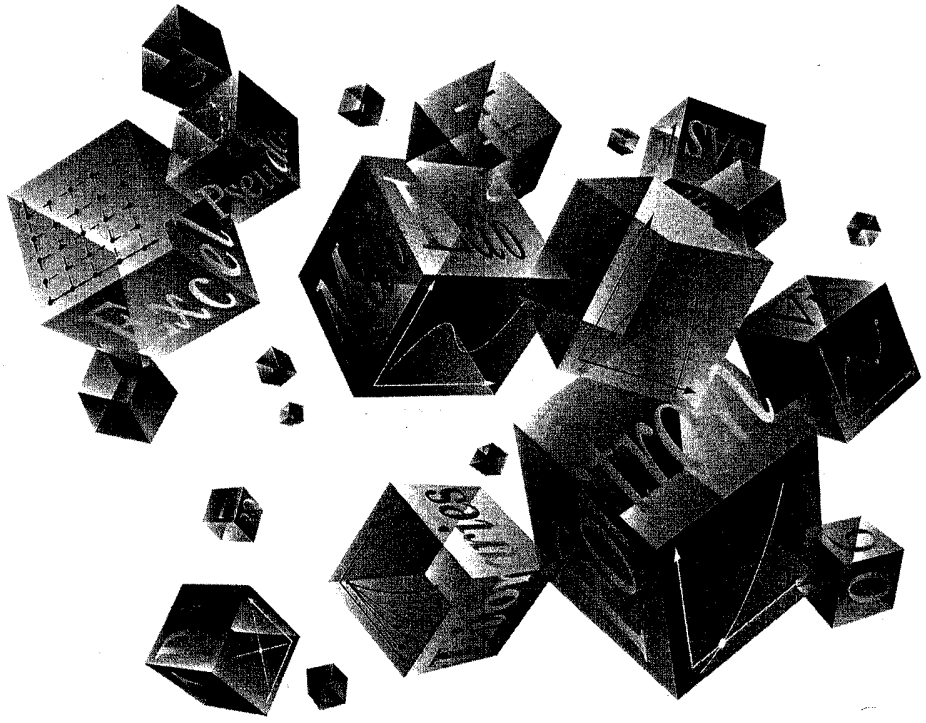
Steven C. Chapra
Berger Chair in Computing and Engineering
Tufts University

Raymond P. Canale
Professor Emeritus of Civil Engineering
University of Michigan

Numerical Methods for Engineers

With Software and Programming Applications

Fourth Edition



Mc Graw Hill

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

CHAPTER 9
Gauss Elimination

This chapter deals with simultaneous linear algebraic equations that can be represented generally as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdot \qquad\qquad \cdot & \\ \cdot \qquad\qquad \cdot & \\ \cdot \qquad\qquad \cdot & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{9.1}$$

where the a's are constant coefficients and the b's are constants.

The technique described in this chapter is called *Gauss elimination* because it involves combining equations to eliminate unknowns. Although it is one of the earliest methods for solving simultaneous equations, it remains among the most important algorithms in use today and is the basis for linear equation solving on many popular software packages.

9.1 SOLVING SMALL NUMBERS OF EQUATIONS

Before proceeding to the computer methods, we will describe several methods that are appropriate for solving small ($n \leq 3$) sets of simultaneous equations and that do not require a computer. These are the graphical method, Cramer's rule, and the elimination of unknowns.

9.1.1 The Graphical Method

A graphical solution is obtainable for two equations by plotting them on Cartesian coordinates with one axis corresponding to x_1 and the other to x_2. Because we are dealing with linear systems, each equation is a straight line. This can be easily illustrated for the general equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned}$$

Both equations can be solved for x_2:

$$x_2 = -\left(\frac{a_{11}}{a_{12}}\right)x_1 + \frac{b_1}{a_{12}}$$

$$x_2 = -\left(\frac{a_{21}}{a_{22}}\right)x_1 + \frac{b_2}{a_{22}}$$

Thus, the equations are now in the form of straight lines; that is, $x_2 = (\text{slope})\, x_1 + \text{intercept}$. These lines can be graphed on Cartesian coordinates with x_2 as the ordinate and x_1 as the abscissa. The values of x_1 and x_2 at the intersection of the lines represent the solution.

EXAMPLE 9.1 The Graphical Method for Two Equations

Problem Statement. Use the graphical method to solve

$$3x_1 + 2x_2 = 18 \quad \text{(E9.1.1)}$$

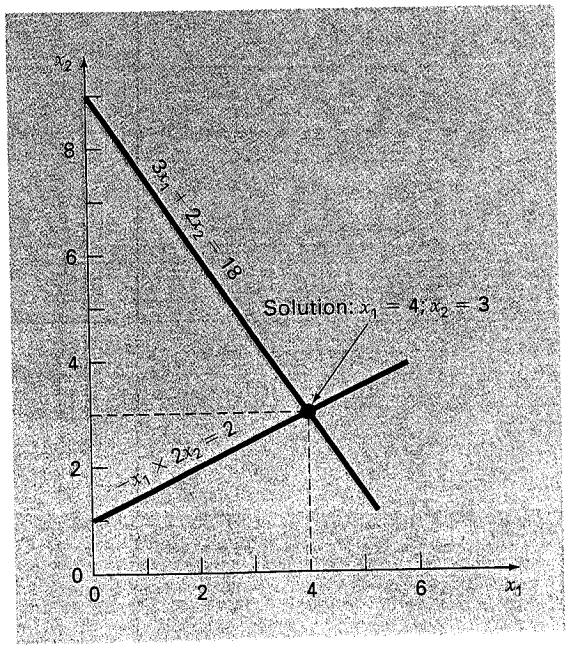
$$-x_1 + 2x_2 = 2 \quad \text{(E9.1.2)}$$

Solution. Let x_1 be the abscissa. Solve Eq (E9.1.1) for x_2:

$$x_2 = -\frac{3}{2}x_1 + 9$$

which, when plotted on Fig. 9.1, is a straight line with an intercept of 9 and a slope of $-3/2$.

FIGURE 9.1
Graphical solution of a set of two simultaneous linear algebraic equations. The intersection of the lines represents the solution.



