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, 2010, 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

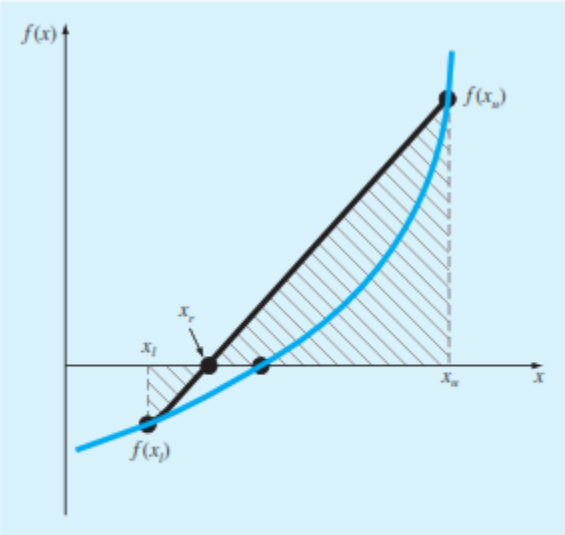


FIGURE 5.8
False position.

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.

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.

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

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	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

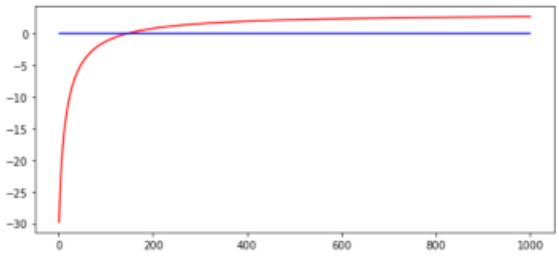
Thus, after five iterations, the true error is reduced to less than 2%. For false position, a very different outcome is obtained:

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

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

In [44]:

```
import numpy as np
from matplotlib import pyplot as plt

plt.rcParams["figure.figsize"] = [7.50, 3.50]
plt.rcParams["figure.autolayout"] = True

def f(x):

   return np.sqrt(((9.81*x)/0.25))*np.tanh(np.sqrt((9.81*0.25)/x)*4)-36
def f1(x):
    return x*0


x = np.linspace(1, 1000, 1000)

plt.plot(x, f(x), color='red')
plt.plot(x, f1(x), color='blue')

plt.show()
```



In [43]:

```
import numpy as np
from matplotlib import pyplot as plt

plt.rcParams["figure.figsize"] = [7.50, 7.50]
plt.rcParams["figure.autolayout"] = False

def f(x):
    return ((x**10)-1)

def f1(x):
    return x*0

x = np.linspace(-1.5, 1.5, 1000)

plt.plot(x,f(x), color='red')
plt.plot(x,f1(x), color='blue')

plt.show()
```