# 5

# Roots of Equations: Bracketing Methods

## CHAPTER OBJECTIVES

The primary objective of this chapter is to acquaint you with bracketing methods for finding the root of a single nonlinear equation. Specific objectives and topics covered are

- Understanding what roots problem are and where they occur in engineering and science.
- Knowing how to determine a root graphically.
- Understanding the incremental search method and its shortcomings.
- Knowing how to solve a roots problem with the bisection method.
- Knowing how to estimate the error of bisection and why it differs from error estimates for other types of root location algorithms.
- Understanding false position and how it differs from bisection.

## YOU'VE GOT A PROBLEM

Medical studies have established that a bungee jumper's chances of sustaining a significant vertebrae injury increase significantly if the free-fall velocity exceeds 36 m/s after 4 s of free fall. Your boss at the bungee-jumping company wants you to determine the mass at which this criterion is exceeded given a drag coefficient of 0.25 kg/m.

You know from your previous studies that the following analytical solution can be used to predict fall velocity as a function of time:

$$v(t) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) \tag{5.1}$$

Try as you might, you cannot manipulate this equation to explicitly solve for $m$—that is, you cannot isolate the mass on the left side of the equation.

An alternative way of looking at the problem involves subtracting $v(t)$ from both sides to give a new function:

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh\left(\sqrt{\frac{gc_d}{m}}t\right) - v(t) \tag{5.2}$$

Now we can see that the answer to the problem is the value of $m$ that makes the function equal to zero. Hence, we call this a "roots" problem. This chapter will introduce you to how the computer is used as a tool to obtain such solutions.

## 5.1 INTRODUCTION AND BACKGROUND

### 5.1.1 What Are Roots?

Years ago, you learned to use the quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{5.3}$$

to solve

$$f(x) = ax^2 + bx + c = 0 \tag{5.4}$$

The values calculated with Eq. (5.3) are called the "roots" of Eq. (5.4). They represent the values of $x$ that make Eq. (5.4) equal to zero. For this reason, roots are sometimes called the *zeros* of the equation.

Although the quadratic formula is handy for solving Eq. (5.4), there are many other functions for which the root cannot be determined so easily. Before the advent of digital computers, there were a number of ways to solve for the roots of such equations. For some cases, the roots could be obtained by direct methods, as with Eq. (5.3). Although there were equations like this that could be solved directly, there were many more that could not. In such instances, the only alternative is an approximate solution technique.

One method to obtain an approximate solution is to plot the function and determine where it crosses the $x$ axis. This point, which represents the $x$ value for which $f(x) = 0$, is the root. Although graphical methods are useful for obtaining rough estimates of roots, they are limited because of their lack of precision. An alternative approach is to use *trial and error*. This "technique" consists of guessing a value of $x$ and evaluating whether $f(x)$ is zero. If not (as is almost always the case), another guess is made, and $f(x)$ is again evaluated to determine whether the new value provides a better estimate of the root. The process is repeated until a guess results in an $f(x)$ that is close to zero.

Such haphazard methods are obviously inefficient and inadequate for the requirements of engineering practice. Numerical methods represent alternatives that are also approximate but employ systematic strategies to home in on the true root. As elaborated in the following pages, the combination of these systematic methods and computers makes the solution of most applied roots-of-equations problems a simple and efficient task.

### 5.1.2 Roots of Equations and Engineering Practice

Although they arise in other problem contexts, roots of equations frequently occur in the area of engineering design. Table 5.1 lists a number of fundamental principles that are routinely used in design work. As introduced in Chap. 1, mathematical equations or models

**TABLE 5.1** Fundamental principles used in engineering design problems.

| Fundamental Principle | Dependent Variable | Independent Variable | Parameters |
|---|---|---|---|
| Heat balance | Temperature | Time and position | Thermal properties of material, system geometry |
| Mass balance | Concentration or quantity of mass | Time and position | Chemical behavior of material, mass transfer, system geometry |
| Force balance | Magnitude and direction of forces | Time and position | Strength of material, structural properties, system geometry |
| Energy balance | Changes in kinetic and potential energy | Time and position | Thermal properties, mass of material, system geometry |
| Newton's laws of motion | Acceleration, velocity, or location | Time and position | Mass of material, system geometry, dissipative parameters |
| Kirchhoff's laws | Currents and voltages | Time | Electrical properties (resistance, capacitance, inductance) |

derived from these principles are employed to predict dependent variables as a function of independent variables, forcing functions, and parameters. Note that in each case, the dependent variables reflect the state or performance of the system, whereas the parameters represent its properties or composition.

An example of such a model is the equation for the bungee jumper's velocity. If the parameters are known, Eq. (5.1) can be used to predict the jumper's velocity. Such computations can be performed directly because $v$ is expressed *explicitly* as a function of the model parameters. That is, it is isolated on one side of the equal sign.

However, as posed at the start of the chapter, suppose that we had to determine the mass for a jumper with a given drag coefficient to attain a prescribed velocity in a set time period. Although Eq. (5.1) provides a mathematical representation of the interrelationship among the model variables and parameters, it cannot be solved explicitly for mass. In such cases, $m$ is said to be *implicit*.