Equation (E9.1.2) can also be solved for x_2:

$$x_2 = \frac{1}{2}x_1 + 1$$

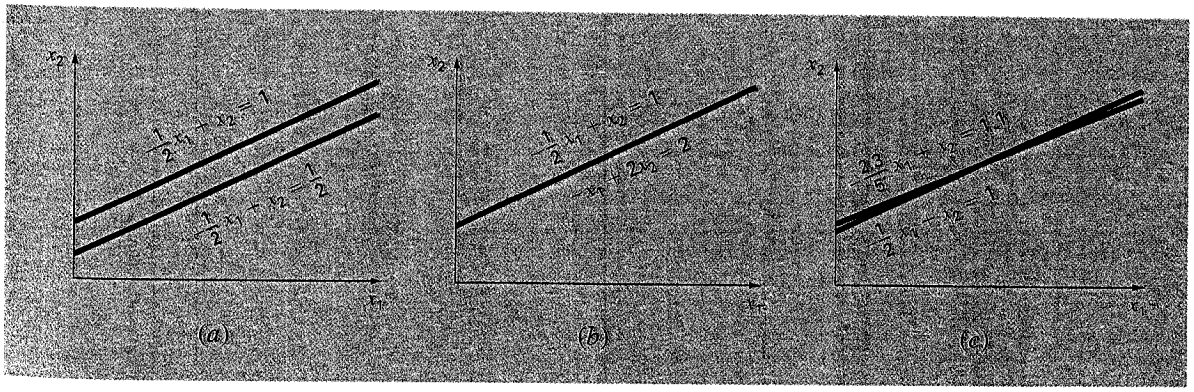
which is also plotted on Fig. 9.1. The solution is the intersection of the two lines at $x_1 = 4$ and $x_2 = 3$. This result can be checked by substituting these values into the original equations to yield

$$3(4) + 2(3) = 18$$
$$-(4) + 2(3) = 2$$

Thus, the results are equivalent to the right-hand sides of the original equations.

For three simultaneous equations, each equation would be represented by a plane in a three-dimensional coordinate system. The point where the three planes intersect would represent the solution. Beyond three equations, graphical methods break down and, consequently, have little practical value for solving simultaneous equations. However, they sometimes prove useful in visualizing properties of the solutions. For example, Fig. 9.2 depicts three cases that can pose problems when solving sets of linear equations. Figure 9.2*a* shows the case where the two equations represent parallel lines. For such situations, there is no solution because the lines never cross. Figure 9.2*b* depicts the case where the two lines are coincident. For such situations there is an infinite number of solutions. Both types of systems are said to be *singular*. In addition, systems that are very close to being singular (Fig. 9.2*c*) can also cause problems. These systems are said to be *ill-conditioned*. Graphically, this corresponds to the fact that it is difficult to identify the exact point at which the lines intersect. Ill-conditioned systems will also pose problems when they are encountered during the numerical solution of linear equations. This is because they will be extremely sensitive to round-off error (recall Sec. 4.2.3).

FIGURE 9.2
Graphical depiction of singular and ill-conditioned systems: (*a*) no solution, (*b*) infinite solutions, and (*c*) ill-conditioned system where the slopes are so close that the point of intersection is difficult to detect visually.



9.1.2 Determinants and Cramer's Rule

Cramer's rule is another solution technique that is best suited to small numbers of equations. Before describing this method, we will briefly introduce the concept of the determinant, which is used to implement Cramer's rule. In addition, the determinant has relevance to the evaluation of the ill-conditioning of a matrix.

Determinants. The determinant can be illustrated for a set of three equations:

$$[A]\{X\} = \{B\}$$

where $[A]$ is the coefficient matrix:

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The determinant D of this system is formed from the coefficients of the equation, as in

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \tag{9.2}$$

Although the determinant D and the coefficient matrix $[A]$ are composed of the same elements, they are completely different mathematical concepts. That is why they are distinguished visually by using brackets to enclose the matrix and straight lines to enclose the determinant. In contrast to a matrix, the determinant is a single number. For example, the value of the second-order determinant

$$D = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

is calculated by

$$D = a_{11}a_{22} - a_{12}a_{21} \tag{9.3}$$

For the third-order case [Eq. (9.2)], a single numerical value for the determinant can be computed as

$$D = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} \tag{9.4}$$

where the 2 by 2 determinants are called ***minors.***

EXAMPLE 9.2 Determinants

Problem Statement. Compute values for the determinants of the systems represented in Figs. 9.1 and 9.2.

Solution. For Fig. 9.1:

$$D = \begin{vmatrix} 3 & 2 \\ -1 & 2 \end{vmatrix} = 3(2) - 2(-1) = 8$$

For Fig. 9.2*a*:

$$D = \begin{vmatrix} -1/2 & 1 \\ -1/2 & 1 \end{vmatrix} = \frac{-1}{2}(1) - 1\left(\frac{-1}{2}\right) = 0$$

For Fig. 9.2*b*:

$$D = \begin{vmatrix} -1/2 & 1 \\ -1 & 2 \end{vmatrix} = \frac{-1}{2}(2) - 1(-1) = 0$$

For Fig. 9.2*c*:

$$D = \begin{vmatrix} -1/2 & 1 \\ -2.3/5 & 1 \end{vmatrix} = \frac{-1}{2}(1) - 1\left(\frac{-2.3}{5}\right) = -0.04$$

In the foregoing example, the singular systems had zero determinants. Additionally, the results suggest that the system that is almost singular (Fig. 9.2*c*) has a determinant that is close to zero. These ideas will be pursued further in our subsequent discussion of ill-conditioning (Sec. 9.3.3).

Cramer's Rule. This rule states that each unknown in a system of linear algebraic equations may be expressed as a fraction of two determinants with denominator D and with the numerator obtained from D by replacing the column of coefficients of the unknown in question by the constants $b_1, b_2, \ldots, b_n$. For example, x_1 would be computed as

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{D} \tag{9.5}$$

EXAMPLE 9.3 Cramer's Rule

Problem Statement. Use Cramer's rule to solve

$$0.3x_1 + 0.52x_2 + x_3 = -0.01$$
$$0.5x_1 + x_2 + 1.9x_3 = 0.67$$
$$0.1x_1 + 0.3x_2 + 0.5x_3 = -0.44$$

Solution. The determinant D can be written as [Eq. (9.2)]

$$D = \begin{vmatrix} 0.3 & 0.52 & 1 \\ 0.5 & 1 & 1.9 \\ 0.1 & 0.3 & 0.5 \end{vmatrix}$$

The minors are [Eq. (9.3)]

$$A_1 = \begin{vmatrix} 1 & 1.9 \\ 0.3 & 0.5 \end{vmatrix} = 1(0.5) - 1.9(0.3) = -0.07$$

$$A_2 = \begin{vmatrix} 0.5 & 1.9 \\ 0.1 & 0.5 \end{vmatrix} = 0.5(0.5) - 1.9(0.1) = 0.06$$

$$A_3 = \begin{vmatrix} 0.5 & 1 \\ 0.1 & 0.3 \end{vmatrix} = 0.5(0.3) - 1(0.1) = 0.05$$

These can be used to evaluate the determinant, as in [Eq. (9.4)]

$$D = 0.3(-0.07) - 0.52(0.06) + 1(0.05) = -0.0022$$

Applying Eq. (9.5), the solution is

$$x_1 = \frac{\begin{vmatrix} -0.01 & 0.52 & 1 \\ 0.67 & 1 & 1.9 \\ -0.44 & 0.3 & 0.5 \end{vmatrix}}{-0.0022} = \frac{0.03278}{-0.0022} = -14.9$$

$$x_2 = \frac{\begin{vmatrix} 0.3 & -0.01 & 1 \\ 0.5 & 0.67 & 1.9 \\ 0.1 & -0.44 & 0.5 \end{vmatrix}}{-0.0022} = \frac{0.0649}{-0.0022} = -29.5$$

$$x_3 = \frac{\begin{vmatrix} 0.3 & 0.52 & -0.01 \\ 0.5 & 1 & 0.67 \\ 0.1 & 0.3 & -0.44 \end{vmatrix}}{-0.0022} = \frac{-0.04356}{-0.0022} = 19.8$$

For more than three equations, Cramer's rule becomes impractical because, as the number of equations increases, the determinants are time consuming to evaluate by hand (or by computer). Consequently, more efficient alternatives are used. Some of these alternatives are based on the last noncomputer solution technique covered in the next section—the elimination of unknowns.

