```
DOFOR k = 1, n
  DOFOR i = 1, k − 1
    sum = 0.
    DOFOR j = 1, i − 1
      sum = sum + a_ij · a_kj
    END DO
    a_kj = (a_kj − sum)/a_ii
  END DO
  sum = 0.
  DOFOR j = 1, k − 1
    sum = sum + a²_kj
  END DO
  a_kk = √(a_kk − sum)
END DO
```

FIGURE 11.3
Pseudocode for Cholesky's *LU* decomposition algorithm.

and Eq. (11.4) yields

$$l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2} = \sqrt{979 - (22.454)^2 - (20.917)^2} = 6.1101$$

Thus, the Cholesky decomposition yields

$$[L] = \begin{bmatrix} 2.4495 & & \\ 6.1237 & 4.1833 & \\ 22.454 & 20.917 & 6.1101 \end{bmatrix}$$

The validity of this decomposition can be verified by substituting it and its transpose into Eq. (11.2) to see if their product yields the original matrix $[A]$. This is left for an exercise.

Figure 11.3 presents pseudocode for implementing the Cholesky decomposition algorithm. It should be noted that the algorithm in Fig. 11.3 could result in an execution error if the evaluation of a_{kk} involves taking the square root of a negative number. However, for cases where the matrix is *positive definite,*[1] this will never occur. Because many symmetric matrices dealt with in engineering are, in fact, positive definite, the Cholesky algorithm has wide application. Another benefit of dealing with positive definite symmetric matrices is that pivoting is not required to avoid division by zero. Thus, we can implement the algorithm in Fig. 11.3 without the complication of pivoting.

11.2 GAUSS-SEIDEL

Iterative or approximate methods provide an alternative to the elimination methods described to this point. Such approaches are similar to the techniques we developed to obtain the roots of a single equation in Chap. 6. Those approaches consisted of guessing a value and then using a systematic method to obtain a refined estimate of the root. Because the present part of the book deals with a similar problem—obtaining the values that simultaneously satisfy a set of equations—we might suspect that such approximate methods could be useful in this context.

The *Gauss-Seidel method* is the most commonly used iterative method. Assume that we are given a set of n equations:

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

Suppose that for conciseness we limit ourselves to a 3×3 set of equations. If the diagonal elements are all nonzero, the first equation can be solved for x_1, the second for x_2, and the third for x_3 to yield

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \tag{11.5a}$$

[1]A *positive definite matrix* is one for which the product $\{X\}^T [A]\{X\}$ is greater than zero for all nonzero vectors $\{X\}$.

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \tag{11.5b}$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \tag{11.5c}$$

Now, we can start the solution process by choosing guesses for the x's. A simple way to obtain initial guesses is to assume that they are all zero. These zeros can be substituted into Eq. (11.5*a*), which can be used to calculate a new value for $x_1 = b_1/a_{11}$. Then, we substitute this new value of x_1 along with the previous guess of zero for x_3 into Eq. (11.5*b*) to compute a new value for x_2. The process is repeated for Eq. (11.5*c*) to calculate a new estimate for x_3. Then we return to the first equation and repeat the entire procedure until our solution converges closely enough to the true values. Convergence can be checked using the criterion [recall Eq. (3.5)]

$$|\varepsilon_{a,i}| = \left| \frac{x_i^j - x_i^{j-1}}{x_i^j} \right| 100\% < \varepsilon_s \tag{11.6}$$

for all i, where j and $j - 1$ are the present and previous iterations.

EXAMPLE 11.3 Gauss-Seidel Method

Problem Statement. Use the Gauss-Seidel method to obtain the solution of the same system used in Example 10.2:

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

Recall that the true solution is $x_1 = 3$, $x_2 = -2.5$, and $x_3 = 7$.

