CHAPTER 5

Bracketing Methods

This chapter on roots of equations deals with methods that exploit the fact that a function typically changes sign in the vicinity of a root. These techniques are called *bracketing methods* because two initial guesses for the root are required. As the name implies, these guesses must "bracket," or be on either side of, the root. The particular methods described herein employ different strategies to systematically reduce the width of the bracket and, hence, home in on the correct answer.

As a prelude to these techniques, we will briefly discuss graphical methods for depicting functions and their roots. Beyond their utility for providing rough guesses, graphical techniques are also useful for visualizing the properties of the functions and the behavior of the various numerical methods.

5.1 GRAPHICAL METHODS

A simple method for obtaining an estimate of the root of the equation $f(x) = 0$ is to make a plot of the function and observe where it crosses the x axis. This point, which represents the x value for which $f(x) = 0$, provides a rough approximation of the root.

EXAMPLE 5.1 The Graphical Approach

Problem Statement. Use the graphical approach to determine the drag coefficient c needed for a parachutist of mass $m = 68.1$ kg to have a velocity of 40 m/s after free-falling for time $t = 10$ s. *Note:* The acceleration due to gravity is 9.8 m/s^2.

Solution. This problem can be solved by determining the root of Eq. (PT2.4) using the parameters $t = 10$, $g = 9.8$, $v = 40$, and $m = 68.1$:

$$f(c) = \frac{9.8(68.1)}{c}\left(1 - e^{-(c/68.1)10}\right) - 40$$

or

$$f(c) = \frac{667.38}{c}\left(1 - e^{-0.146843c}\right) - 40 \qquad \text{(E5.1.1)}$$

Various values of c can be substituted into the right-hand side of this equation to compute

c	f(c)
4	34.115
8	17.653
12	6.067
16	−2.269
20	−8.401

These points are plotted in Fig. 5.1. The resulting curve crosses the c axis between 12 and 16. Visual inspection of the plot provides a rough estimate of the root of 14.75. The validity of the graphical estimate can be checked by substituting it into Eq. (E5.1.1) to yield

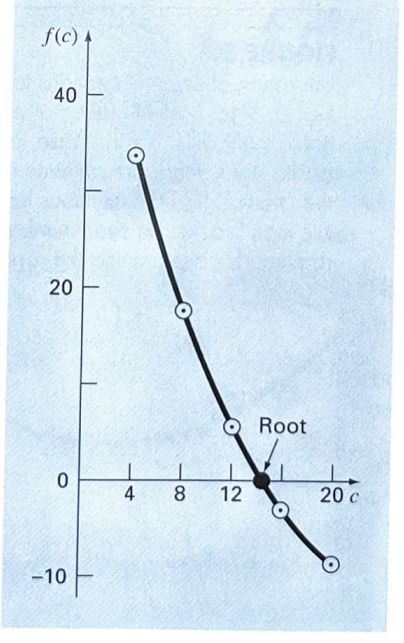
$$f(14.75) = \frac{667.38}{14.75}\left(1 - e^{-0.146843(14.75)}\right) - 40 = 0.059$$

which is close to zero. It can also be checked by substituting it into Eq. (PT2.4) along with the parameter values from this example to give

$$v = \frac{9.8(68.1)}{14.75}\left(1 - e^{-(14.75/68.1)10}\right) = 40.059$$

which is very close to the desired fall velocity of 40 m/s.

FIGURE 5.1
The graphical approach for determining the roots of an equation.



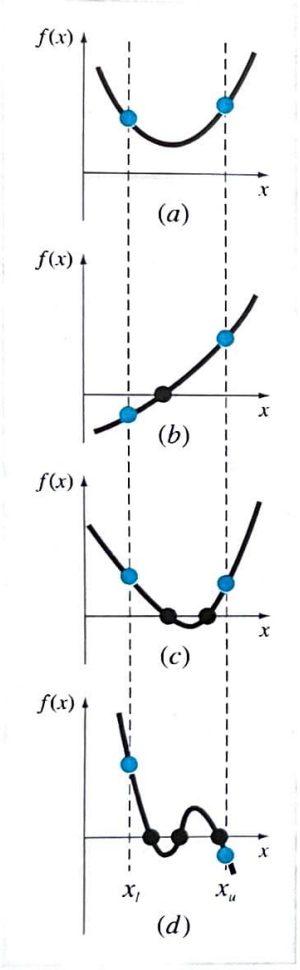


FIGURE 5.2
Illustration of a number of general ways that a root may occur in an interval prescribed by a lower bound x_l and an upper bound x_u. Parts (*a*) and (*c*) indicate that if both $f(x_l)$ and $f(x_u)$ have the same sign, either there will be no roots or there will be an even number of roots within the interval. Parts (*b*) and (*d*) indicate that if the function has different signs at the end points, there will be an odd number of roots in the interval.

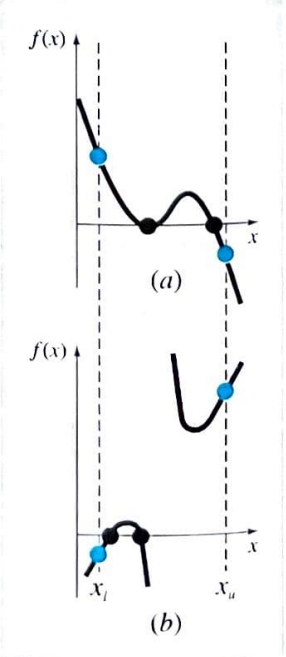
Graphical techniques are of limited practical value because they are not precise. However, graphical methods can be utilized to obtain rough estimates of roots. These estimates can be employed as starting guesses for numerical methods discussed in this and the next chapter.

