changes in the output.*

In some situations, one can combat ill-conditioning by transforming the problem into an equivalent set of equations that are not ill-conditioned. The efficiency of this scheme is related to the relative amount of computation required for the transformation, compared to the cost of doing the calculations in higher precision.

An interesting phenomenon of an ill-conditioned system is that we cannot test for the accuracy of the computed solution merely by substituting it into the equations to see whether the right-hand sides are reproduced. Consider again the ill-conditioned example we have previously examined:

$$A = \begin{bmatrix} 3.02 & -1.05 & 2.53 \\ 4.33 & 0.56 & -1.78 \\ -0.83 & -0.54 & 1.47 \end{bmatrix}, \qquad b = \begin{bmatrix} -1.61 \\ 7.23 \\ -3.38 \end{bmatrix}.$$

If we compute the vector Ax, using the exact solution $x = (1, 2, -1)^T$, we of course get

$$Ax = (-1.61, 7.23, -3.38)^T = b.$$

However, if we substitute a clearly erroneous vector

$$\bar{x} = (0.880, -2.34, -2.66)^T,$$

we get $A\bar{x}$ $(-1.6152, 7.2348, -3.3770)^T$, which is very close to b.

Effect of Precision

We have mentioned that it is difficult to get an accurate solution when a system is ill-conditioned and have demonstrated that small changes in either the coefficients or the right-hand side make large changes in the solution. The solution is also dependent on the accuracy of the arithmetic computations. We use here the computer algebra system, Maple, to show this. We begin by invoking the linalg "package" that enables many operations:

```
with (linalg)
```

Now we define the matrix:

```
>A: =matrix (3,4,[3.02,-1.05,2.53,-1.61,4.33,0.56,-1.78,7.23,
-0.83,-0.54,1.47,-3.38]);
```

$$\text{A:} = \begin{bmatrix} 3.02 & -1.05 & 2.53 & -1.61 \\ 4.33 & .56 & -1.78 & 7.23 \\ -.83 & -.54 & 1.47 & -3.38 \end{bmatrix}.$$

We already have seen that this system is ill-conditioned. What does Maple say the solution is?

```
>rref(A);
```

$$\begin{bmatrix} 1 & 0 & 0 & 1.000000037 \\ 0 & 1 & 0 & 2.000001339 \\ 0 & 0 & 1 & -.9999994882 \end{bmatrix},$$

which is pretty close to the exact solution, $x = [1, 2, -1]$. Maple used its default precision of ten digits in getting this answer. If we change the precision to 20, we get a more accurate solution but it is still not exact. What if we change it to only three digits?

```
Digits: =3; rref(A);
  Digits: =3
```

$$\begin{bmatrix} 1 & 0 & -.073 & 0 \\ 0 & 1 & -2.62 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and Maple tells us that the coefficient matrix is singular at that precision. If we change to four digits:

```
Digits: = 4; rref(A);
  Digits: = 4
```

$$\begin{bmatrix} 1 & 0 & 0 & .9824 \\ 0 & 1 & 0 & 1.346 \\ 0 & 0 & 1 & -1.250 \end{bmatrix}$$

we get a poor approximation to $x = [1, 2, -1]$, the exact solution.

The 'rref' command can be used in MATLAB also. It rather obviously uses the Gauss–Jordan method.

Condition Numbers and Norms

The degree of ill-conditioning of a matrix is measured by its *condition number.* This is defined in terms of its *norms,* a measure of the magnitude of the matrix. We discuss norms before discussing condition numbers.

The magnitude of a single number is just its distance from zero: $|-4.2| = 4.2$. But a matrix is not a single number; its norm is different. For any norm, these properties are essential:

1. The norm must always have a value greater than or equal to zero, and must be zero only when the matrix is the zero matrix (one with all elements equal to zero).
2. The norm must be multiplied by k if the matrix is multiplied by the scalar k.
3. The norm of the sum of two matrices must not exceed the sum of the norms.
4. The norm of the product of two matrices must not exceed the product of the norms.

More formally, we can state these conditions, using $\|A\|$ to represent the *norm of matrix A:*

1. $\|A\| \geq 0$ and $\|A\| = 0$ if and only if $A = 0$.
2. $\|kA\| = |k|\,\|A\|$.
3. $\|A + B\| \leq \|A\| + \|B\|$.
4. $\|AB\| \leq \|A\|\,\|B\|$.

(2.14)

The third relationship is called the *triangle inequality.* The fourth is important when we deal with the product of matrices.

For the special kind of matrices that we call vectors, our past experience can help us. For vectors in two- or three-space, the length satisfies all four requirements and is a good value to use for the norm of a vector. This norm is called the *Euclidean norm,* and is computed by $\sqrt{x_1^2 + x_2^2 + x_3^2}$.

We compute the Euclidean norm of vectors with more than three components by generalizing:

$$\|x\|_e = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}.$$

This is not the only way to compute a vector norm, however. The sum of the absolute values of the x_i can be used as a norm; the maximum value of the magnitudes of the x_i will also serve. These three norms can be interrelated by defining the p-norm as.

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

From this its is readily seen that

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| = \text{sum of magnitudes};$$

$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2} = \text{Euclidean norm};$$

$$\|x\|_\infty = \max_{1\le i\le n} |x_i| = \text{maximum-magnitude norm}.$$

Which of these vector norms is best to use may depend on the problem. In most cases, satisfatory results are obtained with any of these measures of the "size" of a vector.

EXAMPLE 2.3 Compute the 1-, 2-, and ∞-norms of the vector x, if $x = (1.25, 0.02, -5.15, 0)$.

$$\|x\|_1 = |1.25| + |0.02| + |-5.15| + |0| = 6.42.$$

$$\|x\|_2 = [(1.25)^2 + (0.02)^2 + (-5.15)^2 + (0)^2]^{1/2} = 5.2996.$$

$$\|x\|_\infty = |-5.15| = 5.15.$$

■

Matrix Norms

The norms of a matrix are similar to the norms of a vector. Two norms that are closely related are

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| = \text{maximum column sum};$$

$$\|A\|_\infty = \max_{1\le j\le n} \sum_{j=1}^{n} |a_{ij}| = \text{maximum row sum}.$$

The matrix norm $\|A\|_2$ that corresponds to the 2-norm of a vector is not readily computed. It is defined in terms of the eigenvalues of the matrix $A^T * A$. Suppose r is the largest eigenvalue of $A^T * A$. Then $\|A\|_2 = r^{1/2}$, the square root of r. This is called the *spectral norm* of A, and $\|A\|_2$ is always less than (or equal to) $\|A\|_1$ and $\|A\|_\infty$.

For an $m \times n$ matrix, the Frobenius norm is defined as

$$\|A\|_f = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2 \right)^{1/2}.$$

EXAMPLE 2.4 Compute the Frobenius norms of A, B, and C, and the ∞-norms, given that

$$A = \begin{bmatrix} 5 & 9 \\ -2 & 1 \end{bmatrix}; \quad B = \begin{bmatrix} 0.1 & 0 \\ 0.2 & 0.1 \end{bmatrix}; \quad \text{and} \quad C = \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0 \end{bmatrix}.$$

$$\|A\|_f = \sqrt{25 + 81 + 4 + 1} = \sqrt{111} = 10.54; \qquad \|A\|_\infty = 14.0;$$
$$\|B\|_f = \sqrt{0.01 + 0 + 0.04 + 0.01} = \sqrt{0.06} = 0.2449; \qquad \|B\|_\infty = 0.3;$$
$$\|C\|_f = \sqrt{0.04 + 0.01 + 0.01 + 0} = \sqrt{0.06} = 0.2449; \qquad \|C\|_\infty = 0.3.$$

The results of our examples look quite reasonable; certainly A is "larger" than B or C. Although $B \neq C$, both are equally "small." The Frobenius norm is a good measure of the magnitude of a matrix.

We see then that there are a number of ways that the norm of a matrix can be expressed. Which way is preferred? There are certainly differences in their cost; for example, some will require more extensive arithmetic than others. The spectral norm is usually the most "expensive." Which norm is best? The answer to this question depends in part on the use for the norm. In most instances, we want the norm that puts the smallest upper bound on the magnitude of the matrix. In this sense, the spectral norm is usually the "best." We observe, in the next example, that not all the norms give the same value for the magnitude of a matrix.

MATLAB can compute all of the norms of a matrix:

```
A =
     5  -5  -7
    -4   2  -4
    -7  -4   5
```

```
EDU>> norm(A,'fro')
ans =
   15
EDU>> norm(A,inf)
ans =
   17
EDU>> norm(A,1)
ans =
   16
EDU>> norm(A)
ans =
   12.0301
EDU>> norm(A,2)
ans =
   12.0301
```

We observe that the 2-norm, the spectral norm, is the norm we get if we just ask for the "norm." The smallest norm of the matrix is the spectral norm, it is the "tightest" measure.

Errors in the Solution and Norms

When we solve a system of equations, we hope that the result has little error but, as we have seen, that is not always true. If the coefficient matrix is ill-conditioned, we cannot check the accuracy by just substituting our answer into the equations, but we can use norms to see how great the error is.

Let $\bar{x}$ be the computed solution, an approximation to the true solution. Define the *residual, r,* as $r = b - A\bar{x}$, the difference between the b-vector and what we get when the approximate $\bar{x}$ is substituted into the equations. Let e be the error in $\bar{x}$ and x be the true solution to the system (which we don't know), $e = x - \bar{x}$. Because $Ax = b$, we have

$$r = b - A\bar{x} = Ax - A\bar{x} = A(x - \bar{x}) = Ae.$$

Hence,

$$e = A^{-1}r.$$

Taking norms and recalling Eq. (2.14), line 4, for a product, we write

$$\|e\| \le \|A^{-1}\|\,\|r\|. \tag{2.15}$$

From $r = Ae$, we also have $\|r\| \le \|A\|\,\|e\|$, which combines with Eq. (2.15) to give

$$\frac{\|r\|}{\|A\|} \le \|e\| \le \|A^{-1}\|\,\|r\|. \tag{2.16}$$

Applying the same reasoning to $Ax = b$ and $x = A^{-1}b$, we get

$$\frac{\|b\|}{\|A\|} \le \|x\| \le \|A^{-1}\|\,\|b\|. \tag{2.17}$$

Taking Eqs. (2.16) and (2.17) together, we reach a most important relationship:

$$\boxed{\frac{1}{\|A\|\,\|A^{-1}\|}\frac{\|r\|}{\|b\|} \le \frac{\|e\|}{\|x\|} \le \|A\|\,\|A^{-1}\|\frac{\|r\|}{\|b\|}.}$$

Condition Number of a Matrix

The product of the norm of A and the norm of its inverse is called the *condition number* of matrix A and is the best measure of ill-conditioning. A small number means good-conditioning, a large one means ill-conditioning. So, we usually write the previous equation as

$$\boxed{\frac{1}{(\text{Condition no.})}\frac{\|r\|}{\|b\|} \le \frac{\|e\|}{\|x\|} \le (\text{Condition no.})\frac{\|r\|}{\|b\|}. \qquad (2.18)}$$

Equation (2.18) shows that the relative error in the computed solution vector $\bar{x}$ can be as great as the relative residual multiplied by the condition number. Of course it can also be as small as the relative residual divided by the condition number. Therefore, when the condition number is large, the residual gives little information about the accuracy of $\bar{x}$. Conversely, when the condition number is near unity, the relative residual is a good measure of the relative error of $\bar{x}$.