Solution. First, solve each of the equations for its unknown on the diagonal.

$$x_1 = \frac{7.85 + 0.1x_2 + 0.2x_3}{3} \tag{E11.3.1}$$

$$x_2 = \frac{-19.3 - 0.1x_1 + 0.3x_3}{7} \tag{E11.3.2}$$

$$x_3 = \frac{71.4 - 0.3x_1 + 0.2x_2}{10} \tag{E11.3.3}$$

By assuming that x_2 and x_3 are zero, Eq. (E11.3.1) can be used to compute

$$x_1 = \frac{7.85 + 0 + 0}{3} = 2.616667$$

This value, along with the assumed value of $x_3 = 0$, can be substituted into Eq. (E11.3.2) to calculate

$$x_2 = \frac{-19.3 - 0.1(2.616667) + 0}{7} = -2.794524$$

The first iteration is completed by substituting the calculated values for x_1 and x_2 into Eq. (E11.3.3) to yield

$$x_3 = \frac{71.4 - 0.3(2.616667) + 0.2(-2.794524)}{10} = 7.005610$$

For the second iteration, the same process is repeated to compute

$$x_1 = \frac{7.85 + 0.1(-2.794524) + 0.2(7.005610)}{3} = 2.990557 \qquad |\varepsilon_t| = 0.31\%$$

$$x_2 = \frac{-19.3 - 0.1(2.990557) + 0.3(7.005610)}{7} = -2.499625 \qquad |\varepsilon_t| = 0.015\%$$

$$x_3 = \frac{71.4 - 0.3(2.990557) + 0.2(-2.499625)}{10} = 7.000291 \qquad |\varepsilon_t| = 0.0042\%$$

The method is, therefore, converging on the true solution. Additional iterations could be applied to improve the answers. However, in an actual problem, we would not know the true answer a priori. Consequently, Eq. (11.6) provides a means to estimate the error. For example, for x_1,

$$|\varepsilon_{a,1}| = \left|\frac{2.990557 - 2.616667}{2.990557}\right| 100\% = 12.5\%$$

For x_2 and x_3, the error estimates are $|\varepsilon_{a,2}| = 11.8\%$ and $|\varepsilon_{a,3}| = 0.076\%$. Note that, as was the case when determining roots of a single equation, formulations such as Eq. (11.6) usually provide a conservative appraisal of convergence. Thus, when they are met, they ensure that the result is known to at least the tolerance specified by ε_s.

As each new x value is computed for the Gauss-Seidel method, it is immediately used in the next equation to determine another x value. Thus, if the solution is converging, the best available estimates will be employed. An alternative approach, called *Jacobi iteration*, utilizes a somewhat different tactic. Rather than using the latest available x's, this technique uses Eq. (11.5) to compute a set of new x's on the basis of a set of old x's. Thus, as new values are generated, they are not immediately used but rather are retained for the next iteration.

The difference between the Gauss-Seidel method and Jacobi iteration is depicted in Fig. 11.4. Although there are certain cases where the Jacobi method is useful, Gauss-Seidel's utilization of the best available estimates usually makes it the method of preference.

11.2.1 Convergence Criterion for the Gauss-Seidel Method

Note that the Gauss-Seidel method is similar in spirit to the technique of simple fixed-point iteration that was used in Sec. 6.1 to solve for the roots of a single equation. Recall that simple fixed-point iteration had two fundamental problems: (1) it was sometimes nonconvergent and (2) when it converged, it often did so very slowly. The Gauss-Seidel method can also exhibit these shortcomings.

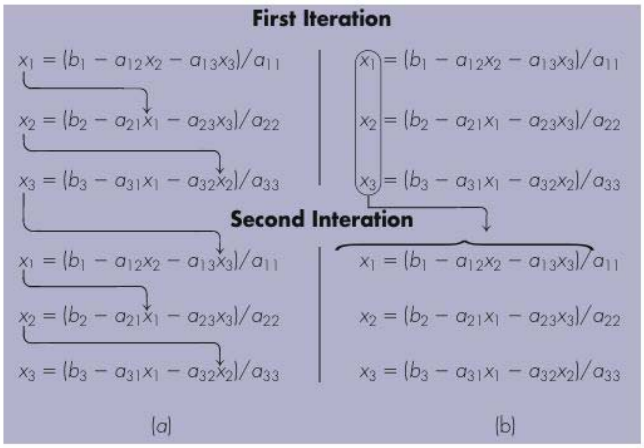


FIGURE 11.4
Graphical depiction of the difference between (*a*) the Gauss-Seidel and (*b*) the Jacobi iterative methods for solving simultaneous linear algebraic equations.