9.1.3 The Elimination of Unknowns

The elimination of unknowns by combining equations is an algebraic approach that can be illustrated for a set of two equations:

$$a_{11}x_1 + a_{12}x_2 = b_1 \tag{9.6}$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{9.7}$$

The basic strategy is to multiply the equations by constants so that one of the unknowns will be eliminated when the two equations are combined. The result is a single equation that can be solved for the remaining unknown. This value can then be substituted into either of the original equations to compute the other variable.

For example, Eq. (9.6) might be multiplied by a_{21} and Eq. (9.7) by a_{11} to give

$$a_{11}a_{21}x_1 + a_{12}a_{21}x_2 = b_1a_{21} \tag{9.8}$$

$$a_{21}a_{11}x_1 + a_{22}a_{11}x_2 = b_2a_{11} \tag{9.9}$$

Subtracting Eq. (9.8) from Eq. (9.9) will, therefore, eliminate the x_1 term from the equations to yield

$$a_{22}a_{11}x_2 - a_{12}a_{21}x_2 = b_2a_{11} - b_1a_{21}$$

which can be solved for

$$x_2 = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{12}a_{21}} \tag{9.10}$$

Equation (9.10) can then be substituted into Eq. (9.6), which can be solved for

$$x_1 = \frac{a_{22}b_1 - a_{12}b_2}{a_{11}a_{22} - a_{12}a_{21}} \tag{9.11}$$

Notice that Eqs. (9.10) and (9.11) follow directly from Cramer's rule, which states

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} = \frac{b_1a_{22} - a_{12}b_2}{a_{11}a_{22} - a_{12}a_{21}}$$

$$x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} = \frac{a_{11}b_2 - b_1a_{21}}{a_{11}a_{22} - a_{12}a_{21}}$$

EXAMPLE 9.4 Elimination of Unknowns

Problem Statement. Use the elimination of unknowns to solve (recall Example 9.1)

$$3x_1 + 2x_2 = 18$$
$$-x_1 + 2x_2 = 2$$

Solution. Using Eqs. (9.11) and (9.10),

$$x_1 = \frac{2(18) - 2(2)}{3(2) - 2(-1)} = 4$$

$$x_2 = \frac{3(2) - (-1)18}{3(2) - 2(-1)} = 3$$

which is consistent with our graphical solution (Fig. 9.1).

The elimination of unknowns can be extended to systems with more than two or three equations. However, the numerous calculations that are required for larger systems make the method extremely tedious to implement by hand. However, as described in the next section, the technique can be formalized and readily programmed for the computer.

9.2 NAIVE GAUSS ELIMINATION

In the previous section, the elimination of unknowns was used to solve a pair of simultaneous equations. The procedure consisted of two steps:

1. The equations were manipulated to eliminate one of the unknowns from the equations. The result of this *elimination* step was that we had one equation with one unknown.
2. Consequently, this equation could be solved directly and the result *back-substituted* into one of the original equations to solve for the remaining unknown.

This basic approach can be extended to large sets of equations by developing a systematic scheme or algorithm to eliminate unknowns and to back-substitute. *Gauss elimination* is the most basic of these schemes.

This section includes the systematic techniques for forward elimination and back substitution that comprise Gauss elimination. Although these techniques are ideally suited for implementation on computers, some modifications will be required to obtain a reliable algorithm. In particular, the computer program must avoid division by zero. The following method is called *"naive" Gauss elimination* because it does not avoid this problem. Subsequent sections will deal with the additional features required for an effective computer program.

The approach is designed to solve a general set of n equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \tag{9.12a}$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \tag{9.12b}$$

$$\vdots \qquad \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \tag{9.12c}$$

As was the case with the solution of two equations, the technique for n equations consists of two phases: elimination of unknowns and solution through back substitution.

Forward Elimination of Unknowns. The first phase is designed to reduce the set of equations to an upper triangular system (Fig. 9.3). The initial step will be to eliminate the first unknown, x_1, from the second through the nth equations. To do this, multiply Eq. (9.12a) by a_{21}/a_{11} to give

$$a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \cdots + \frac{a_{21}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}b_1 \tag{9.13}$$

Now, this equation can be subtracted from Eq. (9.12b) to give

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \cdots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{1n}\right)x_n = b_2 - \frac{a_{21}}{a_{11}}b_1$$

or

$$a'_{22}x_2 + \cdots + a'_{2n}x_n = b'_2$$

where the prime indicates that the elements have been changed from their original values.

The procedure is then repeated for the remaining equations. For instance, Eq. (9.12a) can be multiplied by a_{31}/a_{11} and the result subtracted from the third equation. Repeating

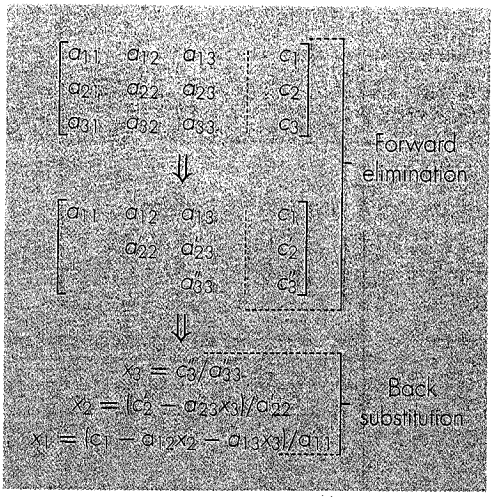


FIGURE 9.3
The two phases of Gauss elimination: forward elimination and back substitution. The primes indicate the number of times that the coefficients and constants have been modified.

the procedure for the remaining equations results in the following modified system:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \tag{9.14a}$$
$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2 \tag{9.14b}$$
$$a'_{32}x_2 + a'_{33}x_3 + \cdots + a'_{3n}x_n = b'_3 \tag{9.14c}$$
$$\vdots \qquad \vdots$$
$$a'_{n2}x_2 + a'_{n3}x_3 + \cdots + a'_{nn}x_n = b'_n \tag{9.14d}$$

For the foregoing steps, Eq. (9.12*a*) is called the *pivot equation* and a_{11} is called the *pivot coefficient* or *element*. Note that the process of multiplying the first row by a_{21}/a_{11} is equivalent to dividing it by a_{11} and multiplying it by a_{21}. Sometimes the division operation is referred to as normalization. We make this distinction because a zero pivot element can interfere with normalization by causing a division by zero. We will return to this important issue after we complete our description of naive Gauss elimination.