When we solve a linear system, we are normally doing so to determine values for a physical system for which the set of equations is a model. We use the measured values of the parameters of the physical system to evaluate the coefficients of the equations, so we expect these coefficients to be known only as precisely as the measurements. When these are in error, the solution of the equations will reflect these errors. We have already seen that an ill-conditioned system is extremely sensitive to small changes in the coefficients. The condition number lets us relate the change in the solution vector to such errors in the coefficients of the set of equations $Ax = b$.

Assume that the errors in measuring the parameters cause errors in the coefficients of A so that the actual set of equations being solved is $(A + E)\bar{x} = b$, where $\bar{x}$ represents the solution of the perturbed system and A represents the true (but unknown) coefficients. We let $\bar{A} = A + E$ represent the perturbed coefficient matrix. We desire to know how large $x - \bar{x}$ is.

Using $Ax = b$ and $\bar{A}\bar{x} = b$, we can write

$$\begin{aligned} x &= A^{-1}b = A^{-1}(\bar{A}\bar{x}) = A^{-1}(A + \bar{A} - A)\bar{x} \\ &= [I + A^{-1}(\bar{A} - A)]\bar{x} \\ &= \bar{x} + A^{-1}(\bar{A} - A)\bar{x}. \end{aligned}$$

Because $\bar{A} - A = E$, we have

$$x - \bar{x} = A^{-1}E\bar{x}.$$

Taking norms, we get

$$\boxed{\|x - \bar{x}\| \leq \|A^{-1}\| \|E\| \|\bar{x}\| = \|A^{-1}\| \|A\| \frac{\|E\|}{\|A\|} \|\bar{x}\|,}$$

so that

$$\boxed{\frac{\|x - \bar{x}\|}{\|\bar{x}\|} \leq (\text{Condition no.}) \frac{\|E\|}{\|A\|}.}$$

This says that the error of the solution relative to the norm of the computed solution can be as large as the relative error in the coefficients of A multiplied by the condition number. The net effect is that, if the coefficients of A are known to only four-digit precision and the condition number is 1000, the computed vector x may have only one digit of accuracy.

Iterative Improvement

When the solution to the system $Ax = b$ has been computed, and, because of round-off error, we obtain the approximate solution vector $\bar{x}$, it is possible to apply iterative improvement to correct $\bar{x}$ so that it more closely agrees with x. Define $e = x - \bar{x}$. Define $r = b - A\bar{x}$.

$$Ae = r. \tag{2.19}$$

If we could solve this equation for e, we could apply this as a correction to $\bar{x}$. Furthermore, if $\|e\|/\|\bar{x}\|$ is small, it means that $\bar{x}$ should be close to x. In fact, if the value of $\|e\|/\|\bar{x}\|$ is 10^{-p}, we know that $\bar{x}$ is probably correct to p digits.

The process of iterative improvement is based on solving Eq. (2.19). Of course this is also subject to the same round-off error as the original solution of the system for $\bar{x}$, so we actually get $\bar{e}$, an approximation to the true error vector. Even so, unless the system is so ill-conditioned that $\bar{e}$ is not a reasonable approximation to e, we will get an improved estimate of x from $\bar{x} + \bar{e}$. One special caution is important to observe: The computation of the residual vector r must be as precise as possible. One always uses double-precision arithmetic; otherwise, iterative improvement will not be successful. An example will make this clear.

We are given

$$A = \begin{bmatrix} 4.23 & -1.06 & 2.11 \\ -2.53 & 6.77 & 0.98 \\ 1.85 & -2.11 & -2.32 \end{bmatrix}, \qquad b = \begin{bmatrix} 5.28 \\ 5.22 \\ -2.58 \end{bmatrix},$$

whose true solution is

$$x = \begin{bmatrix} 1.000 \\ 1.000 \\ 1.000 \end{bmatrix}.$$

If inadequate precision is used, we might get this approximate solution vector: $\bar{x} = (0.991, 0.997, 1.000)^T$. Using double precision, we compute $A\bar{x}$, storing this product in a register that holds six digits, then we get the residual.

$$A\bar{x} = \begin{bmatrix} 5.24511 \\ 5.22246 \\ -2.59032 \end{bmatrix}, \quad r = \begin{bmatrix} 0.0349 \\ -0.00246 \\ 0.0103 \end{bmatrix}.$$

We now solve $A\bar{e} = r$, again using three-digit precision, and get

$$\bar{e} = \begin{bmatrix} 0.00822 \\ 0.00300 \\ -0.00000757 \end{bmatrix}.$$

Finally, correcting $\bar{x}$ with $\bar{x} + \bar{e}$ gives almost exactly the correct solution:

$$\bar{x} + \bar{e} = \begin{bmatrix} 0.999 \\ 1.000 \\ 1.000 \end{bmatrix}.$$

In the general case, the iterations are repeated until the corrections are negligible. Because we want to make the solution of Eq. (2.19) as economical as possible, we should use an *LU* method to solve the original system and apply the *LU* to Eq. (2.19).

Pivoting and Precision

We have previously said that pivoting reduces the errors due to round off. We examine this further with a small system, one with only two equations:

$$\varepsilon x + By = C,$$
$$Dx + Ey = F,$$

with ε a very small number. If this is solved without pivoting and the first column is reduced ($-D/\varepsilon$ is the multiplier), we get

$$\varepsilon x + By = C,$$
$$(E - DB/\varepsilon)y = F - CD\backslash\varepsilon.$$

and, solving for y, we see that

$$y = \frac{F - CD/\varepsilon}{E - DB/\varepsilon} \approx \frac{CD}{DB} = \frac{C}{B} \text{ if } \varepsilon \text{ is very small.}$$

Substituting this for y in the first equation, we find

$$x = \frac{C - B(C/B)}{\varepsilon} = \frac{C - C}{\varepsilon} = 0!$$

showing that $x \approx 0$ for any values of C and F if ε is small enough. Now, suppose that $F = D + E$ and $C = \varepsilon + B$ and we do pivot by interchanging the equations:

$$Dx + Ey = F = D + E,$$
$$\varepsilon x + By = C = \varepsilon + B.$$

(Obviously, the correct solution must be $x = 1$, $y = 1$.) Now, reducing the first column ($-\varepsilon/D$ is the multiplier) we have:

$$Dx + Ey = D + E$$
$$(B - (\varepsilon/D)E)y = \varepsilon + B - (\varepsilon/D)(D + E) = \frac{\varepsilon D + BD - \varepsilon D - \varepsilon E}{D} = \frac{BD - \varepsilon E}{D},$$

so that

$$y = \frac{(BD - \varepsilon E)/D}{(BD - \varepsilon E)/D} = 1.$$

We get x by substituting $y = 1$ into the first equation, so that

$$x = \frac{D + E - E(1)}{D} = 1,$$

which demonstrates how pivoting may be very necessary.

Here is a numerical example of the same thing. The augmented matrix is

$$\begin{bmatrix} 0.02 & 10 & 10.02 \\ 10 & 10 & 20 \end{bmatrix},$$

which must yield $x = 1$, $y = 1$. If precision is infinite, pivoting is not required. Let's reduce in the first column without pivoting ($-10/0.02 = -500$ is the multiplier):

$$\begin{bmatrix} 0.02 & 10 & 10.02 \\ 0 & -4990 & -4490 \end{bmatrix},$$

which gives $y = (-4990)/(-4990) = 1$.

Now, substituting $y = 1$ into the first equation,

$$x = \frac{10.02 - 10}{0.02}.$$

The result for x is 1 if the numerator equals 0.02. But suppose we have only three digits of precision—that means the numerator does not equal 0.02 but is 0.00 and x is zero!

2.5 Iterative Methods

Gaussian elimination and its variants are called *direct methods*. An entirely different way to solve many systems is through *iteration*. In this, we start with an initial estimate of the solution vector and proceed to refine this estimate. There are times when this is preferred over a direct method. This is especially true when the coefficient matrix is sparse.

The two methods for solving $Ax = b$ that we shall discuss in this section are the *Jacobi method* and the *Gauss–Seidel method*. These methods not only can solve a system of equations but they are also the basis for other accelerated methods that we shall introduce in later chapters of this book. When the system of equations can be ordered so that each diagonal entry of the coefficient matrix is larger in magnitude than the sum of the magnitudes of the other coefficients in that row—such a system is called *diagonally dominant*—the iteration will converge for any starting values. Formally, we say that an $n \times n$ matrix A is diagonally dominant if and only if for each $i = 1, 2, \ldots, n$,

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|, \qquad i = 1, 2, \ldots, n.$$

Although this may seem like a very restrictive condition, it turns out that there are very many applied problems that have this property (steady-state and transient heat transfer are two). Our approach is illustrated with the following simple example of a system.

$$\begin{aligned} 6x_1 - 2x_2 + x_3 &= 11, \\ x_1 + 2x_2 - 5x_3 &= -1, \\ -2x_1 + 7x_2 + 2x_3 &= 5. \end{aligned}$$

The solution is $x_1 = 2, x_2 = 1, x_3 = 1$. However, before we begin our iterative scheme we must first reorder the equations so that the coefficient matrix is diagonally dominant.

$$\begin{aligned} 6x_1 - 2x_2 + x_3 &= 11, \\ -2x_1 + 7x_2 + 2x_3 &= 5, \\ x_1 + 2x_2 - 5x_3 &= -1. \end{aligned} \tag{2.20}$$

The iterative methods depend on the rearrangement of the equations in this manner:

$$x_i = \frac{b_i}{a_{i,i}} - \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j, \qquad i = 1, 2, \ldots, n.$$

Each equation is now solved for the variables in succession:

$$\begin{aligned} x_1 &= 1.8333 + 0.3333x_2 - 0.1667x_3, \\ x_2 &= 0.7143 + 0.2857x_1 - 0.2857x_3, \\ x_3 &= 0.2000 + 0.2000x_1 + 0.4000x_2. \end{aligned}$$

We begin with some initial approximation to the value of the variables. (Each component might be taken equal to zero if no better initial estimates are at hand.) Substituting these approximations into the right-hand sides of the set of equations generates new approximations that, we hope, are closer to the true value. The new values are substituted in the right-hand sides to generate a second approximation, and the process is repeated until successive values of each of the variables are sufficiently alike. We indicate the iterative process on Eq. (2.20), as follows, by putting superscripts on variables to indicate successive iterates. Thus our set of equations becomes

$$\begin{aligned} x_1^{(n+1)} &= 1.8333 + 0.3333x_2^{(n)} - 0.1667x_3^{(n)}, \\ x_2^{(n+1)} &= 0.7143 + 0.2857x_1^{(n)} - 0.2857x_3^{(n)}, \\ x_3^{(n+1)} &= 0.2000 + 0.2000x_1^{(n)} + 0.4000x_2^{(n)}. \end{aligned} \tag{2.21}$$

Starting with an initial vector of $x^{(0)} = (0, 0, 0)$, we get

Successive estimates of solution (Jacobi method)

	First	Second	Third	Fourth	Fifth	Sixth	...	Ninth
x_1	0	1.833	2.038	2.085	2.004	1.994	···	2.000
x_2	0	0.714	1.181	1.053	1.001	0.990	···	1.000
x_3	0	0.200	0.852	1.080	1.038	1.001	···	1.000

Note that this method is exactly the same as the method of fixed-point iteration for a single equation that was discussed in Chapter 1, but it is now applied to a set of equations; we see this if we write Eq. (2.21) in the form of

$$x^{(n+1)} = G(x^{(n)}) = b' - Bx^{(n)},$$

which is identical to $x_{n+1} = g(x_n)$ as used in Chapter 1.