This represents a real dilemma, because many engineering design problems involve specifying the properties or composition of a system (as represented by its parameters) to ensure that it performs in a desired manner (as represented by its variables). Thus, these problems often require the determination of implicit parameters.

The solution to the dilemma is provided by numerical methods for roots of equations. To solve the problem using numerical methods, it is conventional to reexpress Eq. (5.1) by subtracting the dependent variable $v$ from both sides of the equation to give Eq. (5.2). The value of $m$ that makes $f(m) = 0$ is, therefore, the root of the equation. This value also represents the mass that solves the design problem.

The following pages deal with a variety of numerical and graphical methods for determining roots of relationships such as Eq. (5.2). These techniques can be applied to many other problems confronted routinely in engineering and science.

## 5.2 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.

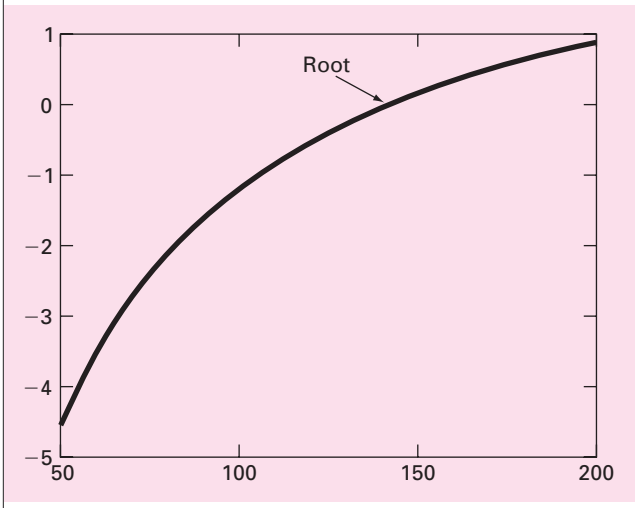
EXAMPLE 5.1

## The Graphical Approach

**Problem Statement.** Use the graphical approach to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is 9.81 m/s$^2$.

**Solution.** The following MATLAB session sets up a plot of Eq. (5.2) versus mass:

```
>> cd = 0.25; g = 9.81; v = 36; t = 4;
>> mp = linspace(50,200);
>> fp = sqrt(g*mp/cd).*tanh(sqrt(g*cd./mp)*t)-v;
>> plot(mp,fp),grid
```




The function crosses the *m* axis between 140 and 150 kg. Visual inspection of the plot provides a rough estimate of the root of 145 kg (about 320 lb). The validity of the graphical estimate can be checked by substituting it into Eq. (5.2) to yield

```
>> sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)-v

ans =
    0.0456
```

which is close to zero. It can also be checked by substituting it into Eq. (5.1) along with the parameter values from this example to give

```
>> sqrt(g*145/cd)*tanh(sqrt(g*cd/145)*t)

ans =
   36.0456
```

which is close to the desired fall velocity of 36 m/s.

Graphical techniques are of limited practical value because they are not very 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 chapter.

Aside from providing rough estimates of the root, graphical interpretations are useful for understanding the properties of the functions and anticipating the pitfalls of the numerical methods. For example, Fig. 5.1 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 5.1*b* depicts the case where a single root is bracketed by negative and positive values of $f(x)$. However, Fig. 5.1*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. 5.1*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. 5.2*a*) and discontinuous functions (Fig. 5.2*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*. Although they are beyond the scope of this book, there are special techniques that are expressly designed to locate multiple roots (Chapra and Canale, 2002).

The existence of cases of the type depicted in Fig. 5.2 makes it difficult to develop foolproof 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 sections are extremely useful for solving many problems confronted routinely by engineers, scientists, and applied mathematicians.

## 5.3 BRACKETING METHODS AND INITIAL GUESSES

If you had a roots problem in the days before computing, you'd often be told to use "trial and error" to come up with the root. That is, you'd repeatedly make guesses until the function was sufficiently close to zero. The process was greatly facilitated by the advent of software tools such as spreadsheets. By allowing you to make many guesses rapidly, such tools can actually make the trial-and-error approach attractive for some problems.

But, for many other problems, it is preferable to have methods that come up with the correct answer automatically. Interestingly, as with trial and error, these approaches require an initial "guess" to get started. Then they systematically home in on the root in an iterative fashion.

The two major classes of methods available are distinguished by the type of initial guess. They are

- *Bracketing methods.* As the name implies, these are based on two initial guesses that "bracket" the root—that is, are on either side of the root.
- *Open methods.* These methods can involve one or more initial guesses, but there is no need for them to bracket the root.

For well-posed problems, the bracketing methods always work but converge slowly (i.e., they typically take more iterations to home in on the answer). In contrast, the open methods do not always work (i.e., they can diverge), but when they do they usually converge quicker.