Now repeat the above to eliminate the second unknown from Eq. (9.14*c*) through (9.14*d*). To do this multiply Eq. (9.14*b*) by a'_{32}/a'_{22} and subtract the result from Eq. (9.14*c*). Perform a similar elimination for the remaining equations to yield

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1$$
$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2$$
$$a''_{33}x_3 + \cdots + a''_{3n}x_n = b''_3$$
$$\vdots \qquad \vdots$$
$$a''_{n3}x_3 + \cdots + a''_{nn}x_n = b''_n$$

where the double prime indicates that the elements have been modified twice.

The procedure can be continued using the remaining pivot equations. The final manipulation in the sequence is to use the $(n-1)$th equation to eliminate the x_{n-1} term from the nth equation. At this point, the system will have been transformed to an upper triangular system (recall Box PT3.1):

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \tag{9.15a}$$

$$a'_{22}x_2 + a'_{23}x_3 + \cdots + a'_{2n}x_n = b'_2 \tag{9.15b}$$

$$a''_{33}x_3 + \cdots + a''_{3n}x_n = b''_3 \tag{9.15c}$$

$$\vdots \qquad \vdots$$

$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)} \tag{9.15d}$$

Pseudocode to implement forward elimination is presented in Fig. 9.4*a*. Notice that three nested loops provide a concise representation of the process. The outer loop moves down the matrix from one pivot row to the next. The middle loop moves below the pivot row to each of the subsequent rows where elimination is to take place. Finally, the innermost loop progresses across the columns to eliminate or transform the elements of a particular row.

Back Substitution. Equation (9.15*d*) can now be solved for x_n:

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \tag{9.16}$$

This result can be back-substituted into the $(n-1)$th equation to solve for x_{n-1}. The procedure, which is repeated to evaluate the remaining x's, can be represented by the following

FIGURE 9.4
Pseudocode to perform (*a*) forward elimination and (*b*) back substitution.

(*a*)

```
DO k = 1, n - 1
 DO i = k + 1, n
  factor = a(i,k) / a(k,k)
  DO j = k + 1 to n
   a(i,j) = a(i,j) - factor * a(k,j)
  END DO
  b(i) = b(i) - factor * b(k)
 END DO
END DO
```

(*b*)

```
x(n) = b(n) / a(n,n)
DO i = n - 1, 1, -1
 sum = 0
 DO j = i + 1, n
  sum = sum + a(i,j) * x(j)
 END DO
 x(i) = (b(i) - sum) / a(i,i)
END DO
```

formula:

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \qquad \text{for } i = n-1, n-2, \ldots, 1 \tag{9.17}$$

Pseudocode to implement Eqs. (9.16) and (9.17) is presented in Fig. 9.4*b*. Notice the similarity between this pseudocode and that in Fig. PT3.4 for matrix multiplication. As with Fig. PT3.4, a temporary variable, *sum*, is used to accumulate the summation from Eq. (9.17). This results in a somewhat faster execution time than if the summation were accumulated in b_i. More importantly, it allows efficient improvement in precision if the variable, *sum*, is declared in double precision.

EXAMPLE 9.5 Naive Gauss Elimination

Problem Statement. Use Gauss elimination to solve

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85 \tag{E9.5.1}$$
$$0.1x_1 + 7x_2 - 0.3x_3 = -19.3 \tag{E9.5.2}$$
$$0.3x_1 - 0.2x_2 + 10x_3 = 71.4 \tag{E9.5.3}$$

Carry six significant figures during the computation.

Solution. The first part of the procedure is forward elimination. Multiply Eq. (E9.5.1) by (0.1)/3 and subtract the result from Eq. (E9.5.2) to give

$$7.00333x_2 - 0.293333x_3 = -19.5617$$

Then multiply Eq. (E9.5.1) by (0.3)/3 and subtract it from Eq. (E9.5.3) to eliminate x_1. After these operations, the set of equations is

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85 \tag{E9.5.4}$$
$$7.00333x_2 - 0.293333x_3 = -19.5617 \tag{E9.5.5}$$
$$-0.190000x_2 + 10.0200x_3 = 70.6150 \tag{E9.5.6}$$

To complete the forward elimination, x_2 must be removed from Eq. (E9.5.6). To accomplish this, multiply Eq. (E9.5.5) by −0.190000/7.00333 and subtract the result from Eq. (E9.5.6). This eliminates x_2 from the third equation and reduces the system to an upper triangular form, as in

$$3x_1 - 0.1x_2 - 0.2x_3 = 7.85 \tag{E9.5.7}$$
$$7.00333x_2 - 0.293333x_3 = -19.5617 \tag{E9.5.8}$$
$$10.0200x_3 = 70.0843 \tag{E9.5.9}$$

We can now solve these equations by back substitution. First, Eq. (E9.5.9) can be solved for

$$x_3 = \frac{70.0843}{10.0200} = 7.00003 \tag{E9.5.10}$$

This result can be back-substituted into Eq. (E9.5.8):

$$7.00333x_2 - 0.293333(7.00003) = -19.5617$$

which can be solved for

$$x_2 = \frac{-19.5617 + 0.293333(7.00003)}{7.00333} = -2.50000 \tag{E9.5.11}$$

Finally, Eqs. (E9.5.10) and (E9.5.11) can be substituted into Eq. (E9.5.4):

$$3x_1 - 0.1(-2.50000) - 0.2(7.00003) = 7.85$$

which can be solved for

$$x_1 = \frac{7.85 + 0.1(-2.50000) + 0.2(7.00003)}{3} = 3.00000$$

Although there is a slight round-off error in Eq. (E9.5.10), the results are very close to the exact solution of $x_1 = 3$, $x_2 = -2.5$, and $x_3 = 7$. This can be verified by substituting the results into the original equation set

$$3(3) - 0.1(-2.5) - 0.2(7.00003) = 7.84999 \cong 7.85$$
$$0.1(3) + 7(-2.5) - 0.3(7.00003) = -19.3000 = -19.3$$
$$0.3(3) - 0.2(-2.5) + 10(7.00003) = 71.4003 \cong 71.4$$

9.2.1 Operation Counting

The execution time of Gauss elimination depends on the amount of *fl*oating-point *op*erations (or *FLOP*s) involved in the algorithm. In general, the time consumed to perform multiplications and divisions is about the same, and is larger than for additions and subtractions.