Aside from providing rough estimates of the root, graphical interpretations are important tools for understanding the properties of the functions and anticipating the pitfalls of the numerical methods. For example, Fig. 5.2 shows a number of ways in which roots can occur (or be absent) in an interval prescribed by a lower bound x_l and an upper bound x_u. Figure 5.2*b* depicts the case where a single root is bracketed by negative and positive values of $f(x)$. However, Fig. 5.2*d*, where $f(x_l)$ and $f(x_u)$ are also on opposite sides of the x axis, shows three roots occurring within the interval. In general, if $f(x_l)$ and $f(x_u)$ have opposite signs, there are an odd number of roots in the interval. As indicated by Fig. 5.2*a* and *c*, if $f(x_l)$ and $f(x_u)$ have the same sign, there are either no roots or an even number of roots between the values.

Although these generalizations are usually true, there are cases where they do not hold. For example, functions that are tangential to the x axis (Fig. 5.3*a*) and discontinuous functions (Fig. 5.3*b*) can violate these principles. An example of a function that is tangential to the axis is the cubic equation $f(x) = (x - 2)(x - 2)(x - 4)$. Notice that $x = 2$ makes two terms in this polynomial equal to zero. Mathematically, $x = 2$ is called a *multiple root.* At the end of Chap. 6, we will present techniques that are expressly designed to locate multiple roots.

The existence of cases of the type depicted in Fig. 5.3 makes it difficult to develop general computer algorithms guaranteed to locate all the roots in an interval. However, when used in conjunction with graphical approaches, the methods described in the following

FIGURE 5.3
Illustration of some exceptions to the general cases depicted in Fig. 5.2. (*a*) Multiple root that occurs when the function is tangential to the x axis. For this case, although the end points are of opposite signs, there are an even number of axis intersections for the interval. (*b*) Discontinuous function where end points of opposite sign bracket an even number of roots. Special strategies are required for determining the roots for these cases.



sections are extremely useful for solving many roots of equations problems confronted routinely by engineers and applied mathematicians.

EXAMPLE 5.2 Use of Computer Graphics to Locate Roots

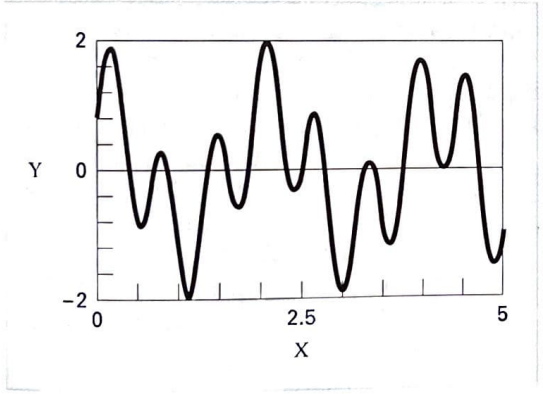
Problem Statement. Computer graphics can expedite and improve your efforts to locate roots of equations. The function

$$f(x) = \sin 10x + \cos 3x$$

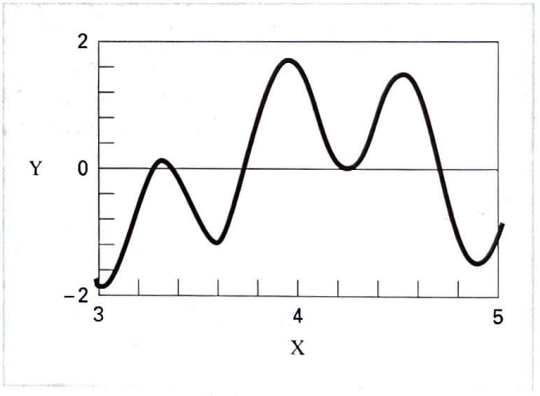
has several roots over the range $x = 0$ to $x = 5$. Use computer graphics to gain insight into the behavior of this function.

Solution. Packages such as Excel and MATLAB software can be used to generate plots. Figure 5.4*a* is a plot of $f(x)$ from $x = 0$ to $x = 5$. This plot suggests the presence of several roots, including a possible double root at about $x = 4.2$ where $f(x)$ appears to be tangent to

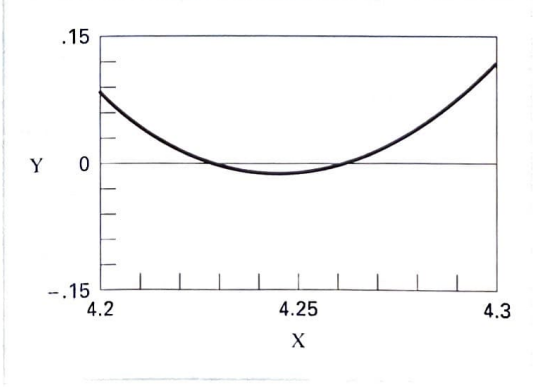
FIGURE 5.4
The progressive enlargement of $f(x) = \sin 10x + \cos 3x$ by the computer. Such interactive graphics permits the analyst to determine that two distinct roots exist between $x = 4.2$ and $x = 4.3$.



(*a*)



(*b*)



(*c*)

the x axis. A more detailed picture of the behavior of $f(x)$ is obtained by changing the plotting range from $x = 3$ to $x = 5$, as shown in Fig. 5.4*b*. Finally, in Fig. 5.4*c*, the vertical scale is narrowed further to $f(x) = -0.15$ to $f(x) = 0.15$ and the horizontal scale is narrowed to $x = 4.2$ to $x = 4.3$. This plot shows clearly that a double root does not exist in this region and that in fact there are two distinct roots at about $x = 4.23$ and $x = 4.26$.

Computer graphics will have great utility in your studies of numerical methods. This capability will also find many other applications in your other classes and professional activities as well.

5.2 THE BISECTION METHOD

When applying the graphical technique in Example 5.1, you have observed (Fig. 5.1) that $f(x)$ changed sign on opposite sides of the root. In general, if $f(x)$ is real and continuous in the interval from x_l to x_u and $f(x_l)$ and $f(x_u)$ have opposite signs, that is,

$$f(x_l)\,f(x_u) < 0 \tag{5.1}$$

then there is at least one real root between x_l and x_u.