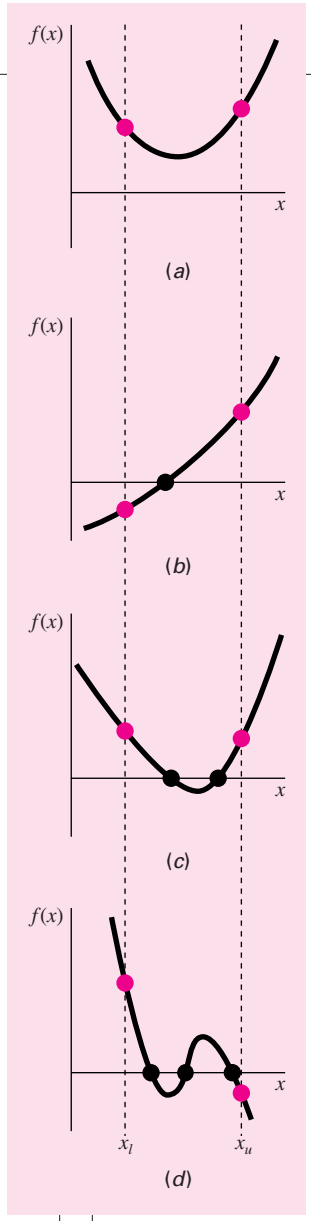



**FIGURE 5.1**
Illustration of a number of general ways that a root may occur in an interval prescribed by a lower bound $x_l$ and an upper bound $x_u$. Parts (*a*) and (*c*) indicate that if both $f(x_l)$ and $f(x_u)$ have the same sign, either there will be no roots or there will be an even number 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.

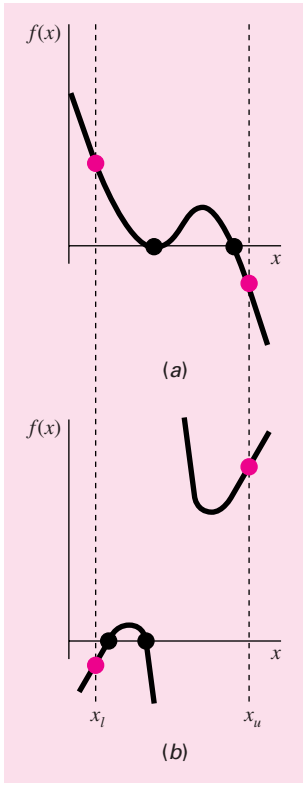



**FIGURE 5.2**
Illustration of some exceptions to the general cases depicted in Fig. 5.1. (*a*) Multiple roots that occur when the function is tangential to the $x$ axis. For this case, although the end points are of opposite signs, there are an even number of axis interceptions for the interval. (*b*) Discontinuous functions where end points of opposite sign bracket an even number of roots. Special strategies are required for determining the roots for these cases.

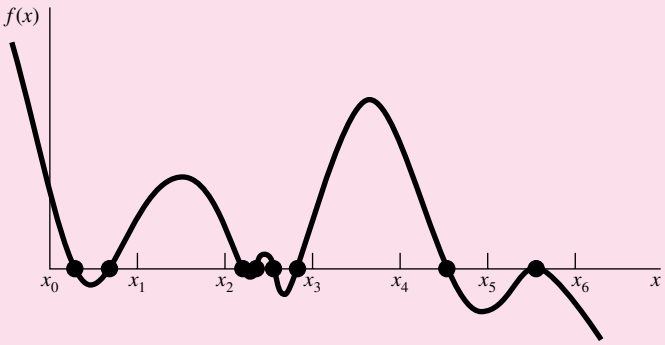



**FIGURE 5.3**
Cases where roots could be missed because the incremental length of the search procedure is too large. Note that the last root on the right is multiple and would be missed regardless of the increment length.

In both cases, initial guesses are required. These may naturally arise from the physical context you are analyzing. However, in other cases, good initial guesses may not be obvious. In such cases, automated approaches to obtain guesses would be useful. The following section describes one such approach, the incremental search.

### 5.3.1 Incremental Search

When applying the graphical technique in Example 5.1, you observed 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,

$$f(x_l)\,f(x_u) < 0$$

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. 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. 5.3). The problem is compounded by the possible existence of multiple roots.

An M-file can be developed[1] that implements an incremental search to locate the roots of a function `func` within the range from `xmin` to `xmax` (Fig. 5.4). An optional argument `ns` allows the user to specify the number of intervals within the range. If `ns` is omitted, it is automatically set to 50. A `for` loop is used to step through each interval. In the event that a sign change occurs, the upper and low bounds are stored in an array `xb`.

[1] This function is a modified version of an M-file originally presented by Recktenwald (2000).

```
function xb = incsearch(func,xmin,xmax,ns)
% incsearch(func,xmin,xmax,ns):
%   finds brackets of x that contain sign changes of
%   a function on an interval
% input:
%   func = name of function
%   xmin, xmax = endpoints of interval
%   ns = (optional) number of subintervals along x
%        used to search for brackets
% output:
%   xb(k,1) is the lower bound of the kth sign change
%   xb(k,2) is the upper bound of the kth sign change
%   If no brackets found, xb = [].

if nargin < 4, ns = 50; end %if ns blank set to 50

% Incremental search
x = linspace(xmin,xmax,ns);
f = feval(func,x);
nb = 0; xb = []; %xb is null unless sign change detected
for k = 1:length(x)-1
  if sign(f(k)) ~= sign(f(k+1)) %check for sign change
    nb = nb + 1;
    xb(nb,1) = x(k);
    xb(nb,2) = x(k+1);
  end
end

if isempty(xb)    %display that no brackets were found
  disp('no brackets found')
  disp('check interval or increase ns')
else
  disp('number of brackets:') %display number of brackets
  disp(nb)
end
```