Convergence criteria can be developed by recalling from Sec. 6.5.1 that sufficient conditions for convergence of two nonlinear equations, $u(x, y)$ and $v(x, y)$, are

$$\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial u}{\partial y}\right| < 1 \tag{11.7a}$$

and

$$\left|\frac{\partial v}{\partial x}\right| + \left|\frac{\partial v}{\partial y}\right| < 1 \tag{11.7b}$$

These criteria also apply to linear equations of the sort we are solving with the Gauss-Seidel method. For example, in the case of two simultaneous equations, the Gauss-Seidel algorithm [Eq. (11.5)] can be expressed as

$$u(x_1, x_2) = \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 \tag{11.8a}$$

and

$$v(x_1, x_2) = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 \tag{11.8b}$$

The partial derivatives of these equations can be evaluated with respect to each of the unknowns as

$$\frac{\partial u}{\partial x_1} = 0 \qquad \frac{\partial u}{\partial x_2} = -\frac{a_{12}}{a_{11}}$$

and

$$\frac{\partial v}{\partial x_1} = -\frac{a_{21}}{a_{22}} \qquad \frac{\partial v}{\partial x_2} = 0$$

which can be substituted into Eq. (11.7) to give

$$\left|\frac{a_{12}}{a_{11}}\right| < 1 \tag{11.9a}$$

and

$$\left|\frac{a_{21}}{a_{22}}\right| < 1 \tag{11.9b}$$

In other words, the absolute values of the slopes of Eq. (11.8) must be less than unity to ensure convergence. This is displayed graphically in Fig. 11.5. Equation (11.9) can also be reformulated as

$$|a_{11}| > |a_{12}|$$

and

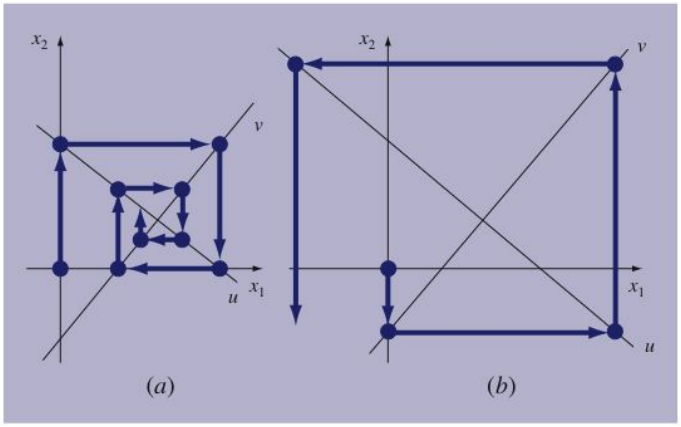
$$|a_{22}| > |a_{21}|$$

That is, the diagonal element must be greater than the off-diagonal element for each row.

The extension of the above to n equations is straightforward and can be expressed as

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \tag{11.10}$$

FIGURE 11.5
Illustration of (*a*) convergence and (*b*) divergence of the Gauss-Seidel method. Notice that the same functions are plotted in both cases (u: $11x_1 + 13x_2 = 286$; v: $11x_1 - 9x_2 = 99$). Thus, the order in which the equations are implemented (as depicted by the direction of the first arrow from the origin) dictates whether the computation converges.



That is, the diagonal coefficient in each of the equations must be larger than the sum of the absolute values of the other coefficients in the equation. This criterion is sufficient but not necessary for convergence. That is, although the method may sometimes work if Eq. (11.10) is not met, convergence is guaranteed if the condition is satisfied. Systems where Eq. (11.10) holds are called *diagonally dominant.* Fortunately, many engineering problems of practical importance fulfill this requirement.