Incremental search methods capitalize on this observation by locating an interval where the function changes sign. Then the location of the sign change (and consequently, the root) is identified more precisely by dividing the interval into a number of subintervals. Each of these subintervals is searched to locate the sign change. The process is repeated and the root estimate refined by dividing the subintervals into finer increments. We will return to the general topic of incremental searches in Sec. 5.4.

The *bisection method,* which is alternatively called binary chopping, interval halving, or Bolzano's method, is one type of incremental search method in which the interval is always divided in half. If a function changes sign over an interval, the function value at the midpoint is evaluated. The location of the root is then determined as lying at the midpoint of the subinterval within which the sign change occurs. The process is repeated to obtain refined estimates. A simple algorithm for the bisection calculation is listed in Fig. 5.5, and a graphical depiction of the method is provided in Fig. 5.6. The following example goes through the actual computations involved in the method.

FIGURE 5.5

Step 1: Choose lower x_l and upper x_u guesses for the root such that the function changes sign over the interval. This can be checked by ensuring that $f(x_l)f(x_u) < 0$.

Step 2: An estimate of the root x_r is determined by

$$x_r = \frac{x_l + x_u}{2}$$

Step 3: Make the following evaluations to determine in which subinterval the root lies:

(*a*) If $f(x_l)f(x_r) < 0$, the root lies in the lower subinterval. Therefore, set $x_u = x_r$ and return to step 2.

(*b*) If $f(x_l)f(x_r) > 0$, the root lies in the upper subinterval. Therefore, set $x_l = x_r$ and return to step 2.

(*c*) If $f(x_l)f(x_r) = 0$, the root equals x_r; terminate the computation.

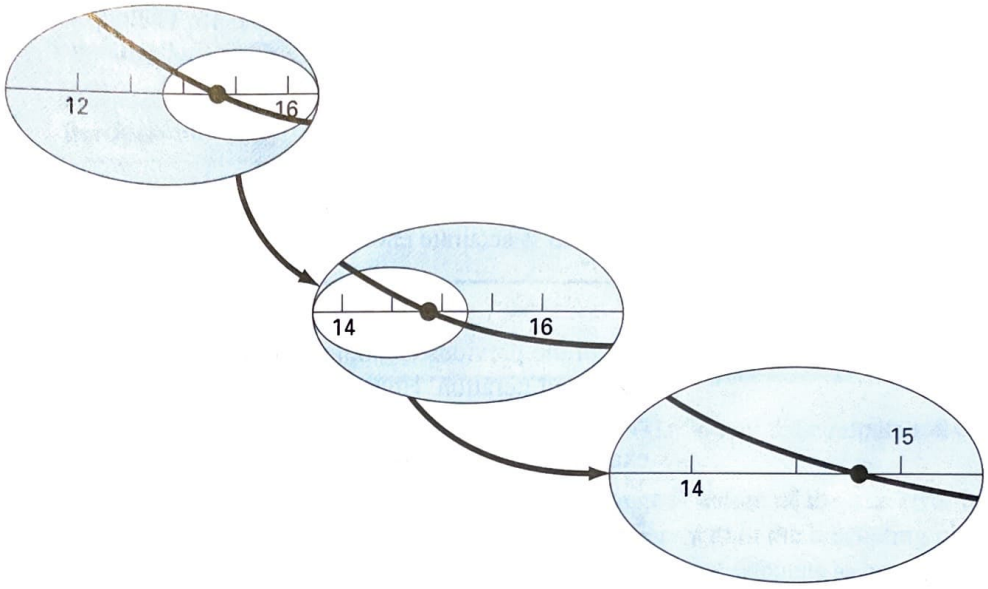


FIGURE 5.6
A graphical depiction of the bisection method. This plot conforms to the first three iterations from Example 5.3.

EXAMPLE 5.3 Bisection

Problem Statement. Use bisection to solve the same problem approached graphically in Example 5.1.

Solution. The first step in bisection is to guess two values of the unknown (in the present problem, c) that give values for $f(c)$ with different signs. From Fig. 5.1, we can see that the function changes sign between values of 12 and 16. Therefore, the initial estimate of the root x_r lies at the midpoint of the interval

$$x_r = \frac{12 + 16}{2} = 14$$

This estimate represents a true percent relative error of $\varepsilon_t = 5.3\%$ (note that the true value of the root is 14.7802). Next we compute the product of the function value at the lower bound and at the midpoint:

$$f(12)f(14) = 6.067(1.569) = 9.517$$

which is greater than zero, and hence no sign change occurs between the lower bound and the midpoint. Consequently, the root must be located between 14 and 16. Therefore, we create a new interval by redefining the lower bound as 14 and determining a revised root estimate as

$$x_r = \frac{14 + 16}{2} = 15$$

which represents a true percent error of $\varepsilon_t = 1.5\%$. The process can be repeated to obtain refined estimates. For example,

$$f(14)f(15) = 1.569(-0.425) = -0.666$$

Therefore, the root is between 14 and 15. The upper bound is redefined as 15, and the root estimate for the third iteration is calculated as

$$x_r = \frac{14 + 15}{2} = 14.5$$

which represents a percent relative error of $\varepsilon_t = 1.9\%$. The method can be repeated until the result is accurate enough to satisfy your needs.

In the previous example, you may have noticed that the true error does not decrease with each iteration. However, the interval within which the root is located is halved with each step in the process. As discussed in the next section, the interval width provides an exact estimate of the upper bound of the error for the bisection method.

5.2.1 Termination Criteria and Error Estimates

We ended Example 5.3 with the statement that the method could be continued to obtain a refined estimate of the root. We must now develop an objective criterion for deciding when to terminate the method.

An initial suggestion might be to end the calculation when the true error falls below some prespecified level. For instance, in Example 5.3, the relative error dropped from 5.3 to 1.9 percent during the course of the computation. We might decide that we should terminate when the error drops below, say, 0.1 percent. This strategy is flawed because the error estimates in the example were based on knowledge of the true root of the function. This would not be the case in an actual situation because there would be no point in using the method if we already knew the root.