In the present context, of course, $x^{(n)}$ and $x^{(n+1)}$ refer to the nth and $(n + 1)$st iterates of a vector rather than a simple variable, and g is a linear transformation rather than a nonlinear function. For the preceding example, we restate Eq. (2.20) in matrix form:

$$Ax = b, \quad \begin{bmatrix} 6 & -2 & 1 \\ -2 & 7 & 2 \\ 1 & 2 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 5 \\ -1 \end{bmatrix}. \tag{2.22}$$

Now, let $A = L + D + U$, where

$$L = \begin{bmatrix} 0 & 0 & 0 \\ -2 & 0 & 0 \\ 1 & 2 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 6 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & -5 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & -2 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}.$$

Then Eq. (2.22) can be rewritten as

$$\begin{aligned} Ax &= (L + D + U)x = b, \quad \text{or} \\ Dx &= -(L + U)x + b, \quad \text{which gives} \\ x &= -D^{-1}(L + U)x + D^{-1}b. \end{aligned}$$

From this we have, identifying x on the left as the new iterate,

$$x^{(n+1)} = -D^{-1}(L + U)x^{(n)} + D^{-1}b. \tag{2.23}$$

In Eqs. (2.21) we see that

$$b' = D^{-1}b = \begin{bmatrix} 1.8333 \\ 0.7143 \\ 0.2000 \end{bmatrix},$$

$$B = D^{-1}(L + U) = \begin{bmatrix} 0 & -0.3333 & 0.1667 \\ -0.2857 & 0 & 0.2857 \\ -0.2000 & -0.4000 & 0 \end{bmatrix}.$$

The procedure we have just described is known as the *Jacobi method,* also called "the method of simultaneous displacements" because each of the equations is simultaneously changed by using the most recent set of x-values.

We can write the algorithm for the Jacobi iterative method as follows:

Algorithm for Jacobi Iteration

We assume that the system $Ax = b$ has been rearranged so that the matrix A is diagonally dominant. That is, for each row of A:

$$|a_{i,i}| > \sum_{\substack{j=i \\ j \neq i}}^{n} |a_{i,j}|, \qquad i = 1, 2, \dots, n.$$

This is a sufficient condition for convergence both for this method and for the one that we discuss next. We begin with an initial approximation to the solution vector, which we store in the vector: *old_x.*

```
For $i = 1$ To $n$
  $b[i] = b[i]/a[i, i]$
  $new\_x[i] = old\_x[i]$
  $a[i, j] = a[i, j]/a[i, i]$; $j = 1 \dots n$ and $i <> j$
End For $i$

Repeat

  For $i = 1$ To $n$
    $old\_x[i] = new\_x[i]$
    $new\_x[i] = b[i]$
  End For $i$

  For $i = 1$ To $n$
    For $j = 1$ To $n$
```

```
            If ($j <> i$) Then
              $new\_x[i] = new\_x[i] - a[i,j] * old\_x[j]$
            End For $j$
        End For $i$
    Until $new\_x$ and $old\_x$ converge to each other.
```

Gauss – Seidel Iteration

Observe that we never use the values *new-x* in the algorithm for Jacobi iteration until we have found all of its components. Even though we have *new_x* [1] available, we do not use it to compute *new_x* [2] even though in nearly all cases the new values are better than the old and ought to be used instead. When this is done, the procedure known as *Gauss–Seidel iteration* results.

We begin exactly as with the Jacobi method by rearranging the equations, solving each equation for the variable whose coefficient is dominant in terms of the others. We proceed to improve each x-value in turn, using always the most recent approximations of the other variables. The rate of convergence is more rapid than for the Jacobi method, as shown by reworking the previous example [Eq. (2.20)].

Successive estimates of solution (Gauss–Seidel method)

	First	Second	Third	Fourth	Fifth	Sixth
x_1	0	1.833	2.069	1.998	1.999	2.000
x_2	0	1.238	1.002	0.995	1.000	1.000
x_3	0	1.062	1.015	0.998	1.000	1.000

These values were computed by using this iterative scheme:

$$\begin{aligned} x_1^{(n+1)} &= 1.8333 + 0.3333x_2^{(n)} - 0.1667x_3^{(n)}, \\ x_2^{(n+1)} &= 0.7143 + 0.2857x_1^{(n+1)} - 0.2857x_3^{(n)}, \\ x_3^{(n+1)} &= 0.2000 + 0.2000x_1^{(n+1)} + 0.4000x_2^{(n+1)}, \end{aligned}$$

beginning with $x^{(1)} = (0, 0, 0)^T$.

The algorithm for the Gauss–Seidel iteration is as follows:

Algorithm for Gauss–Seidel Iteration

We assume as we did in the previous algorithm that the system $Ax = b$ has been rearranged so that the coefficient matrix, A, is diagonally dominant. As before, we begin with an initial approximation to the solution vector, which we store in the vector: x.

```
For $i = 1$ To $n$
  $b[i] = b[i]/a[i, i]$
  $a[i, j] = a[i, j]/a[i, i]; j = 1 \ldots n$ and $i <> j$
End for $i$;

While Not (yet convergent) Do
   For $i = 1$ To $n$
      $x[i] = b[i]$;
      For $j = 1$ To $n$
         If ($j <> i$) Then
            $x[i] = x[i] - a[i, j] * x[j]$
      End For $j$
   End For $i$
End While
```

The matrix formulation for the Gauss–Seidel method is almost the same as the one given in Eq. (2.23). For Gauss–Seidel, $Ax = b$ can be rewritten as

$$(L + D)x = -Ux + b, \tag{2.24}$$

and from this we get

$$x^{(n+1)} = -(L + D)^{-1}Ux^{(n)} + (L + D)^{-1}b. \tag{2.25}$$

The usefulness of this matrix notation will become apparent in Chapter 6 where the eigenvalues of matrices $D^{-1}(L + U)$ of Eq. (2.23) and $(L + D)^{-1}\ U$ of Eq. (2.25) will be studied. The eigenvalues of the two matrices indicate how fast the iterations will converge. We emphasize, however, that without diagonal dominance, neither Jacobi nor Gauss–Seidel is sure to converge. (Some authors use the term *row diagonal dominance* for our term *diagonal dominance*. Their term is perhaps more accurate.)

There are some instances of the system $Ax = b$ where the coefficient matrix does not have (row) diagonal dominance but still both Jacobi and Gauss–Seidel methods do converge. It can be shown that, if the coefficient matrix, A, is symmetric and positive definite,* the Gauss–Seidel method will converge from any starting vector. In another class of problems, where matrix A has diagonal elements that are all positive and off-diagonal elements that are all negative, both Jacobi and Gauss–Seidel methods will either converge or diverge. When both methods converge, the Gauss–Seidel method converges faster. Datta (1995) discusses this and gives examples.

For a general coefficient matrix, there is little that can be said. In fact, there are examples where Jacobi converges and Gauss–Seidel diverges from the same starting vector! Still, returning to the focus of this section, we can say that, given row diagonal dominance in the coefficient matrix, the Gauss–Seidel method is often the better choice. Having said

* Matrix A is positive definite if $x * Ax > 0$ for all nonzero vectors x.

that, we may still prefer the Jacobi method if we are running the program on parallel processors because all n equations can be solved simulaneously at each iteration.

Accelerating Convergence

Convergence in the Gauss–Seidel method can be speeded if we do what is called *overrelaxing.* The term comes from an old hand method where a set of "residuals" (the right-hand-side values for a rearrangement of the equations when the unknowns were given certain values) were "relaxed" to zero. Overelaxation will be encountered again in Chapter 8, where we solve partial-differential equations.

The standard relationship for Gauss–Seidel iteration for the set of equations $Ax = b$, for variable x_i, can be written

$$\boxed{x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \qquad \textbf{(2.26)}}$$

where the superscript $(k + 1)$ indicates that this is the $(k + 1)$st iterate. On the right side we use the most recent estimates of the x_j, which will be either $x_j^{(k)}$ or $x_j^{(k+1)}$.

An algebraically equivalent form for Eq. (2.26) is

$$\boxed{x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(k)}\right),}$$

because $x_i^{(k)}$ is both added to and subtracted from the right side. Overrelaxation can be applied to Gauss–Seidel if we will add to $x_i^{(k)}$ some multiple of the second term. It can be shown that this multiple should never be more than 2 in magnitude (to avoid divergence), and the optimal overrelaxation factor lies between 1.0 and 2.0. Our iteration equations take this form, where w is the *overrelaxation factor:*

$$\boxed{x_i^{(k+1)} = x_i^{(k)} + \frac{w}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=1}^{n} a_{ij}x_j^{(k)}\right).}$$

Table 2.1 shows how the convergence rate is influenced by the value of w for the system

$$\begin{bmatrix} -4 & 1 & 1 & 1 \\ 1 & -4 & 1 & 1 \\ 1 & 1 & -4 & 1 \\ 1 & 1 & 1 & -4 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix},$$

Table 2.1 Acceleration of convergence of Gauss–Seidel iteration

***w*, the overrelaxation factor**	**Number of iterations to reach error $<1 \times 10^{-5}$**	
1.0	24	
1.1	18	
1.2	13	
1.3	11	←Minimum
1.4	14	of iterations
1.5	18	
1.6	24	
1.7	35	
1.8	55	
1.9	100+	

starting with an initial estimate of $x = 0$. The exact solution is

$$x_1 = -1, \quad x_2 = -1, \quad x_3 = -1, \quad x_4 = -1.$$

Sparse Matrices and Banded Matrices

It has already been said that many applied problems are solved with systems whose coefficient matrix is sparse—only a fraction of the elements are nonzero. In Chapters 6 and 8 you will see several instances. The tridiagonal system of Section 2.2 is the prime example. In other applications the coefficient matrix may be sparse and have elements situated in selected positions. A *banded matrix* is one where the nonzero elements lie on diagonals parallel to the main diagonal. A tridiagonal matrix is obviously banded.

Some sparse matrices are not as compact as a tridiagonal one. It often happens that the nonzero coefficients lie on diagonals but some or all of these bands are not adjacent to the main diagonal. Algorithms similar to the one for a tridiagonal matrix can be developed. However, one usually finds that the nonzero elements between the bands and the main diagonal do not stay zero and more arithmetic operations are needed to get a solution or to find an LU equivalent to the coefficient matrix.