11.2.2 Improvement of Convergence Using Relaxation

Relaxation represents a slight modification of the Gauss-Seidel method and is designed to enhance convergence. After each new value of x is computed using Eq. (11.5), that value is modified by a weighted average of the results of the previous and the present iterations:

$$x_i^{\text{new}} = \lambda x_i^{\text{new}} + (1 - \lambda) x_i^{\text{old}} \tag{11.11}$$

where λ is a weighting factor that is assigned a value between 0 and 2.

If $\lambda = 1$, $(1 - \lambda)$ is equal to 0 and the result is unmodified. However, if λ is set at a value between 0 and 1, the result is a weighted average of the present and the previous results. This type of modification is called *underrelaxation.* It is typically employed to make a nonconvergent system converge or to hasten convergence by dampening out oscillations.

For values of λ from 1 to 2, extra weight is placed on the present value. In this instance, there is an implicit assumption that the new value is moving in the correct direction toward the true solution but at too slow a rate. Thus, the added weight of λ is intended to improve the estimate by pushing it closer to the truth. Hence, this type of modification, which is called *overrelaxation,* is designed to accelerate the convergence of an already convergent system. The approach is also called *successive* or *simultaneous overrelaxation,* or *SOR.*

The choice of a proper value for λ is highly problem-specific and is often determined empirically. For a single solution of a set of equations it is often unnecessary. However, if the system under study is to be solved repeatedly, the efficiency introduced by a wise choice of λ can be extremely important. Good examples are the very large systems of partial differential equations that often arise when modeling continuous variations of variables (recall the distributed system depicted in Fig. PT3.1*b*). We will return to this topic in Part Eight.

11.2.3 Algorithm for Gauss-Seidel

An algorithm for the Gauss-Seidel method, with relaxation, is depicted in Fig. 11.6. Note that this algorithm is not guaranteed to converge if the equations are not input in a diagonally dominant form.

The pseudocode has two features that bear mentioning. First, there is an initial set of nested loops to divide each equation by its diagonal element. This reduces the total number of operations in the algorithm. Second, notice that the error check is designated by a variable called *sentinel*. If any of the equations has an approximate error greater than the stopping criterion (e_s), then the iterations are allowed to continue. The use of the sentinel allows us

```
SUBROUTINE Gseid (a,b,n,x,imax,es,lambda)
 DOFOR i = 1,n
  dummy = a_{i,i}
  DOFOR j = 1,n
   a_{i,j} = a_{i,j}/dummy
  END DO
  b_i = b_i/dummy
 END DO
 DOFOR i = 1, n
  sum = b_i
  DOFOR j = 1, n
   IF i≠j THEN sum = sum − a_{i,j}*x_j
  END DO
  x_i=sum
 END DO
 iter=1
 DO
  sentinel = 1
  DOFOR i = 1,n
   old = x_i
   sum = b_i
   DOFOR j = 1,n
    IF i≠j THEN sum = sum − a_{i,j}*x_j
   END DO
   x_i = lambda*sum +(1.−lambda)*old
   IF sentinel = 1 AND x_i ≠ 0. THEN
      ea = ABS((x_i − old)/x_i)*100.
      IF ea > es THEN sentinel = 0
   END IF
  END DO
  iter = iter + 1
  IF sentinel = 1 OR (iter ≥ imax) EXIT
 END DO
END Gseid
```

FIGURE 11.6
Pseudocode for Gauss-Seidel with relaxation.

to circumvent unnecessary calculations of error estimates once one of the equations exceeds the criterion.

11.2.4 Problem Contexts for the Gauss-Seidel Method

Aside from circumventing the round-off dilemma, the Gauss-Seidel technique has a number of other advantages that make it particularly attractive in the context of certain engineering problems. For example, when the matrix in question is very large and very sparse (that is, most of the elements are zero), elimination methods waste large amounts of computer memory by storing zeros.