**FIGURE 5.4**
An M-file to implement an incremental search.

EXAMPLE 5.2 Incremental Search

Problem Statement. Use the M-file `incsearch` (Fig. 5.4) to identify brackets within the interval [3, 6] for the function:

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

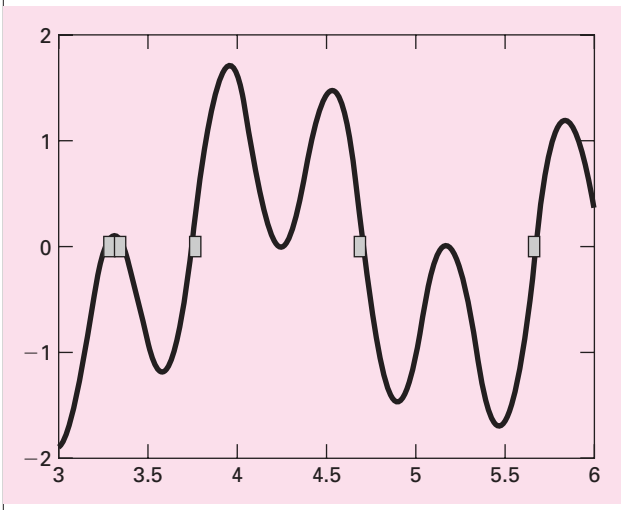
Solution. The MATLAB session using the default number of intervals (50) is

```
>> incsearch(inline('sin(10*x)+cos(3*x)'),3,6)
number of possible roots:
     5

ans =

    3.2449    3.3061
    3.3061    3.3673
    3.7347    3.7959
    4.6531    4.7143
    5.6327    5.6939
```

A plot of the function along with the root locations is shown here.



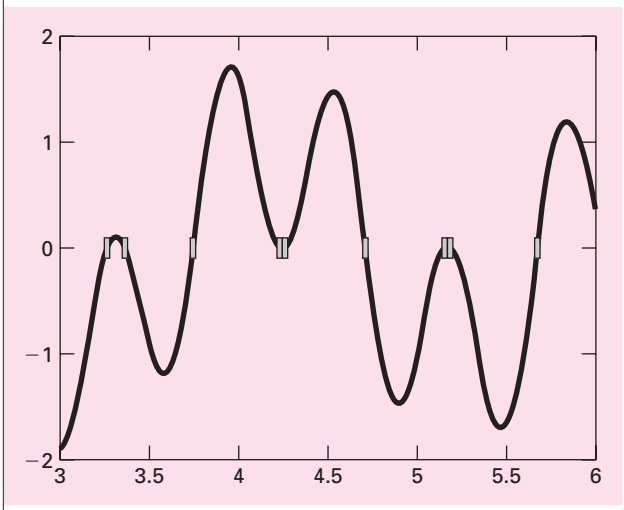

Although five sign changes are detected, because the subintervals are too wide, the function misses possible roots at $x \cong 4.25$ and 5.2. These possible roots look like they might be double roots. However, by using the zoom in tool, it is clear that each represents two real roots that are very close together. The function can be run again with more subintervals with the result that all nine sign changes are located

```
>> incsearch(inline('sin(10*x)+cos(3*x)'),3,6,100)

number of possible roots:
     9

ans =
    3.2424    3.2727
    3.3636    3.3939
    3.7273    3.7576
    4.2121    4.2424
    4.2424    4.2727
    4.6970    4.7273
```

```
5.1515    5.1818
5.1818    5.2121
5.6667    5.6970
```




The foregoing example illustrates 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 by plotting the function and through understanding the physical problem from which the equation originated.

## 5.4 BISECTION

The *bisection method* is a variation of the 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 within the subinterval where the sign change occurs. The subinterval then becomes the interval for the next iteration. The process is repeated until the root is known to the required precision. A graphical depiction of the method is provided in Fig. 5.5. The following example goes through the actual computations involved in the method.

EXAMPLE 5.3 The Bisection Method

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, $m$) that give values for $f(m)$ with different signs. From the graphical solution in Example 5.1, we can see that the function changes sign between values of 50 and 200. The plot obviously suggests better initial guesses, say 140 and 150, but for illustrative purposes let's assume we don't have the benefit of the plot and have made conservative guesses.