Fortunately, most sparse systems have a main diagonal that is dominant so it is easy to set up the rearranged equations that can be solved quickly by iteration. And these iterations can be speeded up by overrelaxation. Here is an example of a small system:

$$\begin{bmatrix} 3 & 1 & 0 & -1 & 0 & 0 & 2.05 \\ 1 & 4 & 2 & 0 & 2 & 0 & 3.33 \\ 0 & 2 & 4 & 1 & 0 & 3 & -6.21 \\ 2 & 0 & -1 & 3 & 3 & 0 & 5.25 \\ 0 & 3 & 0 & 1 & 5 & 2 & 8.92 \\ 0 & 0 & 1 & 0 & -1 & 2 & 10.87 \end{bmatrix}$$

whose solution is

$$x = [-1.7040, 17.9737, -19.1286, 10.8118, -14.3019, 7.8483]^T.$$

which we found by Gaussian elimination. It would be good practice to get this solution by an iterative method.

Iteration Is Minimizing

Getting successive improvements to an initial x-vector, x_0, that converge to the solution to the system $Ax = b$ can be considered to be minimizing the errors in the x-vectors, the *residuals:*

$$r_n = b - Ax_n.$$

For a special class of problems, those whose coefficient matrix is symmetric and positive definite, there is a method that is extremely rapidly convergent, the *conjugate gradient method.* In Chapter 7 we discuss this method of finding the minimum of a function of several variables.

When matrix A is multiplied with vector x, a new transformed vector results. Because the product of matrix A with vector x depends on x, we can say that the product is a function of x because it changes when x is varied—Ax is then a "function of x." Our statement that iteration is minimizing makes sense.

We will not give a full explanation of the conjugate gradient method at this point, only give one example where it works and another where it doesn't.

Consider this small system whose coefficient matrix is symmetric and positive definite:*

$$A = \begin{bmatrix} 4 & -3 & -1 \\ -3 & 5 & 2 \\ -1 & 2 & 3 \end{bmatrix}, \qquad b = \begin{bmatrix} 7 \\ 2 \\ -3 \end{bmatrix},$$

whose solution is $x = [3.9167, 3.5833, -2.0833]^T$. If we start with $x_0 = [0, 0, 0]^T$, Gauss–Seidel iterations converge in 20 iterations. With the Jacobi method, there is no convergence; the successive x-vectors after 34 iterations oscillate about the true answer:

$$\text{Iteration 34: } x = [3.5833, 3.9166, -1.7500]^T,$$
$$\text{Iteration 35: } x = [4.2500, 3.2500, -2.4166]^T,$$

whose averages are exactly the solution.

If the conjugate gradient method is applied, again with $x_0 = [0, 0, 0]^T$, we get these results:

$$x_1 = [2.4520, 0.7006, -1.0508]^T,$$
$$x_2 = [4.0670, 3.4771, -1.6197]^T,$$
$$x_3 = [3.9167, 3.5833, -2.0833]^T,$$

and we obtain the exact solution in three tries!

* A matrix is positive definite if and only if the determinants of all its leading minors are positive. The leading minors are the submatrices whose upper-left elements are the diagonal elements of the matrix. This matrix is clearly symmetric. It is positive definite because the determinants of it leading minors are 24, 11, and 3.

The conjugate gradient method will always converge in n tries with a system of n-equations; it is the preferred iterative method for systems that have the necessary conditions. Still, each iteration of the conjugate gradient method is more expensive than Jacobi or Gauss–Seidel.

If we attempt to use the method when the coefficient matrix is not symmetric, it fails. With this set of two equations:

$$\begin{bmatrix} 2 & 1 & 3 \\ -1 & 3 & 2 \end{bmatrix}$$

which obviously is solved with $x = [1, 1]^T$, the conjugate gradient method actually diverges from $x_0 = [0, 0]^T$, while Gauss–Seidel converges in 7 iterations and Jacobi in 12.

2.6 Parallel Processing

We have mentioned that the operation of many numerical methods can be speeded up by the proper use of parallel processing or distributed systems. In this section, we describe how vector/matrix operations, Gaussian elimination, and Jacobi iteration can be efficiently performed in a parallel or distributed processing environment. We shall show how much the performance can be improved depending on the topology of the network in each case and pay special attention to the implementation of Gaussian elimination.

Vector/Matrix Operations

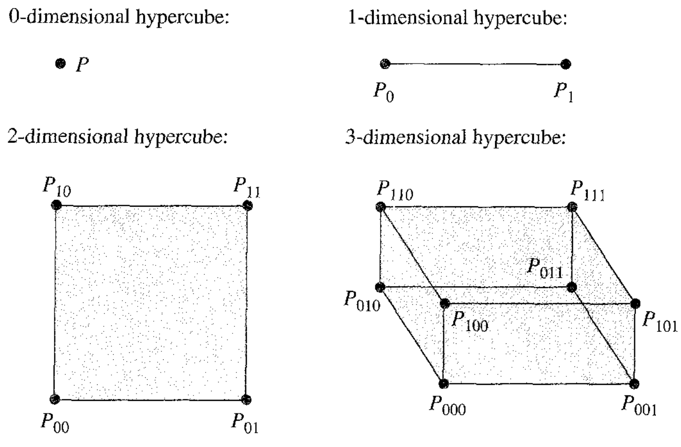
For inner products, a very elementary case, we assume we have two vectors, v, u, of length n, and an equal number of parallel processors, proc(i), $i = 1 \ldots n$, where each proc(i) contains the components, v_i, u_i. Then the multiplication of all the $v_i * u_i$ can be done in parallel in one time unit. In Section 0.6, we found that if the processors are connected suitably we can actually do the addition part in $\log(n)$ time units. Thus, we can estimate the time for an inner product as $1 + \log(n) = O(\log(n))$.

This assumes a high degree of connectivity between the processors. There has been much study of such connectivity. These different designs are referred to as the topologies of the systems. In our present example, we assume the topology of a hypercube. However, before we describe that design, we shall introduce the simpler topology of the linear array. Suppose our processors were only connected as a linear array in which the communication send/receive is just between two adjacent processors:

$$P_1 \leftrightarrow P_2 \leftrightarrow \cdots \leftrightarrow P_{n-1} \leftrightarrow P_n.$$

Then our addition of the n elements would be $n/2$ time steps, because we could do an addition at each end in parallel and proceed to the middle.

The n-dimensional hypercube is a graph with 2^n vertices in which each vertex has n edges (is connected to n other vertices). This graph can be easily defined recursively because there is an easy algorithm to determine the order in which the vertices are connected.



Two vertices are *adjacent* if and only if the indices differ in exactly one bit. We can get to the $(n + 1)$-dimensional hypercube by making two copies of the n-dimensional hypercube and then adding a zero to the leftmost bit of the first n-dimensional cube and then doing the same with a 1 to the second cube. It was this kind of connectivity that allowed us to make the addition of n numbers in $\log(n)$ time steps in Section 0.6. There are many other designs for connecting the processors. Such designs include names like star, ring, torus, mesh, and others. However, for the rest of this section, we shall assume that we have the processors optimally connected.

For the *matrix/vector product, Ax*, we assume that processor proc(i) contains the ith row of A as well as the vector, x. Because one processor is performing this dot product of row i of A and of vector x, this could be done in $2n$ units of time. However, because the other processors are proceeding in parallel, the whole operation will only take $O(n)$ units of time. We would have had a less efficient algorithm had we made use of the inner product algorithm above on the individual rows of A and the vector x.

For a linear array of processors, Bertsekas and Tsitsiklis give the following time for our more simplified case. The time is

$$\alpha n + (n - 1)(\beta + \gamma),$$

where α is the time for an addition or multiplication, β is the time for a transmission of the product along the link, and γ is a positive constant.

For the *matrix/matrix product, AB*, for two $n \times n$ matrices, we suppose that we have n^2 processors. Before we start our computations, each processor, $P_{i,j}$, will have received the values for row i of A and column j of B. Based on our previous discussion, we can expect the time to be $O(n)$. For n^3 processors, this can be reduced to just $O(\log(n))$ time. Here each processor, proc(i,k,j), would store the elements, $A_{i,k}$, $B_{k,j}$. Then all the multiplications can be done in parallel, and the additional $(\log(n))$ time units are for the additions.

Gaussian Elimination

Recall how we achieve a solution to a system of linear equations through Gaussian elimination. We first perform a sequence of row reductions on the augmented matrix $(A : b)$ until the coefficient matrix A is in upper-triangular form. Then we employ back-substitution to find the solution.

To see how this can be done in a parallel-processing environment, we must examine the row-reduction phase and the back-substitution phase in some detail.

We begin with the row-reduction phase: Consider the following example of the first stage of row reduction of a 4×4 system with one right-hand side:

$$\begin{bmatrix} 1 & 2 & 1 & 3 & 4 \\ 2 & 5 & 4 & 3 & 4 \\ 1 & 4 & 2 & 3 & 3 \\ 3 & 2 & 4 & 1 & 8 \end{bmatrix} \begin{matrix} \\ R_2 - (2/1) * R_1 \\ R_3 - (1/1) * R_1 \rightarrow \\ R_4 - (3/1) * R_1 \end{matrix} \begin{bmatrix} 1 & 2 & 1 & 3 & 4 \\ 0 & 1 & 2 & -3 & -4 \\ 0 & 2 & 1 & 0 & -1 \\ 0 & -4 & 1 & -8 & -4 \end{bmatrix}.$$

Although each of these row reductions depends on the elements of row 1, they are completely independent of one another. For example, the elements of rows 2 and 4 play no part in the row operations performed on row 3. Thus, the row reductions on rows 2, 3, and 4 can be computed simultaneously.

If we are computing in a parallel processing environment, we can take advantage of this independence by assigning each row-reduction task to a different processor:

$$\begin{bmatrix} 1 & 2 & 1 & 3 & 4 \\ 2 & 5 & 4 & 3 & 4 \\ 1 & 4 & 2 & 3 & 3 \\ 3 & 2 & 4 & 1 & 8 \end{bmatrix} \begin{matrix} \\ \rightarrow \text{Processor 1: } R_2 - (2/1) * R_1 \\ \rightarrow \text{Processor 2: } R_3 - (1/1) * R_1 \rightarrow \cdots\cdot \\ \rightarrow \text{Processor 3: } R_4 - (3/1) * R_1 \end{matrix}$$

Suppose each row assignment statement requires 4 time units, one for each element in a row. Then the sequential algorithm performs this stage of the row reduction in 12 time units, whereas we need only 4 time units for the parallel algorithm. This example of parallel processing on the first stage of row reduction of a 4×4 system matrix generalizes to any row reduction in stage j of an $n \times n$ system matrix.

Recall that there are $n - 1$ row-reduction stages in Gaussian elimination, one for each of the n columns of the coefficient matrix except for the last column. This suggests that we need $n - 1$ processors to do the reduction in parallel.* Also recall that each row-reduction stage j creates zeros in every cell below the diagonal in the jth column. The following two pseudocodes compare the use of a single processor with the use of n processors to perform the entire row-reduction phase of Gaussian elimination.

* Even so, we will need n processors in the final algorithm, as will be seen.