Before analyzing naive Gauss elimination, we will first define some quantities that facilitate operation counting:

$$\sum_{i=1}^{m} cf(i) = c\sum_{i=1}^{m} f(i) \qquad \sum_{i=1}^{m} f(i) + g(i) = \sum_{i=1}^{m} f(i) + \sum_{i=1}^{m} g(i) \tag{9.18a,b}$$

$$\sum_{i=1}^{m} 1 = 1 + 1 + \cdots + 1 = m \qquad \sum_{i=k}^{m} 1 = m - k + 1 \tag{9.18c,d}$$

$$\sum_{i=1}^{m} i = 1 + 2 + 3 + \cdots + m = \frac{m(m+1)}{2} = \frac{m^2}{2} + O(m) \tag{9.18e}$$

$$\sum_{i=1}^{m} i^2 = 1^2 + 2^2 + 3^2 + \cdots + m^2 = \frac{m(m+1)(2m+1)}{6} = \frac{m^3}{3} + O(m^2) \tag{9.18f}$$

where $O(m^n)$ means "terms of order m^n and lower."

Now let us examine the naive Gauss elimination algorithm in detail. As in Fig. 9.4*a*, we will first count the multiplication/division FLOPs in the elimination stage. On the first pass

through the outer loop, $k = 1$. Therefore, the limits on the middle loop are from $i = 2$ to n. According to Eq. (9.18*d*), this means that the number of iterations of the middle loop will be

$$\sum_{i=2}^{n} 1 = n - 2 + 1 = n - 1 \tag{9.19}$$

Now for every one of these iterations, there is 1 division to define $factor = a_{i,k}/a_{k,k}$. The interior loop then performs a single multiplication ($factor \cdot a_{k,j}$) for each iteration from $j = 2$ to n. Finally, there is one additional multiplication of the right-hand-side value ($factor \cdot b_k$). Thus, for every iteration of the middle loop, the number of multiplications is

$$1 + [n - 2 + 1] + 1 = 1 + n \tag{9.20}$$

The total for the first pass through the outer loop is therefore obtained by multiplying Eq. (9.19) by (9.20) to give $[n - 1](1 + n)$.

A similar procedure can be used to estimate the multiply/divide FLOPs for the subsequent iterations of the outer loop. These can be summarized as

Outer Loop k	Middle Loop i	FLOPs
1	2, n	$[n - 1](1 + n)$
2	3, n	$[n - 2](n)$
.	.	.
.	.	.
.	.	.
k	$k + 1$, n	$[n - k](n + 2 - k)$
.	.	.
.	.	.
.	.	.
$n - 1$	n, n	[1] (3)

Therefore, the total FLOPs for elimination can be computed as

$$\sum_{k=1}^{n-1} [n - k](n + 2 - k) = \sum_{k=1}^{n-1} \{n(n + 2) - k(2n + 2) + k^2\} \tag{9.21}$$

Applying some of the relationships from Eq. (9.18) yields

$$\{n^3 + O(n^2)\} - \{n^3 + O(n^2)\} + \left\{\frac{2n^3}{6} + O(n^2)\right\} = \frac{n^3}{3} + O(n^2) \tag{9.22}$$

Thus, the total number of multiply/divide FLOPs is equal to $n^3/3$ plus an additional component proportional to terms of order n^2 and lower. The result is written in this way because as n gets large, the $O(n^2)$ terms become negligible. We are therefore justified in concluding that for large n, the effort involved in forward elimination is $n^3/3$.

Because only a single loop is used, back substitution is much simpler to evaluate. The number of multiplication FLOPs can be directly taken from Eq. (9.18*e*),

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n + 1)}{2} = \frac{n^2}{2} + O(n)$$

TABLE 9.1 Number of FLOPs for naive Gauss elimination.

n	Elimination	Back Substitution	Total FLOPs	$n^3/3$	Percent Due to Elimination
10	375	55	430	333	87.21%
100	338250	5050	343300	333333	98.53%
1000	3.34E+08	500500	3.34×10^8	3.33×10^8	99.85%

Thus, the total effort in naive Gauss elimination can be represented as

$$\underbrace{\frac{n^3}{3}}_{\text{Forward elimination}} + O(n^2) + \underbrace{\frac{n^2}{2}}_{\text{Back substitution}} + O(n) \xrightarrow{\text{as } n \text{ increases}} \frac{n^3}{3} + O(n^2) \tag{9.23}$$

Two useful general conclusions can be drawn from this analysis:

1. As the system gets larger, the computation time increases greatly. As in Table 9.1, the amount of FLOPs increases nearly 3 orders of magnitude for every order of magnitude increase in the dimension.
2. Most of the effort is incurred in the elimination step. Thus, efforts to make the method more efficient should probably focus on this step.

Throughout the remainder of this part, we will make operation counts to compare alternative solution methods. Although we may not go into the detail of the above analysis, the same general approach will be employed.

9.3 PITFALLS OF ELIMINATION METHODS

Whereas there are many systems of equations that can be solved with naive Gauss elimination, there are some pitfalls that must be explored before writing a general computer program to implement the method. Although the following material relates directly to naive Gauss elimination, the information is relevant for other elimination techniques as well.

9.3.1 Division by Zero

The primary reason that the foregoing technique is called "naive" is that during both the elimination and the back-substitution phases, it is possible that a division by zero can occur. For example, if we use naive Gauss elimination to solve

$$\begin{aligned} 2x_2 + 3x_3 &= 8 \\ 4x_1 + 6x_2 + 7x_3 &= -3 \\ 2x_1 + x_2 + 6x_3 &= 5 \end{aligned}$$

the normalization of the first row would involve division by $a_{11} = 0$. Problems also can arise when a coefficient is very close to zero. The technique of *pivoting* has been developed to partially avoid these problems. It will be described in Sec. 9.4.2.

9.3.2 Round-Off Errors

Even though the solution in Example 9.5 was close to the true answer, there was a slight discrepancy in the result for x_3 [Eq. (E9.5.10)]. This discrepancy, which amounted to a relative error of -0.00043 percent, was due to our use of six significant figures during the computation. If we had used more significant figures, the error in the results would be reduced further. If we had used fractions instead of decimals (and consequently avoided round-off altogether), the answers would have been exact. However, because computers carry only a limited number of significant figures (recall Sec. 3.4.1), round-off errors can occur and must be considered when evaluating the results.

The problem of round-off error can become particularly important when large numbers of equations are to be solved. This is due to the fact that every result is dependent on previous results. Consequently, an error in the early steps will tend to propagate—that is, it will cause errors in subsequent steps.

Specifying the system size where round-off error becomes significant is complicated by the fact that the type of computer and the properties of the equations are determining factors. A rough rule of thumb is that round-off error may be important when dealing with 100 or more equations. In any event, you should always substitute your answers back into the original equations to check whether a substantial error has occurred. However, as discussed below, the magnitudes of the coefficients themselves can influence whether such an error check ensures a reliable result.

9.3.3 Ill-Conditioned Systems

The adequacy of the solution depends on the condition of the system. In Sec. 9.1.1, a graphical depiction of system condition was developed. As discussed in Sec. 4.2.3, *well-conditioned systems* are those where a small change in one or more of the coefficients results in a similar small change in the solution. *Ill-conditioned systems* are those where small changes in coefficients result in large changes in the solution. An alternative interpretation of ill-conditioning is that a wide range of answers can approximately satisfy the equations. Because round-off errors can induce small changes in the coefficients, these artificial changes can lead to large solution errors for ill-conditioned systems, as illustrated in the following example.