Therefore, we require an error estimate that is not contingent on foreknowledge of the root. As developed previously in Sec. 3.3, an approximate percent relative error ε_a can be calculated, as in [recall Eq. (3.5)]

$$\varepsilon_a = \left| \frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}} \right| 100\% \tag{5.2}$$

where x_r^{new} is the root for the present iteration and x_r^{old} is the root from the previous iteration. The absolute value is used because we are usually concerned with the magnitude of ε_a rather than with its sign. When ε_a becomes less than a prespecified stopping criterion ε_s, the computation is terminated.

EXAMPLE 5.4 Error Estimates for Bisection

Problem Statement. Continue Example 5.3 until the approximate error falls below a stopping criterion of $\varepsilon_s = 0.5\%$. Use Eq. (5.2) to compute the errors.

Solution. The results of the first two iterations for Example 5.3 were 14 and 15. Substituting these values into Eq. (5.2) yields

$$|\varepsilon_a| = \left| \frac{15 - 14}{15} \right| 100\% = 6.667\%$$

Recall that the true percent relative error for the root estimate of 15 was 1.5%. Therefore, ε_a is greater than ε_t. This behavior is manifested for the other iterations:

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	12	16	14		5.279
2	14	16	15	6.667	1.487
3	14	15	14.5	3.448	1.896
4	14.5	15	14.75	1.695	0.204
5	14.75	15	14.875	0.840	0.641
6	14.75	14.875	14.8125	0.422	0.219

Thus, after six iterations ε_a finally falls below $\varepsilon_s = 0.5\%$, and the computation can be terminated.

These results are summarized in Fig. 5.7. The "ragged" nature of the true error is due to the fact that, for bisection, the true root can lie anywhere within the bracketing interval. The true and approximate errors are far apart when the interval happens to be centered on the true root. They are close when the true root falls at either end of the interval.

Although the approximate error does not provide an exact estimate of the true error, Fig. 5.7 suggests that ε_a captures the general downward trend of ε_t. In addition, the plot exhibits the extremely attractive characteristic that ε_a is always greater than ε_t. Thus, when

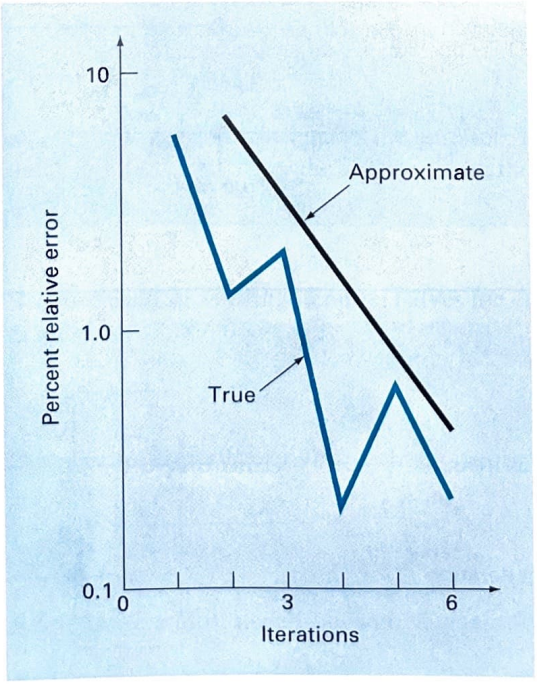


FIGURE 5.7
Errors for the bisection method. True and estimated errors are plotted versus the number of iterations.

ε_a falls below ε_s, the computation could be terminated with confidence that the root is known to be at least as accurate as the prespecified acceptable level.

Although it is always dangerous to draw general conclusions from a single example, it can be demonstrated that ε_a will always be greater than ε_t for the bisection method. This is because each time an approximate root is located using bisection as $x_r = (x_l + x_u)/2$, we know that the true root lies somewhere within an interval of $(x_u - x_l)/2 = \Delta x/2$. Therefore, the root must lie within $\pm\Delta x/2$ of our estimate (Fig. 5.8). For instance, when Example 5.3 was terminated, we could make the definitive statement that

$$x_r = 14.5 \pm 0.5$$

Because $\Delta x/2 = x_r^{\text{new}} - x_r^{\text{old}}$ (Fig. 5.9), Eq. (5.2) provides an exact upper bound on the true error. For this bound to be exceeded, the true root would have to fall outside the bracketing interval, which, by definition, could never occur for the bisection method. As illustrated in a subsequent example (Example 5.7), other root-locating techniques do not always behave as nicely. Although bisection is generally slower than other methods, the

FIGURE 5.8
Three ways in which the interval may bracket the root. In (*a*) the true value lies at the center of the interval, whereas in (*b*) and (*c*) the true value lies near the extreme. Notice that the discrepancy between the true value and the midpoint of the interval never exceeds half the interval length, or $\Delta x/2$.

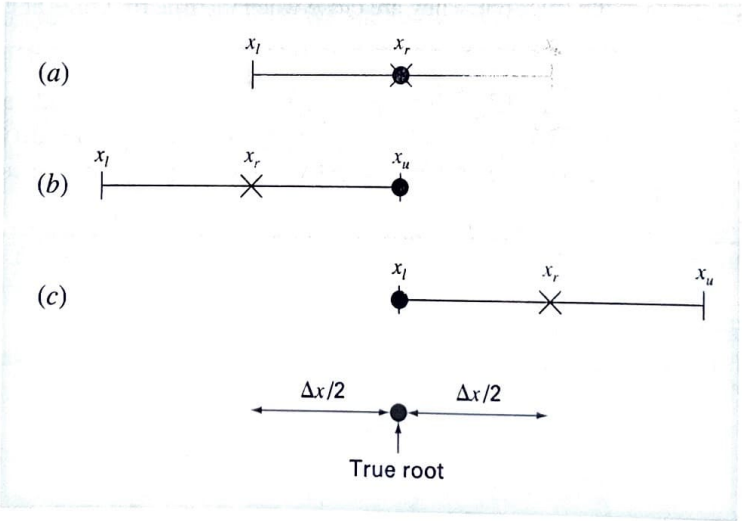
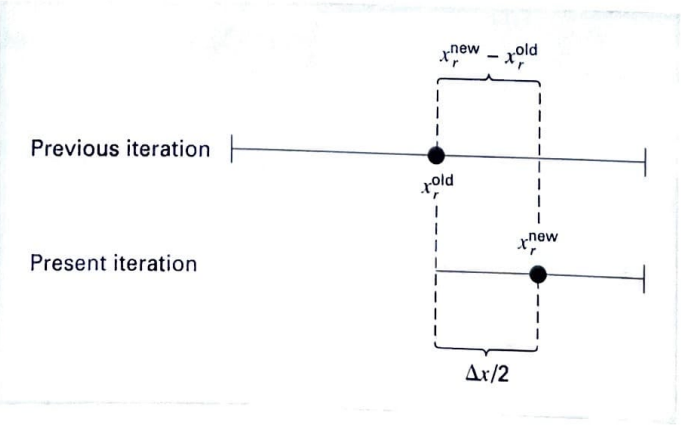