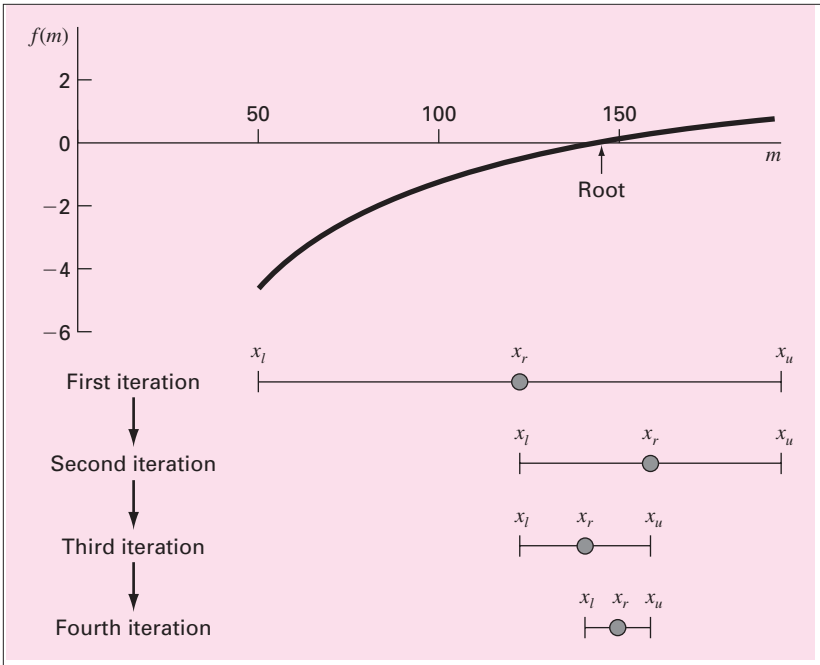



**FIGURE 5.5**
A graphical depiction of the bisection method. This plot corresponds to the first four iterations from Example 5.3.

Therefore, the initial estimate of the root $x_r$ lies at the midpoint of the interval

$$x_r = \frac{50 + 200}{2} = 125$$

Note that the exact value of the root is 142.7376. This means that the value of 125 calculated here has a true percent relative error of

$$|\varepsilon_t| = \left|\frac{142.7376 - 125}{142.7376}\right| \times 100\% = 12.43\%$$

Next we compute the product of the function value at the lower bound and at the midpoint:

$$f(50)\,f(125) = -4.579(-0.409) = 1.871$$

which is greater than zero, and hence no sign change occurs between the lower bound and the midpoint. Consequently, the root must be located in the upper interval between 125 and 200. Therefore, we create a new interval by redefining the lower bound as 125.

At this point, the new interval extends from $x_l = 125$ to $x_u = 200$. A revised root estimate can then be calculated as

$$x_r = \frac{125 + 200}{2} = 162.5$$

which represents a true percent error of $|\varepsilon_t| = 13.85\%$. The process can be repeated to obtain refined estimates. For example,

$$f(125)f(162.5) = -0.409(0.359) = -0.147$$

Therefore, the root is now in the lower interval between 125 and 162.5. The upper bound is redefined as 162.5, and the root estimate for the third iteration is calculated as

$$x_r = \frac{125 + 162.5}{2} = 143.75$$

which represents a percent relative error of $\varepsilon_t = 0.709\%$. The method can be repeated until the result is accurate enough to satisfy your needs.

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 error falls below some prespecified level. For instance, in Example 5.3, the true relative error dropped from 12.43 to 0.709% during the course of the computation. We might decide that we should terminate when the error drops below, say, 0.5%. 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. One way to do this is by estimating an approximate percent relative error as in [recall Eq. (4.5)]

$$|\varepsilon_a| = \left| \frac{x_r^{\text{new}} - x_r^{\text{old}}}{x_r^{\text{new}}} \right| 100\% \tag{5.5}$$

where $x_r^{\text{new}}$ is the root for the present iteration and $x_r^{\text{old}}$ is the root from the previous iteration. When $\varepsilon_a$ becomes less than a prespecified stopping criterion $\varepsilon_s$, 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. (5.5) to compute the errors.

Solution. The results of the first two iterations for Example 5.3 were 125 and 162.5. Substituting these values into Eq. (5.5) yields

$$|\varepsilon_a| = \left| \frac{162.5 - 125}{162.5} \right| 100\% = 23.08\%$$

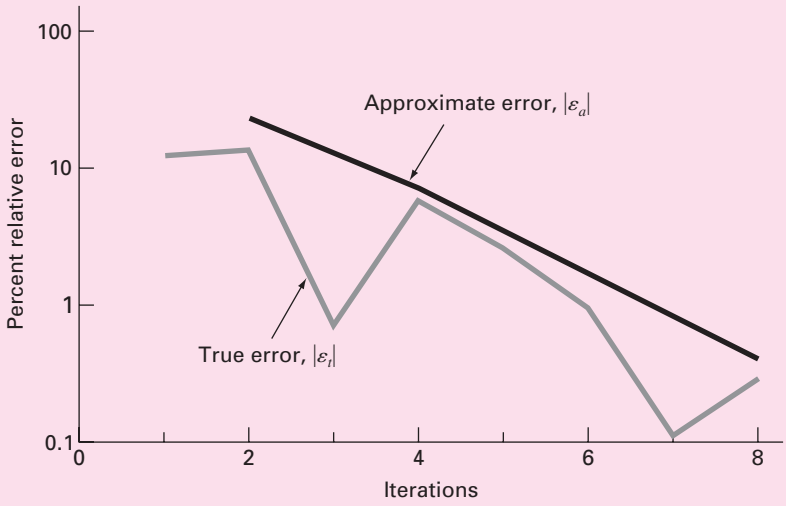
Recall that the true percent relative error for the root estimate of 162.5 was 13.85%. Therefore, $|\varepsilon_a|$ is greater than $|\varepsilon_t|$. This behavior is manifested for the other iterations:

| Iteration | $x_l$ | $x_u$ | $x_r$ | $\lvert\varepsilon_a\rvert$ (%) | $\lvert\varepsilon_t\rvert$ (%) |
|---|---|---|---|---|---|
| 1 | 50 | 200 | 125 | | 12.43 |
| 2 | 125 | 200 | 162.5 | 23.08 | 13.85 |
| 3 | 125 | 162.5 | 143.75 | 13.04 | 0.71 |
| 4 | 125 | 143.75 | 134.375 | 6.98 | 5.86 |
| 5 | 134.375 | 143.75 | 139.0625 | 3.37 | 2.58 |
| 6 | 139.0625 | 143.75 | 141.4063 | 1.66 | 0.93 |
| 7 | 141.4063 | 143.75 | 142.5781 | 0.82 | 0.11 |
| 8 | 142.5781 | 143.75 | 143.1641 | 0.41 | 0.30 |

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

These results are summarized in Fig. 5.6. 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.

**FIGURE 5.6**
Errors for the bisection method. True and estimated errors are plotted versus the number of iterations.




Although the approximate error does not provide an exact estimate of the true error, Fig. 5.6 suggests that $|\varepsilon_a|$, captures the general downward trend of $|\varepsilon_t|$. In addition, the plot exhibits the extremely attractive characteristic that $|\varepsilon_a|$ is always greater than $|\varepsilon_t|$. Thus, when $|\varepsilon_a|$ falls below $\varepsilon_s$, the computation could be terminated with confidence that the root is known to be at least as accurate as the prespecified acceptable level.

While it is dangerous to draw general conclusions from a single example, it can be demonstrated that $|\varepsilon_a|$ will always be greater than $|\varepsilon_t|$ for bisection. This is due to the fact

that 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 $\Delta x = x_u - x_l$. Therefore, the root must lie within $\pm\Delta x/2$ of our estimate. For instance, when Example 5.4 was terminated, we could make the definitive statement that

$$x_r = 143.1641 \pm \frac{143.7500 - 142.5781}{2} = 143.1641 \pm 0.5859$$

In essence, Eq. (5.5) provides an 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 bisection. Other root-locating techniques do not always behave as nicely. Although bisection is generally slower than other methods, the neatness of its error analysis is a positive feature that makes it attractive for certain engineering and scientific applications.

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 computation. This can be seen by recognizing that before starting the technique, the absolute error is

$$E_a^0 = \frac{x_u^0 - x_l^0}{2} = \frac{\Delta x^0}{2}$$

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

$$E_a^1 = \frac{\Delta x^0}{4}$$

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

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

If $E_{a,d}$ is the desired error, this equation can be solved for[2]

$$n = 1 + \frac{\log(\Delta x^0/E_{a,d})}{\log 2} = 1 + \log_2\left(\frac{\Delta x^0}{E_{a,d}}\right) \tag{5.6}$$

Let's test the formula. For Example 5.4, the initial interval was $\Delta x_0 = 200 - 50 = 150$. After eight iterations, the absolute error was

$$E_a = \frac{|143.7500 - 142.5781|}{2} = 0.5859$$

We can substitute these values into Eq. (5.6) to give

$$n = 1 + \log_2\left(\frac{150/0.5859}{2}\right) = 8$$

Thus, if we knew beforehand that an error of less than 0.5859 was acceptable, the formula tells us that eight 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

[2] MATLAB provides the `log2` function to evaluate the base-2 logarithm directly. If the pocket calculator or computer language you are using does not include the base-2 logarithm as an intrinsic function, this equation shows a handy way to compute it. In general, $\log_b(x) = \log(x)/\log(b)$.

specify an absolute error. For these cases, bisection along with Eq. (5.6) can provide a useful root location algorithm.

### 5.4.1 MATLAB M-file: `bisection`

An M-file to implement bisection is displayed in Fig. 5.7. It is passed the function (`func`) along with lower (`xl`) and upper (`xu`) guesses. In addition an optional stopping criterion

**FIGURE 5.7**
An M-file to implement the bisection method.

```
function root = bisection(func,xl,xu,es,maxit)
% bisection(xl,xu,es,maxit):
%   uses bisection method to find the root of a function
% input:
%   func = name of function
%   xl, xu = lower and upper guesses
%   es = (optional) stopping criterion (%)
%   maxit = (optional) maximum allowable iterations
% output:
%   root = real root

if func(xl)*func(xu)>0  %if guesses do not bracket a sign
  error('no bracket')   %change, display an error message
  return                %and terminate
end
% if necessary, assign default values
if nargin<5, maxit = 50; end     %if maxit blank set to 50
if nargin<4, es = 0.001; end     %if es blank set to 0.001

% bisection
iter = 0;
xr = xl;
while (1)
  xrold = xr;
  xr = (xl + xu)/2;
  iter = iter + 1;
  if xr ~= 0, ea = abs((xr - xrold)/xr) * 100; end
  test = func(xl)*func(xr);
  if test < 0
    xu = xr;
  elseif test > 0
    xl = xr;
  else
    ea = 0;
  end
  if ea <= es | iter >= maxit, break, end
end
root = xr;
```

(es) and maximum iterations (maxit) can be entered. The function first checks whether the initial guesses bracket a sign change. If not, an error estimate is displayed and the function is terminated. It also supplies default values if maxit and es are not supplied. Then a while . . . break loop is employed to implement the bisection algorithm until the approximate error falls below the stopping criterion or the iterations exceed maxit.