Algorithms for Row Reduction in Gaussian Elimination

Sequential Processing (without pivoting)

```
For $j = 1$ To $(n - 1)$
    For $i = (j + 1)$ To $n$
        For $k = j$ To $(n + 1)$
            $a[i, k] = a[i, k] - a[i, j]/a[j, j] * a[j, k]$
        End For $k$
    End For $i$
End For $j$
```

Parallel Processing

```
For $i = 1$ To $(n - 1)$          (Counts stages = columns)
    For $k = i$ To $(n + 1)$      (On Processor $j = (i + 1)$ To $n$)
        $a[i, k] = a[i, k] - a[i, j]/a[j, j] * a[j, k]$
    End For $k$
End For $i$
```

If we total the arithmetic operations to carry out the reduction of an $n \times n$ coefficient matrix to upper-triangular form, we find that the sequential algorithm requires $O(n^3)$ successive operations and that the parallel algorithm with n processors accomplishes the same task in $O(n^2)$ successive operations.

What happens if we have more than n processors? As indicated in our earlier discussion, we can speed up the reduction process even more. Suppose we increase the number of processors from n to, say, $n^2 + n$. We can effectively use this extra power for Gaussian elimination just as we did for the matrix/vector operations earlier. The complete algorithm for the row-reduction phase of Gaussian elimination on these processors runs only $O(n)$ successive steps, and each step requires just three time steps for one subtraction, one division, and one multiplication. If, as before, we label each processor as proc(i,j), $i = 1, \dots, n$, $j = 1, \dots, n + 1$, proc(i,j) is responsible for each element a_{ij} of the matrix $[A: b]$. We can now rewrite the algorithm for parallel processing to reflect this improvement:

```
For $i = 2$ To $n$
    {On Processor $(j, k)$}
    $a[j, k] = a[j, k] - a[j, i]/a[i, i] * a[j, k]$
End For $i$
```

The row-reduction phase of Gaussian elimination leaves us with an upper-triangular coefficient matrix and an appropriately adjusted right-hand side. In the sequential algorithm we now find a solution using back-substitution. Before we consider parallelization of back-substitution, let us examine the activity of the processors during row reduction in greater detail.

As we have observed, the processors responsible for computations on the elements of row 1 sit idle during row reduction, because those elements of the matrix never change. In addition, after the first row-reduction stage in which zeros are placed in the first column, the processors for row 2 also sit idle. In fact, each stage of row reduction frees $n + 1$ processors.

It is natural to wonder if these idle processors could be employed in our algorithm. Indeed, they can. We use them to perform row reductions above the diagonal at the same time that corresponding row reductions occur below the diagonal. Thus at each stage j of the reduction, zeros appear in all but the diagonal element of the jth column.

This diagram illustrates our improved procedure, continuing the simple 4×4 example examined before and doing stages 3 and 4:

$$\rightarrow \begin{bmatrix} 1 & 0 & -3 & 9 & 12 \\ 0 & 1 & 2 & -3 & -4 \\ 0 & 0 & -3 & 6 & 7 \\ 0 & 0 & 9 & -20 & -20 \end{bmatrix} \begin{array}{l} R_1 - (3/3) * R_3 \\ R_2 - (-2/3) * R_3 \rightarrow \\ \\ R_4 - (9/-3) * R_3 \end{array} \begin{bmatrix} 1 & 0 & 0 & 3 & 5 \\ 0 & 1 & 0 & 1 & \frac{2}{3} \\ 0 & 0 & -3 & 6 & 7 \\ 0 & 0 & 0 & -2 & 1 \end{bmatrix}$$

$$\begin{array}{l} R_1 - (3/-2) * R_4 \\ R_2 - (1/-2) * R_4 \\ R_3 - (6/-2) * R_4 \\ \\ \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{13}{2} \\ 0 & 1 & 0 & 0 & \frac{7}{6} \\ 0 & 0 & -3 & 0 & 10 \\ 0 & 0 & 0 & -2 & 1 \end{bmatrix} \rightarrow x = \begin{bmatrix} \frac{13}{2} \\ \frac{7}{6} \\ -\frac{10}{3} \\ -\frac{1}{2} \end{bmatrix}.$$

The result of n such reductions—one for each column of the coefficient matrix A—is $[D : b']$, where D is a diagonal matrix and b' is an appropriately adjusted right-hand-side vector.

Specifically, the solution x for $Dx = b'$ also satisfies $Ax = b$. This solution is the vector x whose elements are $x_i = b_i'/d_{ii}$ for $i = 1, \ldots, n$. We can use n processors to perform these n divisions simultaneously. Notice that the back-substitution phase of Gaussian elimination is no longer necessary! We find that the Gauss–Jordan procedure is preferred when doing parallel processing!

The parallel algorithm for n^2 processors required n time units for row reduction, and one additional time unit for division. Recall that the sequential algorithm required $O(n^3)$ time units. To understand the magnitude of the improvement in running time, consider that a solution achieved in 10 seconds via the parallel algorithm would require around 15 minutes via the sequential algorithm.*

Our final parallel algorithm for solving a system of linear equations more closely resembles the Gauss–Jacobi solution technique than it does Gaussian elimination. This is not surprising. It is not uncommon for good parallel algorithms to differ dramatically from their speediest sequential counterparts.

* This neglects the overhead of interprocessor communications.

Problems in Using Parallel Processors

It is essential to mention some important concerns that have been neglected in the preceding discussion. When we actually implement this parallel algorithm, we must worry about four issues.

1. The algorithm described here does not pivot. Thus, our solution may not be as numerically stable as one obtained via a sequential algorithm with partial pivoting. In fact, if a zero appears on the diagonal at any stage of the reduction, we are in big trouble. Bertsekas and Tsitsiklis observe that Gaussian elimination with pivoting can have an upper bound of $O(n \log(n))$ time when $n^2 + n$ processors are used, and still $O(n^2)$ time in the case of n processors.
2. The coefficient matrix A is assumed to be nonsingular. It is easy to check for singularity at each stage of the row reduction, but such error-handling will more than double the running time of the algorithm.
3. We have ignored the communication and overhead time costs that are involved in parallelization. Because of these costs, it is probably more efficient to solve small systems of equations using a sequential algorithm.
4. Other, perhaps faster, parallel algorithms exist for solving systems of linear equations. One technique, which is easily derived from ours, involves computing A^{-1} via row operations and simply multiplying the right-hand side to get the solution $x = A^{-1}b$. Another technique requires computing the coefficients of the characteristic polynomial and then applying these coefficients in building A^{-1} from powers of A. This method finds a solution in only $[2 \log_2 n + O(\log n)]$ time units, but it requires $n^4/2$ processors to do so. In addition, it often leads to numeric instability.*

Despite these concerns, our algorithm is an effective approach to solving systems of linear equations in a parallel environment

Iterative Solutions—The Jacobi Method

The method of simultaneous displacements (the Jacobi method) that was discussed in Section 2.5 is adapted very simply to a parallel environment. Recall that at each iteration of the algorithm a new solution vector $x^{(n+1)}$ is computed using only the elements of the solution vector from the previous iteration, $x^{(n)}$. In fact, the elements of the vector $x^{(n)}$ can be considered fixed with respect to the iteration $(n + 1)$. Thus, though each element $x_i^{(n+1)}$ in the vector $x^{(n+1)}$ depends on the elements in $x^{(n)}$, these $x_i^{(n+1)}$ are independent of one another and can be computed simultaneously.

Suppose the solution vector x has m elements. Then each iteration of the Jacobi algorithm in a sequential environment requires m assignment statements. If we have m processors in parallel, these m assignment statements can be performed simultaneously, thereby reducing the running time of the algorithm by a factor of m.

* JaJa (1992) describes these alternative algorithms in some detail.

Notice also that each assignment statement is a summation over approximately m terms. As demonstrated in Section 0.6, this summation can be performed in $\log_2 m$ time units with m parallel processors, compared to m time units for sequential addition. If m^2 processors are available, we can employ both of these parallelizations and reduce the time for each iteration of the Jacobi algorithm to $\log_2 m$ time units. This is a significant speedup over the sequential algorithm, which requires m^2 time units per iteration.

As seen in Section 2.5, the actual running time of the algorithm (the number of iterations) depends on the degree of diagonal dominance of the coefficient matrix. Parallelization decreases only the time required for each iteration.

Because Gauss–Seidel iteration requires that the new iterates for each variable be used after they have been obtained, this method cannot be speeded up by parallel processing. Again, the preferred algorithm for sequential processing is not the best for parallel processing.

Exercises

Section 2.1

1. For these four matrices:

$$A = \begin{bmatrix} 3 & 2 & -1 \\ 2 & 1 & 3 \\ -3 & -2 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 4 \\ -1 & -2 \\ 3 & 0 \end{bmatrix},$$

$$C = \begin{bmatrix} 3 & -1 & 3 \\ 1 & 1 & -2 \\ 4 & -2 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & -2 & 3 \\ -1 & 5 & 0 \\ -2 & -1 & 6 \end{bmatrix},$$

a. Which pairs can be added? Find the sums.
b. Which pairs can be subtracted? Get their differences, then repeat in opposite order.
c. Which pairs can be multiplied? Find the products.
d. Which of these has a trace? Compute the traces.

2. Get the transpose for each matrix in Exercise 1. Then repeat each part of Exercise 1 with the transposes.

▶3. For these vectors:

$$v1 = \begin{bmatrix} -2 \\ 3 \\ 4 \end{bmatrix}, \quad v2 = [3 \ 4 \ -1], \quad v3 = \begin{bmatrix} 4 \\ 2 \\ -3 \end{bmatrix},$$

a. Which pairs can be multiplied? Find the products.
b. Using the matrices in Exercise 1, which matrices can multiply these vectors? Compute the products.
c. Can any of these vectors multiply with one of the matrices of Exercise 1 in the order $v * M$? Get the products for those that do.
d. Find the product of each vector times its transpose. Repeat with the transpose as the first factor.

4. Given the matrices:

$$A = \begin{bmatrix} -2 & 1 & 2 \\ 2 & 3 & -2 \\ 1 & -2 & -3 \end{bmatrix}, \quad B = \begin{bmatrix} -2 & 3 & 5 \\ 2 & 1 & -4 \\ 4 & -1 & 6 \end{bmatrix},$$

a. Find BA, B^3, AA^T.
b. Get det (A) and det (B).
c. A square matrix can always be expressed as the sum of a lower-triangular matrix L and an upper-triangular matrix U. Find two different combinations of L and U for both A and B.

▶5. Let

$$A = \begin{bmatrix} 2 & 9 \\ 3 & -10 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 6 & 2 \\ 4 & -1 & 3 \\ 1 & -3 & -1 \end{bmatrix}.$$

a. Find the characteristic polynomials of both A and B.
b. Find the eigenvalues of both A and B.
c. Is [0.2104, 0.8401] an eigenvector of A?

6. Write this as a set of equations:

$$\begin{bmatrix} 4 & 2 & -2 & -1 \\ 0 & 4 & 1 & 2 \\ 3 & -2 & 1 & 2 \\ 2 & 0 & 3 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \\ 2 \\ 3 \end{bmatrix}.$$

7. Write these equations in matrix form:

$$\begin{aligned} 6x - 2y + 3z &= 12, \\ x + y - 4z &= 8, \\ -2x - 3y &= 12. \end{aligned}$$

8. It is true that $(A * B)^T = B^T * A^T$.

a. Test this statement with two 3×3 matrices of your choice.

b. Is this true if A is 3×2 and B is 2×3.

c. Prove the statement.