FIGURE 5.9
Graphical depiction of why the error estimate for bisection ($\Delta x/2$) is equivalent to the root estimate for the present iteration (x_r^{new}) minus the root estimate for the previous iteration (x_r^{old}).



neatness of its error analysis is certainly a positive aspect that could make it attractive for certain engineering applications.

Before proceeding to the computer program for bisection, we should note that the relationships (Fig. 5.9)

$$x_r^{\text{new}} - x_r^{\text{old}} = \frac{x_u - x_l}{2}$$

and

$$x_r^{\text{new}} = \frac{x_l + x_u}{2}$$

can be substituted into Eq. (5.2) to develop an alternative formulation for the approximate percent relative error

$$\varepsilon_a = \left| \frac{x_u - x_l}{x_u + x_l} \right| 100\% \tag{5.3}$$

This equation yields identical results to Eq. (5.2) for bisection. In addition, it allows us to calculate an error estimate on the basis of our initial guesses—that is, on our first iteration. For instance, on the first iteration of Example 5.2, an approximate error can be computed as

$$\varepsilon_a = \left| \frac{16 - 12}{16 + 12} \right| 100\% = 14.29\%$$

Another benefit of the bisection method is that the number of iterations required to attain an absolute error can be computed *a priori*—that is, before starting the iterations. This can be seen by recognizing that before starting the technique, the absolute error is

$$E_a^0 = x_u^0 - x_l^0 = \Delta x^0$$

where the superscript designates the iteration. Hence, before starting the method, we are at the "zero iteration." After the first iteration, the error becomes

$$E_a^1 = \frac{\Delta x^0}{2}$$

Because each succeeding iteration halves the error, a general formula relating the error and the number of iterations, n, is

$$E_a^n = \frac{\Delta x^0}{2^n} \tag{5.4}$$

If $E_{a,d}$ is the desired error, this equation can be solved for

$$n = \frac{\log(\Delta x^0 / E_{a,d})}{\log 2} = \log_2 \left(\frac{\Delta x^0}{E_{a,d}} \right) \tag{5.5}$$

Let us test the formula. For Example 5.4, the initial interval was $\Delta x_0 = 16 - 12 = 4$. After six iterations, the absolute error was

$$E_a = \frac{|14.875 - 14.75|}{2} = 0.0625$$

We can substitute these values into Eq. (5.5) to give

$$n = \frac{\log(4/0.0625)}{\log 2} = 6$$

Thus, if we knew beforehand that an error of less than 0.0625 was acceptable, the formula tells us that six iterations would yield the desired result.

Although we have emphasized the use of relative errors for obvious reasons, there will be cases where (usually through knowledge of the problem context) you will be able to specify an absolute error. For these cases, bisection along with Eq. (5.5) can provide a useful root-location algorithm. We will explore such applications in the end-of-chapter problems.

5.2.2 Bisection Algorithm

The algorithm in Fig. 5.5 can now be expanded to include the error check (Fig. 5.10). The algorithm employs user-defined functions to make root location and function evaluation more efficient. In addition, an upper limit is placed on the number of iterations. Finally, an error check is included to avoid division by zero during the error evaluation. Such would be the case when the bracketing interval is centered on zero. For this situation Eq. (5.2) becomes infinite. If this occurs, the program skips over the error evaluation for that iteration.

The algorithm in Fig. 5.10 is not user-friendly; it is designed strictly to come up with the answer. In Prob. 5.14 at the end of this chapter, you will have the task of making it easier to use and understand.

FIGURE 5.10
Pseudocode for function to implement bisection.

```
FUNCTION Bisect(xl, xu, es, imax, xr, iter, ea)
  iter = 0
  DO
    xrold = xr
    xr = (xl + xu) / 2
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = ABS((xr - xrold) / xr) * 100
    END IF
    test = f(xl) * f(xr)
    IF test < 0 THEN
      xu = xr
    ELSE IF test > 0 THEN
      xl = xr
    ELSE
      ea = 0
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Bisect = xr
END Bisect
```

5.2.3 Minimizing Function Evaluations

The bisection algorithm in Fig. 5.10 is just fine if you are performing a single root evaluation for a function that is easy to evaluate. However, there are many instances in engineering when this is not the case. For example, suppose that you develop a computer program that must locate a root numerous times. In such cases you could call the algorithm from Fig. 5.10 thousands and even millions of times in the course of a single run.

Further, in its most general sense, a univariate function is merely an entity that returns a single value in return for a single value you send to it. Perceived in this sense, functions are not always simple formulas like the one-line equations solved in the preceding examples in this chapter. For example, a function might consist of many lines of code that could take a significant amount of execution time to evaluate. In some cases, the function might even represent an independent computer program.

Because of both these factors, it is imperative that numerical algorithms minimize function evaluations. In this light, the algorithm from Fig. 5.10 is deficient. In particular, notice that in making two function evaluations per iteration, it recalculates one of the functions that was determined on the previous iteration.