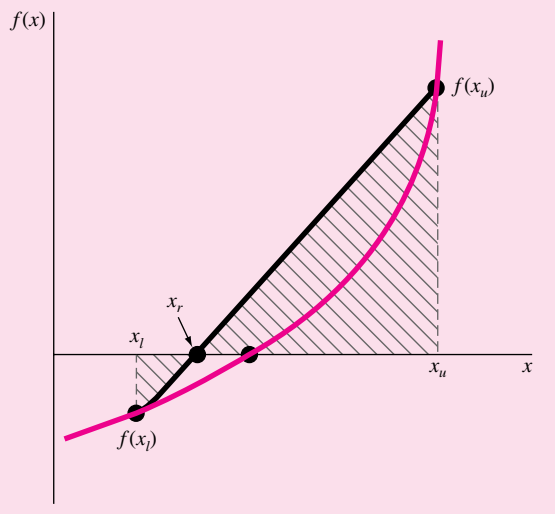
## 5.5 FALSE POSITION

*False position* (also called the linear interpolation method) is another well-known bracketing method. It is very similar to bisection with the exception that it uses a different strategy to come up with its new root estimate. Rather than bisecting the interval, it locates the root by joining $f(x_l)$ and $f(x_u)$ with a straight line (Fig. 5.8). The intersection of this line with the $x$ axis represents an improved estimate of the root. Thus, the shape of the function influences the new root estimate. Using similar triangles, the intersection of the straight line with the $x$ axis can be estimated as (see Chapra and Canale, 2002, for details),

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

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. 5.7) with the exception that Eq. (5.7) is used.

**FIGURE 5.8**
False position.



EXAMPLE 5.5 The False-Position Method

**Problem Statement.** Use false position to solve the same problem approached graphically and with bisection in Examples 5.1 and 5.3.

**Solution.** As in Example 5.3, initiate the computation with guesses of $x_l = 50$ and $x_u = 200$.

First iteration:

$$x_l = 50 \qquad f(x_l) = -4.579387$$
$$x_u = 200 \qquad f(x_u) = 0.860291$$
$$x_r = 200 - \frac{0.860291(50 - 200)}{-4.579387 - 0.860291} = 176.2773$$

which has a true relative error of 23.5%.

Second iteration:

$$f(x_l) f(x_r) = -2.592732$$

Therefore, the root lies in the first subinterval, and $x_r$ becomes the upper limit for the next iteration, $x_u = 176.2773$.

$$x_l = 50 \qquad f(x_l) = -4.579387$$
$$x_u = 176.2773 \qquad f(x_u) = 0.566174$$
$$x_r = 176.2773 - \frac{0.566174(50 - 176.2773)}{-4.579387 - 0.566174} = 162.3828$$

which has true and approximate relative errors of 13.76% and 8.56%, respectively. Additional iterations can be performed to refine the estimates of the root.

Although false position often performs better than bisection, there are other cases where it does not. 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

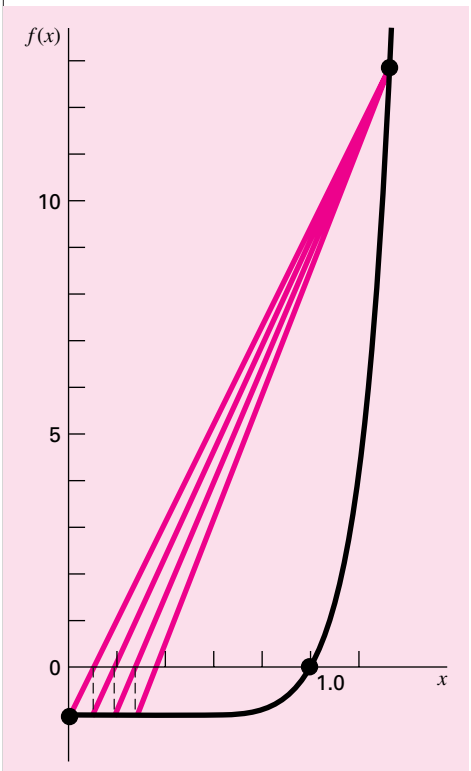
| Iteration | $x_l$ | $x_u$ | $x_r$ | $\varepsilon_a$ (%) | $\varepsilon_t$ (%) |
|---|---|---|---|---|---|
| 1 | 0 | 1.3 | 0.65 | 100.0 | 35 |
| 2 | 0.65 | 1.3 | 0.975 | 33.3 | 2.5 |
| 3 | 0.975 | 1.3 | 1.1375 | 14.3 | 13.8 |
| 4 | 0.975 | 1.1375 | 1.05625 | 7.7 | 5.6 |
| 5 | 0.975 | 1.05625 | 1.015625 | 4.0 | 1.6 |

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

| Iteration | $x_l$ | $x_u$ | $x_r$ | $\varepsilon_a$ (%) | $\varepsilon_t$ (%) |
|---|---|---|---|---|---|
| 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 |

After five iterations, the true error has only been reduced to about 59%. Insight into these results can be gained by examining a plot of the function. As in Fig. 5.9, the curve violates the premise on 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. 5.8). Because of the shape of the present function, the opposite is true.