9. For matrix A, write the transposition matrices that perform the interchanges.

$$A = \begin{bmatrix} 3 & 5 & -2 & -1 & 0 \\ -2 & 3 & 4 & -5 & 3 \\ 5 & 2 & -1 & 3 & -6 \\ 2 & -3 & 4 & 2 & 0 \\ -5 & 5 & 3 & -3 & 4 \end{bmatrix}.$$

a. Row 3 with row 5.

b. Column 2 with column 1.

▶ c. Both row 3 with row 5 and column 4 with column 2.

d. Multiply A with each of your matrices in parts (a), (b), and (c) to confirm that the correct interchanges occur.

e. What happens if the transposition matrix of part (a) is used to postmultiply with A rather than to premultiply?

10. Confirm that $P^T * P = I$ and that $P^T = P$ for each transposition matrix of Exercise 9.

Section 2.2

11. a. Solve by back-substitution:

$$\begin{aligned} 3x_1 + 3x_2 + x_3 &= 12, \\ -4x_2 - 3x_3 &= -10, \\ 2x_3 &= 4. \end{aligned}$$

b. Solve by forward-substitution:

$$\begin{aligned} 3x_1 &= 15, \\ 2x_1 - x_2 &= 10, \\ 5x_1 + x_2 - 2x_3 &= 5. \end{aligned}$$

12. The first procedure described in Section 2.2 is sometimes called "Naive Gaussian Elimination." Use it to solve Exercises 13 and 14.

13. Solve the following (given as the augmented matrix):

$$\left[\begin{array}{rrr|r} 3 & 1 & -4 & 7 \\ -2 & 3 & 1 & -5 \\ 2 & 0 & 5 & 10 \end{array}\right].$$

14. Here is a system of equations that is called "ill-conditioned," meaning that the solution is not easy to get accurately. Section 2.4 discusses this; here, we give a "taste" of the problem. This is the system as an augmented matrix:

$$\begin{bmatrix} 3 & 2 & 4 & 9 \\ 8 & -6 & -8 & -6 \\ -1 & 2 & 3 & 4 \end{bmatrix}.$$

You can see that $x = [1, 1, 1]^T$ is the solution.

a. Confirm the solution by doing naive Gaussian elimination using exact arithmetic (use fractions throughout).

b. Now get the solution using only three significant figures in your computations. Observe that the solution is different.

c. Compute the solution when the system is changed only slightly: Change the coefficient in the first column of the first row to 3.1. Use more precise computations, perhaps single or even double precision. Observe that this makes a large change in the solution.

▶15. Use Gaussian elimination with partial pivoting to solve the equations of Exercise 13. Are any row interchanges needed?

16. In which column(s) are row interchanges needed to solve the equations in Exercise 6 by Gaussian elimination with partial pivoting?

17. Solve this system by Gaussian elimination with partial pivoting:

$$\begin{bmatrix} 1 & -2 & 4 & 6 \\ 8 & -3 & 2 & 2 \\ -1 & 10 & 2 & 4 \end{bmatrix}.$$

a. How many row interchanges are needed?

b. Solve again but use only three significant digits of precision.

c. Repeat part (b) without any row interchanges. Do you get the same results?

18. Solve the system

$$\begin{aligned} 2.51x + 1.48y + 4.53z &= 0.05, \\ 1.48x + 0.93y - 1.30z &= 1.03, \\ 2.68x + 3.04y - 1.48z &= -0.53. \end{aligned}$$

a. Use Gaussian elimination, but use only three significant digits and do no interchanges. Observe the small divisor in reducing the third column. The correct solution is $x = 1.45310$, $y = -1.58919$, $z = -0.27489$.

b. Repeat part (a) but now do partial pivoting.

c. Repeat part (b) but now chop the numbers rather than rounding.

d. Substitute the solutions found in (a), (b), and (c) into the equations. How well do these match the original right-hand sides?

19. Use the Gauss–Jordan method to solve the equations of Exercise 17.

20. Use the Gauss–Jordan method to solve the equations of Exercise 18.

▶21. Confirm that the Gauss–Jordan method requires $O(n^3)$ total arithmetic operations.

22. What if we solve a system with m right-hand sides rather than just one? How many total operations are then required for both Gaussian elimination and the Gauss–Jordan method?

23. Suppose that multiplication takes twice as long to do as an addition/subtraction and that division takes three times as long (which used to be true). For a system of ten equations, how much longer does it take to get a solution compared to when each operation takes the same amount of time? Do this for both Gaussian elimination and for Gauss–Jordan.

24. Write an algorithm for the Gauss–Jordan method. Provide for partial pivoting.

a. When there is only one right-hand side.
b. When there are m right-hand sides.

25. Modify the algorithm for Gaussian elimination to incorporate scaled partial pivoting.

26. Repeat Exercise 25 but now employ an order vector to avoid actually interchanging the rows.

27. Use scaled partial pivoting to solve:

$$\begin{bmatrix} 4.13 & -2.20 & 0.95 & 3.02 \\ 6.14 & 4.45 & -1.45 & -4.02 \\ 1.03 & 1.86 & 0.44 & 5.22 \end{bmatrix}.$$

a. Employ six significant digits.
b. Repeat with only three significant digits. Is the solution much different?

▶28. If a comparison takes one-half as long as an addition/subtraction and to interchange two numbers takes twice times as long, how much time is saved by using an order vector rather than doing the actual row interchanges? Express the answer in terms of addition times for a system of n equations.

29. A system of two equations can be solved by graphing the two lines and finding where they intersect. (Graphing three equations could be done, but locating the intersection of the three planes is difficult.) Graph this system; you should find the intersection at (6, 2).

$$\begin{aligned} 0.1x + 51.7y &= 104, \\ 5.1x - 7.3y &= 16. \end{aligned}$$

a. Now, solve using three significant digits of precision and no row interchanges. Compare the answer to the correct value.
b. Repeat part (a) but do partial pivoting.
c. Repeat part (a) but use scaled partial pivoting. Which of part (a) or (b) does this match, if any?
d. Complete pivoting chooses the largest of all of the coefficients at the current stage as the pivot element. Repeat part (a) with complete pivoting. How does this answer compare to those of parts (a), (b), and (c)?

30. The determinant of a matrix can be found by expanding in terms of its minors. Compare the number of arithmetic operations when done this way with the number if the matrix is reduced to a triangular one by Gaussian elimination. Do this for a 4×4 matrix. Then find a relation for an $n \times n$ matrix.

▶31. When you solved Exercise 17, you could have saved the row multipliers and obtained a LU equivalent of the coefficient matrix. Use this LU to solve Exercise 17 but with right-hand sides of:

a. $[1, -3, 5]^T$.
b. $[-3, 7, -2]^T$.

32. Repeat Exercise 17, but now use the LU.

33. Repeat Exercise 27, but now use the LU.

▶34. Given this tridiagonal system:

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 100 \\ -1 & 4 & -1 & 0 & 0 & 0 & 200 \\ 0 & -1 & 4 & -1 & 0 & 0 & 200 \\ 0 & 0 & -1 & 4 & -1 & 0 & 200 \\ 0 & 0 & 0 & -1 & 4 & -1 & 200 \\ 0 & 0 & 0 & 0 & -1 & 4 & 100 \end{bmatrix}.$$

a. Solve the system using the algorithm for a compacted system matrix that has n rows but only four columns.
b. How many arithmetic operations are needed to solve a tridiagonal system of n equations in this compacted arrangement? How does this compare to solving such a system with Gaussian elimination without compacting?

35. The system of Exercise 34 is an example of a symmetric matrix. Because the elements at opposite positions across the diagonal are exactly the same, it can be stored as a matrix with n rows but only three columns.

a. Write an algorithm for solving a symmetric tridiagonal system that takes advantage of such compacting.

b. Use the algorithm from part (a) to solve the system in Exercise 34.
c. How many arithmetic operations are needed with this algorithm for a system of n equations?

▶**36.** Write the algorithm for LU reduction that puts ones on the diagonal of U.

37. When are row interchanges absolutely required in forming the LU equivalent of matrix A?

38. Given system A:

$$A = \begin{bmatrix} 2 & -1 & 3 & 2 \\ 2 & 2 & 0 & 4 \\ 1 & 1 & -2 & 2 \\ 1 & 3 & 4 & -1 \end{bmatrix}.$$

Find the LU equivalent of matrix A that has 2's in each diagonal position of L rather than 1's.

39. Repeat Exercise 38, but now make the diagonal elements of L equal to [1, 2, 3, 4].

40. If you were asked to create a LU reduction of matrix A that has at least one zero on a diagonal,

a. When can you do this, putting the zero(s) on the diagonal of L?
b. When can you do this, putting the zero(s) on the diagonal of U?
c. Give examples where A is 3 × 3.

Section 2.3

▶**41.** Which of these matrices are singular?

a. $$\begin{bmatrix} -2 & 1 & -1 \\ -3 & 4 & -6 \\ 2 & 7 & 15 \end{bmatrix}.$$

b. $$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 3 & 4 & 1 \\ 7 & 0 & -4 & 2 \\ 4 & -6 & -9 & 0 \end{bmatrix}.$$

c. $$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 3 & 4 & 1 \\ 7 & 0 & -4 & 2 \\ 4 & -6 & 9 & 0 \end{bmatrix}.$$

42. For this matrix:

$$A = \begin{bmatrix} 3 & 5 & 1 \\ -1 & 3 & 2 \\ a & b & -1 \end{bmatrix},$$

a. Find values for a and b that make A singular.
b. Find values for a and b that make A nonsingular.

43. The matrix in Exercise 41, part (b), is singular.

a. That means its rows form vectors that are linearly dependent. Find the weighting factors for the rows that makes their sum zero.
b. Repeat part (a), but with the columns.

44. Do these equations have a solution? Find the solution if it exists. Explain why when it doesn't.

a. $$\begin{aligned} -2x + 3y + z &= 2, \\ -3x + y + z &= 5, \\ x + y - z &= -5, \\ 3y + z &= 0. \end{aligned}$$

b. $$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & -2 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 4 \end{bmatrix}.$$

c. $$\begin{bmatrix} 2 & -1 & 6 & 1 \\ 1 & 0 & 2 & 0 \\ 3 & 2 & 2 & 0 \end{bmatrix}.$$

▶**45.** The Hilbert matrix is a classic case of the pathological situation called "ill-conditioning." The 4 × 4 Hilbert matrix is

$$H_4 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}.$$

For the system $Hx = b^T$, with $b^T = [25/12, 77/60, 57/60, 319/420]$, the exact solution is $x^T = [1, 1, 1, 1]$.

a. Show that the matrix is ill-conditioned by showing that it is nearly singular.
b. Using only three significant digits (chopped) in your arithmetic, find the solution to $Hx = b$. Explain why the answers are so poor.
c. Using only three significant digits, but rounding, again find the solution and compare it to that obtained in part (b).