EXAMPLE 9.6 Ill-Conditioned Systems

Problem Statement. Solve the following system:

$$x_1 + 2x_2 = 10 \tag{E9.6.1}$$

$$1.1x_1 + 2x_2 = 10.4 \tag{E9.6.2}$$

Then, solve it again, but with the coefficient of x_1 in the second equation modified slightly to 1.05.

Solution. Using Eqs. (9.10) and (9.11), the solution is

$$x_1 = \frac{2(10) - 2(10.4)}{1(2) - 2(1.1)} = 4$$

$$x_2 = \frac{1(10.4) - 1.1(10)}{1(2) - 2(1.1)} = 3$$

However, with the slight change of the coefficient a_{21} from 1.1 to 1.05, the result is changed dramatically to

$$x_1 = \frac{2(10) - 2(10.4)}{1(2) - 2(1.05)} = 8$$

$$x_2 = \frac{1(10.4) - 1.1(10)}{1(2) - 2(1.05)} = 1$$

Notice that the primary reason for the discrepancy between the two results is that the denominator represents the difference of two almost-equal numbers. As illustrated previously in Sec. 3.4.2, such differences are highly sensitive to slight variations in the numbers being manipulated.

At this point, you might suggest that substitution of the results into the original equations would alert you to the problem. Unfortunately, for ill-conditioned systems this is often not the case. Substitution of the erroneous values of $x_1 = 8$ and $x_2 = 1$ into Eqs. (E9.6.1) and (E9.6.2) yields

$$8 + 2(1) = 10 = 10$$

$$1.1(8) + 2(1) = 10.8 \cong 10.4$$

Therefore, although $x_1 = 8$ and $x_2 = 1$ is not the true solution to the original problem, the error check is close enough to possibly mislead you into believing that your solutions are adequate.

As was done previously in the section on graphical methods, a visual representative of ill-conditioning can be developed by plotting Eqs. (E9.6.1) and (E9.6.2) (recall Fig. 9.2). Because the slopes of the lines are almost equal, it is visually difficult to see exactly where they intersect. This visual difficulty is reflected quantitatively in the nebulous results of Example 9.6. We can mathematically characterize this situation by writing the two equations in general form:

$$a_{11}x_1 + a_{12}x_2 = b_1 \tag{9.24}$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \tag{9.25}$$

Dividing Eq. (9.24) by a_{12} and Eq. (9.25) by a_{22} and rearranging yields alternative versions that are in the format of straight lines [$x_2 =$ (slope) $x_1 +$ intercept]:

$$x_2 = -\frac{a_{11}}{a_{12}}x_1 + \frac{b_1}{a_{12}}$$

$$x_2 = -\frac{a_{21}}{a_{22}}x_1 + \frac{b_2}{a_{22}}$$

Consequently, if the slopes are nearly equal,

$$\frac{a_{11}}{a_{12}} \cong \frac{a_{21}}{a_{22}}$$

or, cross-multiplying,

$$a_{11}a_{22} \cong a_{12}a_{21}$$

which can be also expressed as

$$a_{11}a_{22} - a_{12}a_{21} \cong 0 \tag{9.26}$$

Now, recalling that $a_{11}a_{22} - a_{12}a_{21}$ is the determinant of a two-dimensional system [Eq. (9.3)], we arrive at the general conclusion that an ill-conditioned system is one with a determinant close to zero. In fact, if the determinant is exactly zero, the two slopes are identical, which connotes either no solution or an infinite number of solutions, as is the case for the singular systems depicted in Fig. 9.2*a* and *b*.

It is difficult to specify how close to zero the determinant must be to indicate ill-conditioning. This is complicated by the fact that the determinant can be changed by multiplying one or more of the equations by a scale factor without changing the solution. Consequently, the determinant is a relative value that is influenced by the magnitude of the coefficients.

EXAMPLE 9.7 Effect of Scale on the Determinant

Problem Statement. Evaluate the determinant of the following systems:

(a) From Example 9.1:

$$3x_1 + 2x_2 = 18 \tag{E9.7.1}$$

$$-x_1 + 2x_2 = 2 \tag{E9.7.2}$$

(b) From Example 9.6:

$$x_1 + 2x_2 = 10 \tag{E9.7.3}$$

$$1.1x_1 + 2x_2 = 10.4 \tag{E9.7.4}$$

(c) Repeat (*b*) but with the equations multiplied by 10.

Solution.

(a) The determinant of Eqs. (E9.7.1) and (E9.7.2), which are well-conditioned, is

$$D = 3(2) - 2(-1) = 8$$

(b) The determinant of Eqs. (E9.7.3) and (E9.7.4), which are ill-conditioned, is

$$D = 1(2) - 2(1.1) = -0.2$$

(c) The results of (*a*) and (*b*) seem to bear out the contention that ill-conditioned systems have near-zero determinants. However, suppose that the ill-conditioned system in (*b*) is multiplied by 10 to give

$$10x_1 + 20x_2 = 100$$

$$11x_1 + 20x_2 = 104$$

The multiplication of an equation by a constant has no effect on its solution. In addition, it is still ill-conditioned. This can be verified by the fact that multiplying by a

constant has no effect on the graphical solution. However, the determinant is dramatically affected:

$$D = 10(20) - 20(11) = -20$$

Not only has it been raised two orders of magnitude, but it is now over twice as large as the determinant of the well-conditioned system in (*a*).

As illustrated by the previous example, the magnitude of the coefficients interjects a scale effect that complicates the relationship between system condition and determinant size. One way to partially circumvent this difficulty is to scale the equations so that the maximum element in any row is equal to 1.

EXAMPLE 9.8 Scaling

Problem Statement. Scale the systems of equations in Example 9.7 to a maximum value of 1 and recompute their determinants.

Solution.

(a) For the well-conditioned system, scaling results in

$$x_1 + 0.667x_2 = 6$$
$$-0.5x_1 + x_2 = 1$$

for which the determinant is

$$D = 1(1) - 0.667(-0.5) = 1.333$$

(b) For the ill-conditioned system, scaling gives

$$0.5x_1 + x_2 = 5$$
$$0.55x_1 + x_2 = 5.2$$

for which the determinant is

$$D = 0.5(1) - 1(0.55) = -0.05$$

(c) For the last case, scaling changes the system to the same form as in (*b*) and the determinant is also −0.05. Thus, the scale effect is removed.

In a previous section (Sec. 9.1.2), we suggested that the determinant is difficult to compute for more than three simultaneous equations. Therefore, it might seem that it does not provide a practical means for evaluating system condition. However, as described in Box 9.1, there is a simple algorithm that results from Gauss elimination that can be used to evaluate the determinant.