Figure 5.11 provides a modified algorithm that does not have this deficiency. We have highlighted the lines that differ from Fig. 5.10. In this case, only the new function value at

FIGURE 5.11
Pseudocode for bisection subprogram which minimizes function evaluations.

```
FUNCTION Bisect(xl, xu, es, imax, xr, iter, ea)
  iter = 0
  fl = f(xl)
  DO
    xrold = xr
    xr = (xl + xu) / 2
    fr = f(xr)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = ABS((xr - xrold) / xr) * 100
    END IF
    test = fl * fr
    IF test < 0 THEN
      xu = xr
    ELSE IF test > 0 THEN
      xl = xr
      fl = fr
    ELSE
      ea = 0
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  Bisect = xr
END Bisect
```

the root estimate is calculated. Previously calculated values are saved and merely reassigned as the bracket shrinks. Thus, $n + 1$ function evaluations are performed, rather than $2n$.

5.3 THE FALSE-POSITION METHOD

Although bisection is a perfectly valid technique for determining roots, its "brute-force" approach is relatively inefficient. False position is an alternative based on a graphical insight.

A shortcoming of the bisection method is that, in dividing the interval from x_l to x_u into equal halves, no account is taken of the magnitudes of $f(x_l)$ and $f(x_u)$. For example, if $f(x_l)$ is much closer to zero than $f(x_u)$, it is likely that the root is closer to x_l than to x_u (Fig. 5.12). An alternative method that exploits this graphical insight is to join $f(x_l)$ and $f(x_u)$ by a straight line. The intersection of this line with the x axis represents an improved estimate of the root. The fact that the replacement of the curve by a straight line gives a "false position" of the root is the origin of the name, *method of false position*, or in Latin, *regula falsi*. It is also called the *linear interpolation method*.

Using similar triangles (Fig. 5.12), the intersection of the straight line with the x axis can be estimated as

$$\frac{f(x_l)}{x_r - x_l} = \frac{f(x_u)}{x_r - x_u} \tag{5.6}$$

which can be solved for (see Box 5.1 for details).

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)} \tag{5.7}$$

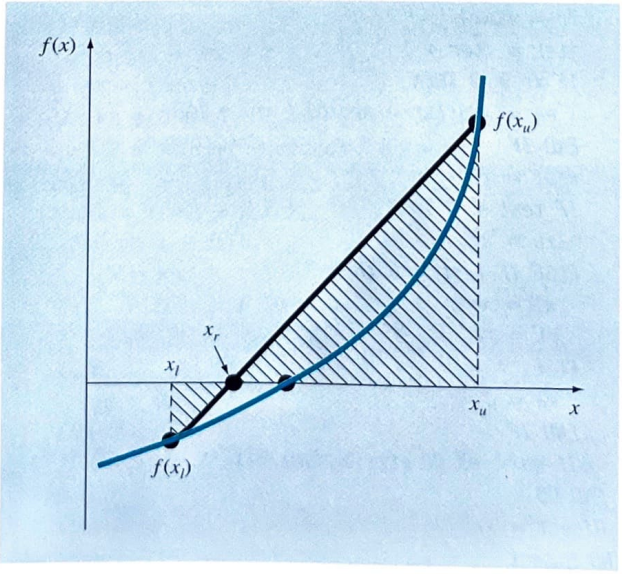


FIGURE 5.12
A graphical depiction of the method of false position. Similar triangles used to derive the formula for the method are shaded.

Box 5.1 Derivation of the Method of False Position

Cross-multiply Eq. (5.6) to yield

$$f(x_l)(x_r - x_u) = f(x_u)(x_r - x_l)$$

Collect terms and rearrange:

$$x_r\,[f(x_l) - f(x_u)] = x_u\,f(x_l) - x_l\,f(x_u)$$

Divide by $f(x_l) - f(x_u)$:

$$x_r = \frac{x_u\,f(x_l) - x_l\,f(x_u)}{f(x_l) - f(x_u)} \tag{B5.1.1}$$

This is one form of the method of false position. Note that it allows the computation of the root x_r as a function of the lower and upper guesses x_l and x_u. It can be put in an alternative form by expanding it:

$$x_r = \frac{x_u\,f(x_l)}{f(x_l) - f(x_u)} - \frac{x_l\,f(x_u)}{f(x_l) - f(x_u)}$$

then adding and subtracting x_u on the right-hand side:

$$x_r = x_u + \frac{x_u\,f(x_l)}{f(x_l) - f(x_u)} - x_u - \frac{x_l\,f(x_u)}{f(x_l) - f(x_u)}$$

Collecting terms yields

$$x_r = x_u + \frac{x_u\,f(x_u)}{f(x_l) - f(x_u)} - \frac{x_l\,f(x_u)}{f(x_l) - f(x_u)}$$

or

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

which is the same as Eq. (5.7). We use this form because it involves one less function evaluation and one less multiplication than Eq. (B5.1.1). In addition, it is directly comparable with the secant method which will be discussed in Chap. 6.

This is the *false-position formula*. The value of x_r computed with Eq. (5.7) then replaces whichever of the two initial guesses, x_l or x_u, yields a function value with the same sign as $f(x_r)$. In this way, the values of x_l and x_u always bracket the true root. The process is repeated until the root is estimated adequately. The algorithm is identical to the one for bisection (Fig. 5.5) with the exception that Eq. (5.7) is used for step 2. In addition, the same stopping criterion [Eq. (5.2)] is used to terminate the computation.

EXAMPLE 5.5 False Position

Problem Statement. Use the false-position method to determine the root of the same equation investigated in Example 5.1 [Eq. (E5.1.1)].

Solution. As in Example 5.3, initiate the computation with guesses of $x_l = 12$ and $x_u = 16$.

First iteration:

$$x_l = 12 \qquad f(x_l) = 6.0699$$
$$x_u = 16 \qquad f(x_u) = -2.2688$$
$$x_r = 16 - \frac{-2.2688(12 - 16)}{6.0669 - (-2.2688)} = 14.9113$$

which has a true relative error of 0.89 percent.