46. For this system of equations

$$\begin{aligned} ax + 4y + z &= 6, \\ 2ax - y + 2z &= 3, \\ x + 3y + az &= 5, \end{aligned}$$

a. What value of a gives a unique solution to the system?
b. What value of a makes the system have no solution?

c. What value of a makes the system have an infinity of solutions?

47. Solve this pair of equations by Gaussian elimination:

$$0.2205x + 0.1254y = 0.6606,$$
$$0.4457x + 0.2506y = 0.8897.$$

a. Use only four significant digits in the solution.
b. Compare the solution using seven significant digits with that of part (a). Explain why the solutions are different.

48. Use the Gaussian elimination method to triangularize this matrix and from that get its determinant:

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 1 & 1 & 3 \\ -3 & 0 & 5 \end{bmatrix}.$$

49. Repeat Exercise 48 but convert matrix A to a LU that has ones on the diagonal of U rather than on L.

▶50. Change the element in row 3, column 3 of Exercise 48 from $+5$ to -5 and repeat Exercise 48. Explain why this causes the determinant to become smaller.

51. For this matrix:

$$\begin{bmatrix} 4 & -2 & 3 & -5 \\ 3 & 3 & 5 & -8 \\ -6 & -1 & 4 & 3 \\ -4 & 2 & -3 & 5 \end{bmatrix},$$

a. Show that the matrix is singular.
b. Change the element in row 4 column 4 from 5.0 to 5.1 and get its determinant. Even though this matrix is larger than the matrix in Exercise 48 and most of its elements are greater, why is the determinant a smaller number than for Exercise 48?

52. First show that det $(A * B) = \det(A) * \det(B)$ for two 4×4 matrices that you compose, then prove that this will always be true for any two square matrices of the same size.

53. If some of the elements of a matrix are very small in magnitude and others are very large, will the value of its determinant be large or small? What if only one element is very large and the rest very small? What if only one is very small and the rest very large? Are there situations where the magnitudes of the elements are not important?

▶54. Get the inverse of the matrix in Exercise 48.

a. Do it through Gaussian elimination.
b. Repeat, but with the Gauss–Jordan method.
c. How many arithmetic operations are used in parts (a) and (b)?

55. Find the inverse of the matrix in part (b) of Exercise 51.

a. Do this using only three significant digits of precision.
b. Repeat part(a), but now use seven digits. Why are the results different?

56. Repeat Exercise 55, but for the matrix in Exercise 48. Why are the results with three digits the same as those with seven when rounded?

57. Find the determinant of matrix A and the determinant of its inverse.

$$A = \begin{bmatrix} -2 & 3 & 2 & 8 \\ 1 & 3 & -2 & 6 \\ 5 & -1 & 3 & 9 \\ 2 & 3 & 8 & -1 \end{bmatrix}.$$

Section 2.4

▶58. Evaluate the 1-, 2- and ∞-norms of these vectors:

a. $[3.06, -2.11, 8.12, -4.45]$.
b. $[-5, -3, 2, 7]$.

59. Verify the relations of Eq. (2.14) for each of the definitions of a vector norm.

60. Which vector norm usually gives the smallest value? Is there an instance when all vector norms have the same value?

▶61. Evaluate the 1-, 2-, and ∞-norms of these matrices:

$$A = \begin{bmatrix} 5 & -9 & 6 \\ 2 & -7 & 4 \\ 1 & 5 & 8 \end{bmatrix},$$

$$B = \begin{bmatrix} 10.2 & 2.4 & 4.5 \\ -2.3 & 7.7 & 11.1 \\ -5.5 & -3.2 & 0.9 \end{bmatrix}.$$

62. Is the spectral norm of a matrix always the smallest norm? Is there a case where all matrix norms have the same value?

63. For the matrices of Exercise 61, compare these norms:

a. norm $(A + B)$ with norm (A) + norm(B).
b. norm $(A * B)$ with norm (A) * norm(B).
c. norm (A^2) with norm (A) * norm(A).
d. What conclusion do you draw from these results?

▶64. Find the ∞-norm of the Hilbert matrix of Exercise 45.

65. If a matrix is nearly singular, how does its norm compare to the norm of its inverse?

66. Given this system of equations:

$$\begin{bmatrix} 6.03 & 1.99 & 3.01 & 1 \\ 4.16 & -1.23 & 1.27 & 1 \\ -4.81 & 9.34 & 0.987 & 1 \end{bmatrix},$$

a. Solve with a precision of ten significant digits.
b. Solve again with a precision of four significant digits.
c. What happens if you solve with a precision of only three significant digits?
d. Let x be the solution from part (a) and let ξ be the solution from part (b). Let $e = x - \xi$. What are the norms of e?
e. Is the system ill-conditioned? What is the condition number of the coefficient matrix? Compute this for each definition of condition number.

67. Repeat Exercise 66 after changing element a_{32} to -9.34. Why are the results so different?

68. What if we discover that one of the coefficients in Exercise 66 is slightly in error due to measuring errors? Specifically, suppose that a_{13} should be 3.02 rather than 3.01. How does this affect the answers to parts (a) and (b) of Exercise 66?

69. What are the residuals for the imperfect solutions of Exercises 66, 67, and 68?

▶70. What is the condition number for the coefficient matrix of Exercise 67. Why is it so different from that for Exercise 66?

71. Verify Eq. (2.16) with the residuals from Exercises 66 and 67.

72. Verify Eq. (2.18) with the residuals from Exercises 66 and 67.

73. Apply iterative improvement to the solution from Exercise 66, part (b).

74. Compare the condition numbers for the Hilbert matrix of order-4:

a. Using exact numbers (use fractional numbers throughout).
b. Using floating-point values with only three significant digits.

▶75. Prove that cond $(A) \geq 1$ for any square matrix. Are there any exceptions to this?

76. For what values of a does this matrix have a condition number greater than 100?

$$\begin{bmatrix} a & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & a \end{bmatrix}$$

77. Find a 2×2 matrix whose condition number is exactly 289 using infinity norms.

Section 2.5

▶78. Solve this system with the Jacobi method. First rearrange to make it diagonally dominant if possible. Use [0, 0, 0] as the starting vector. How many iterations to get the solution accurate to five significant digits?

$$\begin{bmatrix} 7 & -3 & 4 & 6 \\ -3 & 2 & 6 & 2 \\ 2 & 5 & 3 & -5 \end{bmatrix}.$$

▶79. Repeat Exercise 78 with the Gauss–Seidel method. Are fewer iterations required?

80. Is convergence faster in Exercises 78 and 79 if the starting vector is [−0.26602, −0.26602, −0.26602] which is the average value of the elements of the solution vector?

81. Solve this system of equations, starting with the initial vector of [0, 0, 0]:

$$\begin{aligned} 4.63x_1 - 1.21x_2 + 3.22x_3 &= 2.22, \\ -3.07x_1 + 5.48x_2 + 2.11x_3 &= -3.17, \\ 1.26x_1 + 3.11x_2 + 4.57x_3 &= 5.11. \end{aligned}$$

a. Solve using the Jacobi method.
b. Solve using the Gauss–Seidel method.

82. The coefficient matrix of Exercise 81 is diagonally dominant. If the value of the element in position (2, 2) is smaller in magnitude than 5.48, it is no longer diagonally dominant. How small can it be and still converge to a solution by iterating with

a. The Jacobi method?
b. The Gauss–Seidel method?

83. This 2×2 matrix is obviously singular and is almost diagonally dominant. If the right-hand-side vector is [0, 0], the equations are satisfied by any pair where $x = y$.

$$\begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix}.$$

a. What happens if you use the Jacobi method with these starting vectors: [1, 1], [1, −1], [−1, 1], [2, 5], [5, 2]?
b. What happens if the Gauss–Seidel method is used with the same starting vectors as in part (a)?
c. If the elements whose values are −2 in the matrix are changed slightly, to −1.99, the matrix is no longer singular but is almost singular. Repeat parts (a) and (b) with these new matrix.

84. For the system of equations in Exercise 78, find the matrices that correspond to Eqs. (2.23) and (2.25). For which method is the norm of the multiplier of $x^{(n)}$ a smaller number?

85. What is the optimal values of the overrelaxation factors that speed the solutions of Exercise 81?

Section 2.6

86. Section 2.6 says that the time to compute the inner product of two n-component vectors is proportional to $1 + \log(n)$, when n processors are available and each processor holds just one component of each vector. However, we can multiply an $n \times n$ matrix times a vector in $2n$ units of time if each processor holds one entire row of the matrix as well as the vector. Make a table that compares the times for these alternative methods for values of $n = 10 * e$ for $e = 1$ to 5.

▶**87.** Develop an algorithm for inverting an $n \times n$ matrix by parallel processing with approximately n^2 processors.

88. The final algorithm developed in Section 2.6 used $n^2 + n$ processors. Show that this can be further improved so that only $(n + 1)(n - 1) = n^2 - 1$ processors are required.

89. Develop an algorithm for doing Jacobi iterations to solve a system of n linear equations using n^2 processors.

Applied Problems and Projects

APP1. In considering the movement of space vehicles, it is frequently necessary to transform coordinate systems. The standard inertial coordinate system has the N-axis pointed north, the E-axis pointed east, and the D-axis pointed toward the center of the earth. A second system is the vehicle's local coordinate system (with the i-axis straight ahead of the vehicle, the j-axis to the right, and the k-axis downward). We can transform the vector whose local coordinates are (i, j, k) to the inertial system by multiplying by transformation matrices:

$$\begin{bmatrix} n \\ e \\ d \end{bmatrix} = \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos b & 0 & \sin b \\ 0 & 1 & 0 \\ -\sin b & 0 & \cos b \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos c & -\sin c \\ 0 & \sin c & \cos c \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}.$$

Transform the vector $[2.06, -2.44, -0.47]^T$ to the inertial system if $a = 27°$, $b = 5°$, $c = 72°$.

APP2. Exercise 45 showed the pattern for a Hilbert matrix. The $n \times n$ Hilbert matrix can be defined more formally as:

$$H_n = (1/(i + j + 1)).\ i, j = 0, 1, \ldots, n - 1.$$

a. Use this in a program that displays the Hilbert matrix of order-5.
b. What is the condition number of the 9×9 Hilbert matrix, H_9?
c. Solve $H_9 x = [1, 1, 1, 1, 1, 1, 1, 1, 1]^T$. Then change the first component of the right-hand side to 1.01 and solve again. Which component of x is most changed?

APP3. Electrical engineers often must find the currents flowing and voltages existing in a complex resistor network. Here is a typical problem.

Seven resistors are connected as shown, and voltage is applied to the circuit at points 1 and 6 (see Fig. 2.1) You may recognize the network as a variation on a Wheatstone bridge.

Although we are especially interested in finding the current that flows through the ammeter, the computational method can give the voltages at each numbered point (these are called *nodes*) and the current through each of the branches of the circuit. Two laws are involved:

Kirchhoff's law: The sum of all currents flowing into a node is zero.

Ohm's law: The current through a resistor equals the voltage across it divided by its resistance.