Aside from the approach used in the previous example, there are a variety of other ways to evaluate system condition. For example, there are alternative methods for normalizing the elements (see Stark, 1970). In addition, as described in the next chapter (Sec. 10.3), the matrix inverse and matrix norms can be employed to evaluate system condition. Finally, a simple (but time-consuming) test is to modify the coefficients slightly and repeat the

Box 9.1 Determinant Evaluation Using Gauss Elimination

In Sec. 9.1.2, we stated that determinant evaluation by expansion of minors was impractical for large sets of equations. Thus, we concluded that Cramer's rule would be applicable only to small systems. However, as mentioned in Sec. 9.3.3, the determinant has value in assessing system condition. It would, therefore, be useful to have a practical method for computing this quantity.

Fortunately, Gauss elimination provides a simple way to do this. The method is based on the fact that the determinant of a triangular matrix can be simply computed as the product of its diagonal elements:

$$D = a_{11}a_{22}a_{33} \cdots a_{nn} \tag{B9.1.1}$$

The validity of this formulation can be illustrated for a 3 by 3 system:

$$D = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix}$$

where the determinant can be evaluated as [recall Eq. (9.4)]

$$D = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ 0 & a_{33} \end{vmatrix} - a_{12}\begin{vmatrix} 0 & a_{23} \\ 0 & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} 0 & a_{22} \\ 0 & 0 \end{vmatrix}$$

or, by evaluating the minors (that is, the 2 by 2 determinants),

$$D = a_{11}a_{22}a_{33} - a_{12}(0) + a_{13}(0) = a_{11}a_{22}a_{33}$$

Recall that the forward-elimination step of Gauss elimination results in an upper triangular system. Because the value of the determinant is not changed by the forward-elimination process, the determinant can be simply evaluated at the end of this step via

$$D = a_{11}a'_{22}a''_{33} \cdots a^{(n-1)}_{nn} \tag{B9.1.2}$$

where the superscripts signify the number of times that the elements have been modified by the elimination process. Thus, we can capitalize on the effort that has already been expended in reducing the system to triangular form and, in the bargain, come up with a simple estimate of the determinant.

There is a slight modification to the above approach when the program employs partial pivoting (Sec. 9.4.2). For this case, the determinant changes sign every time a row is pivoted. One way to represent this is to modify Eq. (B9.1.2):

$$D = a_{11}a'_{22}a''_{33} \cdots a^{(n-1)}_{nn}(-1)^p \tag{B9.1.3}$$

where p represents the number of times that rows are pivoted. This modification can be incorporated simply into a program; merely keep track of the number of pivots that take place during the course of the computation and then use Eq. (B9.1.3) to evaluate the determinant.

solution. If such modifications lead to drastically different results, the system is likely to be ill-conditioned.

As you might gather from the foregoing discussion, ill-conditioned systems are problematic. Fortunately, most linear algebraic equations derived from engineering-problem settings are naturally well-conditioned. In addition, some of the techniques outlined in Sec. 9.4 help to alleviate the problem.

9.3.4 Singular Systems

In the previous section, we learned that one way in which a system of equations can be ill-conditioned is when two or more of the equations are nearly identical. Obviously, it is even worse when the two are identical. In such cases, we would lose one degree of freedom, and would be dealing with the impossible case of $n - 1$ equations with n unknowns. Such cases might not be obvious to you, particularly when dealing with large equation sets. Consequently, it would be nice to have some way of automatically detecting singularity.

The answer to this problem is neatly offered by the fact that the determinant of a singular system is zero. This idea can, in turn, be connected to Gauss elimination by recognizing that after the elimination step, the determinant can be evaluated as the product of the diagonal elements (recall Box 9.1). Thus, a computer algorithm can test to discern whether a zero diagonal element is created during the elimination stage. If one is discovered, the calculation can be immediately terminated and a message displayed alerting the user. We

will show the details of how this is done when we present a full algorithm for Gauss elimination later in this chapter.

9.4 TECHNIQUES FOR IMPROVING SOLUTIONS

The following techniques can be incorporated into the naive Gauss elimination algorithm to circumvent some of the pitfalls discussed in the previous section.

9.4.1 Use of More Significant Figures

The simplest remedy for ill-conditioning is to use more significant figures in the computation. If your application can be extended to handle larger word size, such a feature will greatly reduce the problem. However, a price must be paid in the form of the computational and memory overhead connected with using extended precision (recall Sec. 3.4.1).

9.4.2 Pivoting

As mentioned at the beginning of Sec. 9.3, obvious problems occur when a pivot element is zero because the normalization step leads to division by zero. Problems may also arise when the pivot element is close to, rather than exactly equal to, zero because if the magnitude of the pivot element is small compared to the other elements, then round-off errors can be introduced.

Therefore, before each row is normalized, it is advantageous to determine the largest available coefficient in the column below the pivot element. The rows can then be switched so that the largest element is the pivot element. This is called *partial pivoting*. If columns as well as rows are searched for the largest element and then switched, the procedure is called *complete pivoting*. Complete pivoting is rarely used because switching columns changes the order of the x's and, consequently, adds significant and usually unjustified complexity to the computer program. The following example illustrates the advantages of partial pivoting. Aside from avoiding division by zero, pivoting also minimizes round-off error. As such, it also serves as a partial remedy for ill-conditioning.

EXAMPLE 9.9 Partial Pivoting

Problem Statement. Use Gauss elimination to solve

$$0.0003x_1 + 3.0000x_2 = 2.0001$$
$$1.0000x_1 + 1.0000x_2 = 1.0000$$

Note that in this form the first pivot element, $a_{11} = 0.0003$, is very close to zero. Then repeat the computation, but partial pivot by reversing the order of the equations. The exact solution is $x_1 = 1/3$ and $x_2 = 2/3$.

Solution. Multiplying the first equation by $1/(0.0003)$ yields

$$x_1 + 10{,}000x_2 = 6667$$

which can be used to eliminate x_1 from the second equation:

$$-9999x_2 = -6666$$

which can be solved for

$$x_2 = \frac{2}{3}$$

This result can be substituted back into the first equation to evaluate x_1:

$$x_1 = \frac{2.0001 - 3(2/3)}{0.0003} \tag{E9.9.1}$$

However, due to subtractive cancellation, the result is very sensitive to the number of significant figures carried in the computation:

Significant Figures	x_2	x_1	Absolute Value of Percent Relative Error for x_1
3	0.667	−3.33	1099
4	0.6667	0.0000	100
5	0.66667	0.30000	10
6	0.666667	0.330000	1
7	0.6666667	0.3330000	0.1

Note how the solution for x_1 is highly dependent on the number of significant figures. This is because in Eq. (E9.9.1), we are subtracting two almost-equal numbers. On the other hand, if the equations are solved in reverse order, the row with the larger pivot element is normalized. The equations are

$$1.0000x_1 + 1.0000x_2 = 1.0000$$
$$0.0003x_1 + 3.0000x_2 = 2.0001$$

Elimination and substitution yield $x_2 = 2/3$. For different numbers of significant figures, x_1 can be computed from the first equation, as in

$$x_1 = \frac{1 - (2/3)}{1} \tag{E9.9.2}$$

This case is much less sensitive to the number of significant figures in the computation:

Significant Figures	x_2	x_1	Absolute Value of Percent Relative Error for x_1
3	0.667	0.333	0.1
4	0.6667	0.3333	0.01
5	0.66667	0.33333	0.001
6	0.666667	0.333333	0.0001
7	0.6666667	0.3333333	0.00001

Thus, a pivot strategy is much more satisfactory.

```
p = k
big = |a_k,k|
DO ii = k+1, n
  dummy = |a_ii,k|
  IF (dummy > big)
    big = dummy
    p = ii
  END IF
END DO
IF (p ≠ k)
  DO jj = k, n
    dummy = a_p,jj
    a_p,jj = a_k,jj
    a_k,jj = dummy
  END DO
  dummy = b_p
  b_p = b_k
  b_k = dummy
END IF
```

FIGURE 9.5
Pseudocode to implement partial pivoting.