Second iteration:

$$f(x_l)\,f(x_r) = -1.5426$$

Therefore, the root lies in the first subinterval, and x_r becomes the upper limit for the next iteration, $x_u = 14.9113$:

$$x_l = 12 \qquad f(x_l) = 6.0699$$

$$x_u = 14.9113 \qquad f(x_u) = -0.2543$$

$$x_r = 14.9113 - \frac{-0.2543(12 - 14.9113)}{6.0669 - (-0.2543)} = 14.7942$$

which has true and approximate relative errors of 0.09 and 0.79 percent. Additional iterations can be performed to refine the estimate of the roots.

A feeling for the relative efficiency of the bisection and false-position methods can be appreciated by referring to Fig. 5.13, where we have plotted the true percent relative errors for Examples 5.4 and 5.5. Note how the error for false position decreases much faster than for bisection because of the more efficient scheme for root location in the false-position method.

Recall in the bisection method that the interval between x_l and x_u grew smaller during the course of a computation. The interval, as defined by $\Delta x/2 = |x_u - x_l|/2$ for the first iteration, therefore provided a measure of the error for this approach. This is not the case

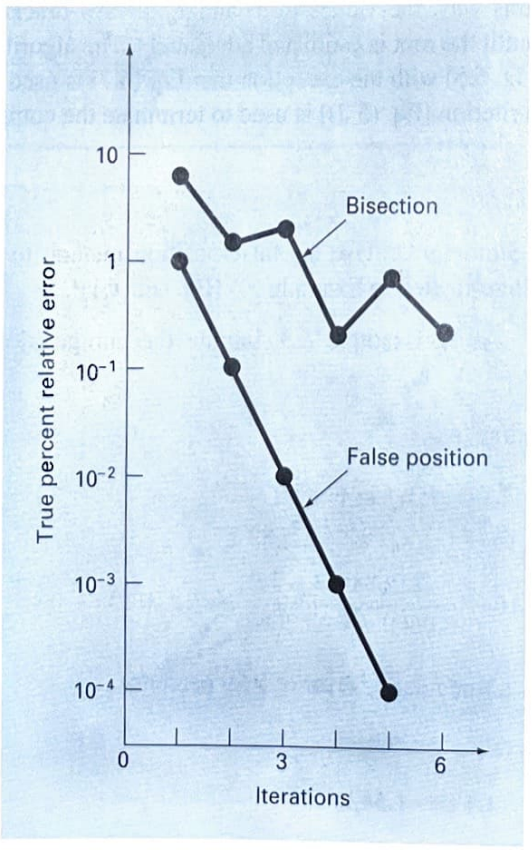


FIGURE 5.13
Comparison of the relative errors of the bisection and the false-position methods.

for the method of false position because one of the initial guesses may stay fixed throughout the computation as the other guess converges on the root. For instance, in Example 5.6 the lower guess x_l remained at 12 while x_u converged on the root. For such cases, the interval does not shrink but rather approaches a constant value.

Example 5.6 suggests that Eq. (5.2) represents a very conservative error criterion. In fact, Eq. (5.2) actually constitutes an approximation of the discrepancy of the previous iteration. This is because for a case such as Example 5.6, where the method is converging quickly (for example, the error is being reduced nearly an order of magnitude per iteration), the root for the present iteration x_r^{new} is a much better estimate of the true value than the result of the previous iteration x_r^{old}. Thus, the quantity in the numerator of Eq. (5.2) actually represents the discrepancy of the previous iteration. Consequently, we are assured that satisfaction of Eq. (5.2) ensures that the root will be known with greater accuracy than the prescribed tolerance. However, as described in the next section, there are cases where false position converges slowly. For these cases, Eq. (5.2) becomes unreliable, and an alternative stopping criterion must be developed.

5.3.1 Pitfalls of the False-Position Method

Although the false-position method would seem to always be the bracketing method of preference, there are cases where it performs poorly. In fact, as in the following example, there are certain cases where bisection yields superior results.

EXAMPLE 5.6 A Case Where Bisection Is Preferable to False Position

Problem Statement. Use bisection and false position to locate the root of

$$f(x) = x^{10} - 1$$

between $x = 0$ and 1.3.

Solution. Using bisection, the results can be summarized as

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.65	100.0	35
2	0.65	1.3	0.975	33.3	2.5
3	0.975	1.3	1.1375	14.3	13.8
4	0.975	1.1375	1.05625	7.7	5.6
5	0.975	1.05625	1.015625	4.0	1.6

Thus, after five iterations, the true error is reduced to less than 2 percent. For false position, a very different outcome is obtained:

Iteration	x_l	x_u	x_r	ε_a (%)	ε_t (%)
1	0	1.3	0.09430		90.6
2	0.09430	1.3	0.18176	48.1	81.8
3	0.18176	1.3	0.26287	30.9	73.7
4	0.26287	1.3	0.33811	22.3	66.2
5	0.33811	1.3	0.40788	17.1	59.2

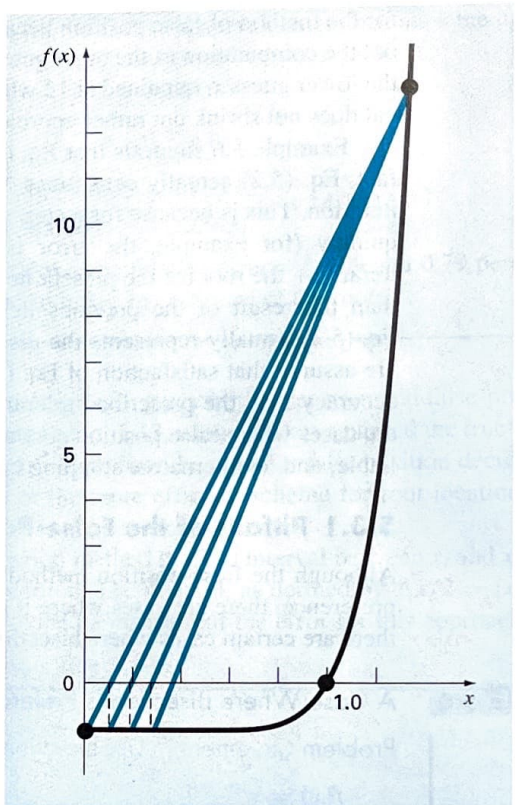


FIGURE 5.14
Plot of $f(x) = x^{10} - 1$, illustrating slow convergence of the false-position method.