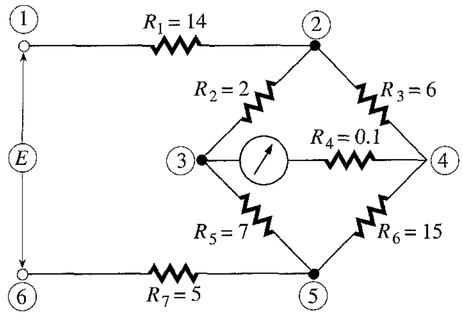


Figure 2.1

We can set up 11 equations using these laws and from these solve for 11 unknown quantities (the four voltages and seven currents). If $V_1 = 5$ volts and $V_6 = 0$ volts, set up the 11 equations and solve to find the voltage at each other node and the currents flowing in each branch of the circuit.

APP4. A square matrix can be partitioned into submatrices. We write

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

where A and D are square. Suppose that

$$M^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Ralston shows that we can get M^{-1} in the following way:

1. Invert A to get A^{-1}.
2. Compute $D - C * A^{-1} * B$.
3. Invert $D - C * A^{-1} * B$, which gives H.
4. Compute $-A^{-1} * B * H$, which gives F.
5. Compute $-H * C * A^{-1}$, which gives G.
6. Compute $A^{-1} - A^{-1} * B * G$, which gives E.

We then get the inverse of M by inverting two smaller matrices and doing some arithmetic operations on matrices that are smaller than M. In this, matrices A and D must not be singular.

Choose some 4×4 matrix. Call it M. Then,

a. Partition M into submatrices. This can be done in three different ways.
b. Get the inverse of M with Ralston's technique. Do all partitionings give the same result?
c. Is Ralston's technique more or less efficient than inverting M directly? Does the difference in operational count depend on the size of M? Does it depend on the way that M is partitioned?

APP5. Mass spectrometry analysis gives a series of peak height readings for various ion masses. For each peak, the height h_j is contributed to by the various constituents. These make different contributions c_{ij} per unit concentration p_i so that the relation

$$h_j = \sum_{i=1}^{n} c_{ij} p_i$$

Table 2.2

Peak number	Component				
	CH_4	C_2H_4	C_2H_6	C_3H_6	C_3H_8
1	0.165	0.202	0.317	0.234	0.182
2	27.7	0.862	0.062	0.073	0.131
3		22.35	13.05	4.420	6.001
4			11.28	0	1.110
5				9.850	1.684
6					15.94

holds, with n being the number of components present. Carnahan (1964) gives the values shown in Table 2.2 for c_{ij}.

If a sample had measured peak heights of $h_1 = 5.20$, $h_2 = 61.7$, $h_3 = 149.2$, $h_4 = 79.4$, $h_5 = 89.3$, and $h_6 = 69.3$, calculate the values of p_i for each component. The total of all the p_i values was 21.53.

APP6. Figure 2.2 shows a structure that might support a bridge (a "truss"). The support at point a is constrained so that it cannot move; the one at point f can move horizontally. There are two external loads, at joints b and d.

In analyzing a truss, the members are assumed not to bend, so the forces within them act only in the direction of the member; these are considered to act from the joint toward the center.

This truss has nine members, so there are nine member forces, F_i, $i = 1, \dots, 9$. If we set the sum of all forces acting either vertically or horizontally within each member, nine equations can be written. Solving these equations gives the values for the nine forces, the F_i.

a. Set up the equations and solve.
b. The matrix is sparse. Is it banded?
c. Can the band width be reduced by reordering the equations?

APP7. The truss in APP6, Figure 2.2, is called *statically determinant,* because nine linearly independent equations can be set up to solve for the nine forces. If a tenth member is added to give better stability to the structure, as shown in Figure 2.3, there are ten member forces to be determined but only nine force equations can be written. This truss is called *statically indeterminant.*

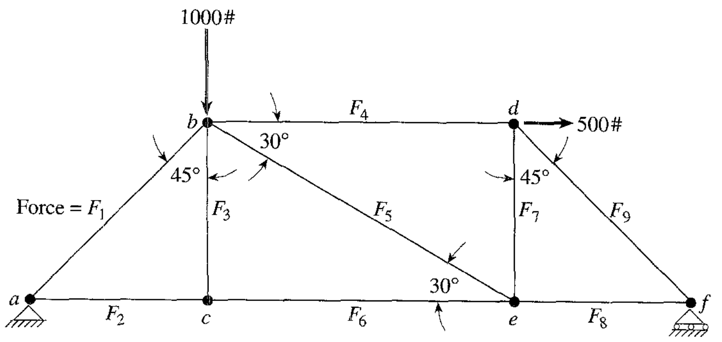


Figure 2.2

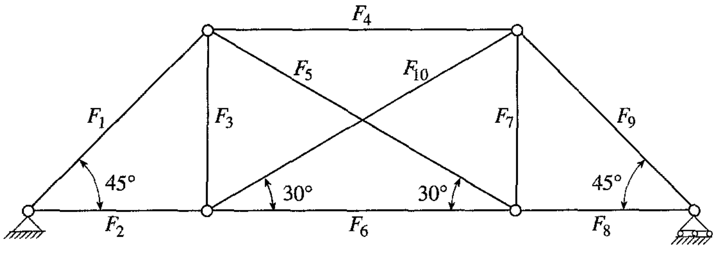


Figure 2.3

A solution can be found if the stretching or compression of the members is considered. We need to solve a set of equations that gives the displacement of each joint; these are of the form $ASA^Tx = P$. We get the tensions in the members, f, by the matrix multiplication $SA^Tx = f$

The required matrices and vectors are

$$A = \begin{bmatrix} 0.7071 & 0 & 0 & -1 & -0.8660 & 0 & 0 & 0 & 0 & 0 \\ 0.7071 & 0 & 1 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0\,\mathrm{p} \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -0.8660 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0.7071 & 0.5 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -0.7071 & 0.8660 \\ 0 & 0 & 0 & 0 & 0.8660 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.5 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.7071 & 0 \end{bmatrix}$$

S is a diagonal matrix with values (from upper left to lower right) of

4255, 6000, 6000, 3670, 3000,
3670, 6000, 6000, 4255, 3000.

(These quantities are the values of aE/L, where a is the cross-sectional area of a member, E is the Young's modulus for the material, and L is the length.)

Solve the system of equations to determine the values of f for each of three loading vectors:

$$P_1 = [0, -1000, 0, 0, 500, 0, 0, -500, 0]^T,$$
$$P_2 = [1000, 0, 0, -500, 0, 1000, 0, -500, 0]^T,$$
$$P_3 = [0, 0, 0, -500, 0, 0, 0, -500, 0]^T.$$

APP8. For turbulent flow of fluids in an interconnected network (see Fig. 2.4) the flow rate V from one node to another is about proportional to the square root of the difference in pressures at the nodes. (Thus, fluid flow differs from flow of electrical current in a network in that nonlinear equations result.) For the conduits in Figure 2.4, find the pressure at each node. The values of b represent conductance factors in the relation $v_{ij} = b_{ij}(p_i - p_j)^{1/2}$.

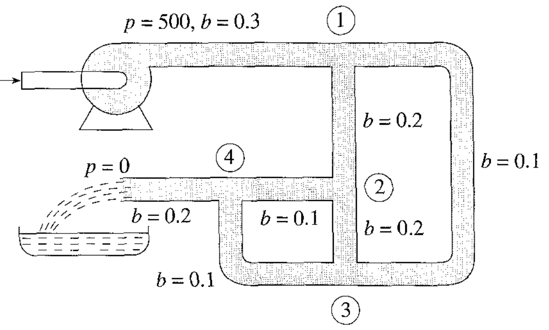


Figure 2.4

These equations can be set up for the pressures at each node:

$$\text{At node 1: } 0.3\sqrt{500 - p_1} = 0.2\sqrt{p_1 - p_2} + 0.1\sqrt{p_1 - p_3};$$

$$\text{node 2: } 0.2\sqrt{p_1 - p_2} = 0.1\sqrt{p_2 - p_4} + 0.2\sqrt{p_2 - p_3};$$

$$\text{node 3: } 0.1\sqrt{p_1 - p_3} + 0.2\sqrt{p_2 - p_3} = 0.1\sqrt{p_3 - p_4};$$

$$\text{node 4: } 0.1\sqrt{p_2 - p_4} + 0.1\sqrt{p_3 - p_4} = 0.2\sqrt{p_4 - 0}.$$

APP9. a. The elements in the matrix equation $Ax = B$ may be complex valued. Write a program to do Gaussian elimination in some computer language that permits complex values and solve a few examples. How will you determine the proper pivot rows in this program? Does the coefficient matrix have an inverse? If so, multiply this inverse by the original coefficient matrix but, before doing this, try to predict the result.

b. It is not necessary to use complex arithmetic to solve a system that has complex-valued elements. How can this be done? Solve the examples that you used in part (a) in this way. You should get the same solutions; do you?

APP10. Electrical circuits always have some capacitance and inductance in addition to resistance. Suppose that a 500 μF capacitor is added to the network of APP3 between nodes 1 and 2 and a 4 mH inductance is added between nodes 5 and 6. Of course, if the voltage source E is a direct current source, no current will flow after the capacitance becomes saturated, but if E is an alternating voltage source, there will be continuous (though fluctuating) current in the network. Set up the equations that can be solved for the voltages at the nodes and the currents in each branch of the network. You may need to consult a reference to handle this mixture of resistors, capacitors, and inductance.

APP11. We have shown how a tridiagonal system is especially advantageous in that it can be solved with fewer arithmetic operations than a full $n \times n$ system. A banded matrix is similarly advantageous, and this is particularly true if the coefficient matrix is symmetric. What are the number of multiplies and divides for a symmetrical system of n equations that has m elements to the right and to the left of the diagonal? Your answer should be expressed in terms of n and m.

APP12. It has been claimed that the National Weather Service uses extremely large sets of equations to forecast the weather. Do research to see if this is true and if these equations are linear or nonlinear. There are several models in use. Five of these are

1. NGM (Nested Grid Model—also called RAFS or Regional Analysis Forecast System).
2. ETA-forecast out to 48 hours.
3. MESO-ETA—forecast to 33 hours.
4. AVN-aviation model to 72 hours.
5. MRF—medium range forecast.

You may find the answer to this question from NWS/CIO (the Office of the Chief Information Officer); the Internet may provide a link to this office. If this is a group project, one member might send an inquiry to:

National Weather Service, NOAA
1325 West-West Highway
Silver Spring, MD 20910

APP13. MATLAB has commands that give you quantitative information on sparse matrices. A tridiagonal matrix is sparse if there are many equations. Generate a large tridiagonal matrix in MATLAB, name it A, and then use these commands to investigate it. What is the information given by each command?

```
nnz (A)
nonzeros (A)
nzmax (A)
spy(A)
[i, j, s) = find (A);
   [m, n) = size (A);
     B = sparse (i, j, s, m, n)
```

APP14. Can Gaussian elimination be used to solve a system where there are inequalities in addition to equalities? Try to do it with this small system. (4, 0) is a solution; what other points are a solution? (Chapter 7 discusses this type of problem in detail.) A graphical solution is easy.

$$\begin{aligned} 5x_1 - 3x_2 &\geq 12 \\ 2x_1 + 4x_2 &\leq 15 \\ x_1 + 3x_2 &= 4 \end{aligned}$$