General-purpose computer programs must include a pivot strategy. Figure 9.5 provides a simple algorithm to implement such a strategy. Notice that the algorithm consists of two major loops. After storing the current pivot element and its row number as the variables, *big* and *p*, the first loop compares the pivot element with the elements below it to check whether any of these is larger than the pivot element. If so, the new largest element and its row number are stored in *big* and *p*. Then, the second loop switches the original pivot row with the one with the largest element so that the latter becomes the new pivot row. This pseudocode can be integrated into a program based on the other elements of Gauss elimination outlined in Fig. 9.4. The best way to do this is to employ a modular approach and write Fig. 9.5 as a subroutine (or procedure) that would be called directly after the beginning of the first loop in Fig. 9.4*a*.

Note that the second IF/THEN construct in Fig. 9.5 physically interchanges the rows. For large matrices, this can become quite time consuming. Consequently, most codes do not actually exchange rows but rather keep track of the pivot rows by storing the appropriate subscripts in a vector. This vector then provides a basis for specifying the proper row ordering during the forward-elimination and back-substitution operations. Thus, the operations are said to be implemented *in place*.

9.4.3 Scaling

In Sec. 9.3.3, we proposed that scaling had value in standardizing the size of the determinant. Beyond this application, it has utility in minimizing round-off errors for those cases where some of the equations in a system have much larger coefficients than others. Such situations are frequently encountered in engineering practice when widely different units are used in the development of simultaneous equations. For instance, in electric-circuit problems, the unknown voltages can be expressed in units ranging from microvolts to kilovolts. Similar examples can arise in all fields of engineering. As long as each equation is consistent, the system will be technically correct and solvable. However, the use of widely differing units can lead to coefficients of widely differing magnitudes. This, in turn, can have an impact on round-off error as it affects pivoting, as illustrated by the following example.

EXAMPLE 9.10 Effect of Scaling on Pivoting and Round-Off

Problem Statement.

(a) Solve the following set of equations using Gauss elimination and a pivoting strategy:

$$2x_1 + 100{,}000x_2 = 100{,}000$$
$$x_1 + \quad x_2 = 2$$

(b) Repeat the solution after scaling the equations so that the maximum coefficient in each row is 1.

(c) Finally, use the scaled coefficients to determine whether pivoting is necessary. However, actually solve the equations with the original coefficient values. For all cases, retain only three significant figures. Note that the correct answers are $x_1 = 1.00002$ and $x_2 = 0.99998$ or, for three significant figures, $x_1 = x_2 = 1.00$.

Solution.

(a) Without scaling, forward elimination is applied to give

$$2x_1 + 100{,}000x_2 = 100{,}000$$
$$-50{,}000x_2 = -50{,}000$$

which can be solved by back substitution for

$$x_2 = 1.00$$
$$x_1 = 0.00$$

Although x_2 is correct, x_1 is 100 percent in error because of round-off.

(b) Scaling transforms the original equations to

$$0.00002x_1 + x_2 = 1$$
$$x_1 + x_2 = 2$$

Therefore, the rows should be pivoted to put the greatest value on the diagonal.

$$x_1 + x_2 = 2$$
$$0.00002x_1 + x_2 = 1$$

Forward elimination yields

$$x_1 + x_2 = 2$$
$$x_2 = 1.00$$

which can be solved for

$$x_1 = x_2 = 1$$

Thus, scaling leads to the correct answer.

(c) The scaled coefficients indicate that pivoting is necessary. We therefore pivot but retain the original coefficients to give

$$x_1 + x_2 = 2$$
$$2x_1 + 100{,}000x_2 = 100{,}000$$

Forward elimination yields

$$x_1 + x_2 = 2$$
$$100{,}000x_2 = 100{,}000$$

which can be solved for the correct answer: $x_1 = x_2 = 1$. Thus, scaling was useful in determining whether pivoting was necessary, but the equations themselves did not require scaling to arrive at a correct result.

```
SUB Gauss (a, b, n, x, tol, er)
  DIMENSION s (n)
  er = 0
  DO i = 1, n
    s_i = ABS(a_i,1)
    DO j = 2, n
      IF ABS(a_i,j)>s_i THEN s_i = ABS(a_i,j)
    END DO
  END DO
  CALL Eliminate(a, s, n, b, tol, er)
  IF er ≠ -1 THEN
     CALL Substitute(a, n, b, x)
  END IF
END Gauss


SUB Eliminate (a, s, n, b, tol, er)
  DO k = 1, n - 1
    CALL Pivot (a, b, s, n, k)
    IF ABS (a_k,k/s_k) < tol THEN
      er = -1
      EXIT DO
    END IF
    DO i = k + 1, n
      factor = a_i,k/a_k,k
      DO j = k + 1, n
        a_i,j = a_i,j - factor*a_k,j
      END DO
      b_i = b_i - factor * b_k
    END DO
  END DO
  IF ABS(a_k,k/s_k) < tol THEN er = -1
END Eliminate


SUB Pivot (a, b, s, n, k)
  p = k
  big = ABS(a_k,k/s_k)
  DO ii = k + 1, n
    dummy = ABS(a_ii,k/s_ii)
    IF dummy > big THEN
       big = dummy
       p = ii
    END IF
  END DO
  IF p ≠ k THEN
     DO jj = k, n
       dummy = a_p,jj
       a_p,jj = a_k,jj
       a_k,jj = dummy
     END DO
     dummy = b_p
     b_p = b_k
     b_k = dummy
     dummy = s_p
     s_p = s_k
     s_k = dummy
  END IF
END pivot


SUB Substitute (a, n, b, x)
  x_n = b_n/a_n,n
  DO i = n - 1, 1, -1
    sum = 0
    DO j = i + 1, n
      sum = sum + a_i,j * x_j
    END DO
    x_i = (b_i - sum) / a_i,i
  END DO
END Substitute
```

FIGURE 9.6
Pseudocode to implement Gauss elimination with partial pivoting.