**FIGURE 5.9**
Plot of $f(x) = x^{10} - 1$, illustrating slow convergence of the false-position method.



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. (5.5), 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. Possible remedies for this shortcoming are available elsewhere (Chapra and Canale, 2002).

## PROBLEMS

**5.1** Use bisection to determine the drag coefficient needed so that an 80-kg bungee jumper has a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is 9.81 m/s$^2$. Start with initial guesses of $x_l = 0.1$ and $x_u = 0.2$ and iterate until the approximate relative error falls below 2%.

**5.2** Develop your own M-file for bisection in a similar fashion to Fig. 5.7. However, rather than using the maximum iterations and Eq. (5.5), employ Eq. (5.6) as your stopping criterion. Make sure to round the result of Eq. (5.6) up to the next highest integer. Test your function by solving Prob. 5.1 using $E_{a,d} = 0.0001$.

**5.3** Repeat Prob. 5.1, but use the false-position method to obtain your solution.

**5.4** Develop an M-file for the false-position method. Test it by solving Prob. 5.1.

**5.5** A beam is loaded as shown in Fig. P5.5. Use the bisection method to solve for the position inside the beam where there is no moment.

**5.6** **(a)** Determine the roots of $f(x) = -12 - 21x + 18x^2 - 2.75x^3$ graphically. In addition, determine the first root of the function with **(b)** bisection and **(c)** false position. For **(b)** and **(c)** use initial guesses of $x_l = -1$ and $x_u = 0$ and a stopping criterion of 1%.

**5.7** Locate the first nontrivial root of $\sin(x) = x^2$ where $x$ is in radians. Use a graphical technique and bisection with the initial interval from 0.5 to 1. Perform the computation until $\varepsilon_a$ is less than $\varepsilon_s = 2\%$.

**5.8** Determine the positive real root of $\ln(x^2) = 0.7$ **(a)** graphically, **(b)** using three iterations of the bisection method, with initial guesses of $x_l = 0.5$ and $x_u = 2$, and **(c)** using three iterations of the false-position method, with the same initial guesses as in **(b)**.

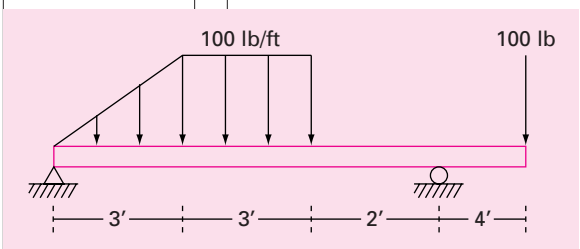
**5.9** The saturation concentration of dissolved oxygen in freshwater can be calculated with the equation

$$\ln o_{sf} = -139.34411 + \frac{1.575701 \times 10^5}{T_a} - \frac{6.642308 \times 10^7}{T_a^2} + \frac{1.243800 \times 10^{10}}{T_a^3} - \frac{8.621949 \times 10^{11}}{T_a^4}$$

where $o_{sf}$ = the saturation concentration of dissolved oxygen in freshwater at 1 atm (mg L$^{-1}$); and $T_a$ = absolute temperature (K). Remember that $T_a = T + 273.15$, where $T$ = temperature (°C). According to this equation, saturation decreases with increasing temperature. For typical natural waters in temperate climates, the equation can be used to determine that oxygen concentration ranges from 14.621 mg/L at 0 °C to 6.949 mg/L at 35 °C. Given a value of oxygen concentration, this formula and the bisection method can be used to solve for temperature in °C.

**(a)** If the initial guesses are set as 0 and 35 °C, how many bisection iterations would be required to determine temperature to an absolute error of 0.05 °C?

**FIGURE P5.5**



**(b)** Based on **(a)**, develop and test a bisection M-file function to determine $T$ as a function of a given oxygen concentration. Test your function for $o_{sf} = 8$, 10 and 14 mg/L. Check your results.

**5.10** Water is flowing in a trapezoidal channel at a rate of $Q = 20$ m$^3$/s. The critical depth $y$ for such a channel must satisfy the equation

$$0 = 1 - \frac{Q^2}{gA_c^3}B$$

where $g = 9.81$ m/s$^2$, $A_c$ = the cross-sectional area (m$^2$), and $B$ = the width of the channel at the surface (m). For this case, the width and the cross-sectional area can be related to depth $y$ by

$$B = 3 + y$$

and

$$A_c = 3y + \frac{y^2}{2}$$

Solve for the critical depth using **(a)** the graphical method, **(b)** bisection, and **(c)** false position. For **(b)** and **(c)** use initial guesses of $x_l = 0.5$ and $x_u = 2.5$, and iterate until the approximate error falls below 1% or the number of iterations exceeds 10. Discuss your results.