After five iterations, the true error has only been reduced to about 59 percent. In addition, note that $\varepsilon_a < \varepsilon_t$. Thus, the approximate error is misleading. Insight into these results can be gained by examining a plot of the function. As in Fig. 5.14, the curve violates the premise upon which false position was based—that is, if $f(x_l)$ is much closer to zero than $f(x_u)$, then the root is closer to x_l than to x_u (recall Fig. 5.12). Because of the shape of the present function, the opposite is true.

The forgoing example illustrates that blanket generalizations regarding root-location methods are usually not possible. Although a method such as false position is often superior to bisection, there are invariably cases that violate this general conclusion. Therefore, in addition to using Eq. (5.2), the results should always be checked by substituting the root estimate into the original equation and determining whether the result is close to zero. Such a check should be incorporated into all computer programs for root location.

The example also illustrates a major weakness of the false-position method: its one-sidedness. That is, as iterations are proceeding, one of the bracketing points will tend to

stay fixed. This can lead to poor convergence, particularly for functions with significant curvature. The following section provides a remedy.

5.3.2 Modified False Position

One way to mitigate the "one-sided" nature of false position is to have the algorithm detect when one of the bounds is stuck. If this occurs, the function value at the stagnant bound can be divided in half. This is called the *modified false-position method.*

The algorithm in Fig. 5.15 implements this strategy. Notice how counters are used to determine when one of the bounds stays fixed for two iterations. If this occurs, the function value at this stagnant bound is halved.

The effectiveness of this algorithm can be demonstrated by applying it to Example 5.6. If a stopping criterion of 0.01% is used, the bisection and standard false-position methods

FIGURE 5.15
Pseudocode for the modified false-position method.

```
FUNCTION ModFalsePos(xl, xu, es, imax, xr, iter, ea)
  iter = 0
  fl = f(xl)
  fu = f(xu)
  DO
    xrold = xr
    xr = xu - fu * (xl - xu) / (fl - fu)
    fr = f(xr)
    iter = iter + 1
    IF xr <> 0 THEN
      ea = Abs((xr - xrold) / xr) * 100
    END IF
    test = fl * fr
    IF test < 0 THEN
      xu = xr
      fu = f(xu)
      iu = 0
      il = il + 1
      If il ≥ 2 THEN fl = fl / 2
    ELSE IF test > 0 THEN
      xl = xr
      fl = f(xl)
      il = 0
      iu = iu + 1
      IF iu ≥ 2 THEN fu = fu / 2
    ELSE
      ea = 0
    END IF
    IF ea < es OR iter ≥ imax THEN EXIT
  END DO
  ModFalsePos = xr
END ModFalsePos
```

would converge in 14 and 39 iterations, respectively. In contrast, the modified false-position method would converge in 12 iterations. Thus, for this example, it is somewhat more efficient than bisection and is vastly superior to the unmodified false-position method.

5.4 INCREMENTAL SEARCHES AND DETERMINING INITIAL GUESSES

Besides checking an individual answer, you must determine whether all possible roots have been located. As mentioned previously, a plot of the function is usually very useful in guiding you in this task. Another option is to incorporate an incremental search at the beginning of the computer program. This consists of starting at one end of the region of interest and then making function evaluations at small increments across the region. When the function changes sign, it is assumed that a root falls within the increment. The x values at the beginning and the end of the increment can then serve as the initial guesses for one of the bracketing techniques described in this chapter.

A potential problem with an incremental search is the choice of the increment length. If the length is too small, the search can be very time consuming. On the other hand, if the length is too great, there is a possibility that closely spaced roots might be missed (Fig. 5.16). The problem is compounded by the possible existence of multiple roots. A partial remedy for such cases is to compute the first derivative of the function $f'(x)$ at the beginning and the end of each interval. If the derivative changes sign, it suggests that a minimum or maximum may have occurred and that the interval should be examined more closely for the existence of a possible root.

Although such modifications or the employment of a very fine increment can alleviate the problem, it should be clear that brute-force methods such as incremental search are not foolproof. You would be wise to supplement such automatic techniques with any other information that provides insight into the location of the roots. Such information can be found in plotting and in understanding the physical problem from which the equation originated.

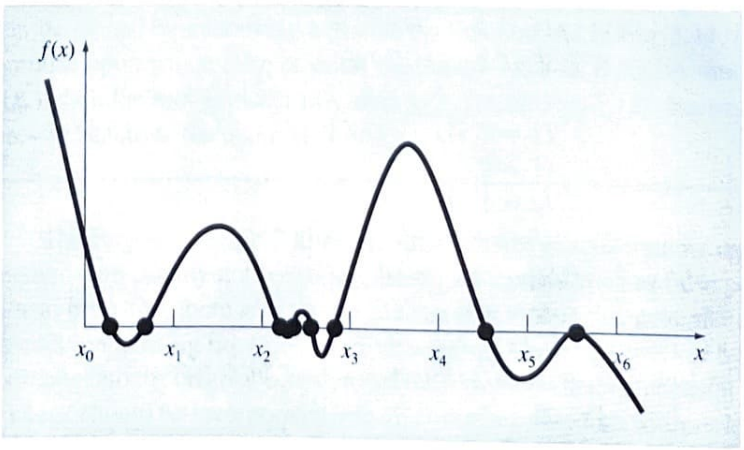


FIGURE 5.16
Cases where roots could be missed because the increment length of the search procedure is too large. Note that the last root on the right is multiple and would be missed regardless of increment length.