

# Chapter 7

## The Solution of Nonlinear Equations

**I**n Chapter 3 we identified the solution of a system of linear equations as one of the fundamental problems in numerical analysis. It is also one of the easiest, in the sense that there are efficient algorithms for it, and that it is relatively straightforward to evaluate the results produced by them. Nonlinear equations, however, are more difficult, even when the number of unknowns is small.

### Example 7.1

The following is an abstracted and greatly simplified version of a missile-intercept problem.

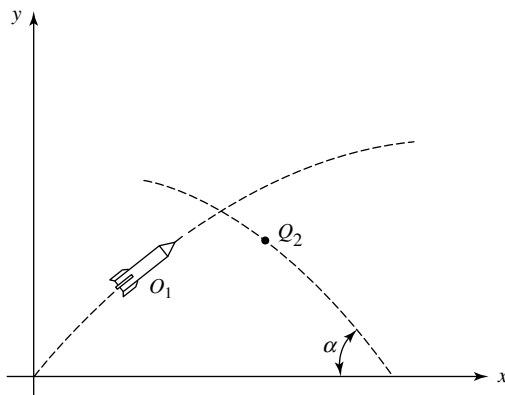
The movement of an object  $O_1$  in the  $xy$  plane is described by the parameterized equations

$$\begin{aligned}x_1(t) &= t, \\ y_1(t) &= 1 - e^{-t}.\end{aligned}\tag{7.1}$$

A second object  $O_2$  moves according to the equations

$$\begin{aligned}x_2(t) &= 1 - \cos(\alpha)t, \\ y_2(t) &= \sin(\alpha)t - 0.1t^2.\end{aligned}\tag{7.2}$$

**Figure 7.1**  
The missile-  
intercept problem.



Is it possible to choose a value for  $\alpha$  so that both objects will be in the same place at some  $t$ ? (See Figure 7.1.)

When we set the  $x$  and  $y$  coordinates equal, we get the system

$$\begin{aligned} t &= 1 - \cos(\alpha)t, \\ 1 - e^{-t} &= \sin(\alpha)t - 0.1t^2, \end{aligned} \tag{7.3}$$

that needs to be solved for the unknowns  $\alpha$  and  $t$ . If real values exist for these unknowns that satisfy the two equations, both objects will be in the same place at some value of  $t$ . But even though the problem is a rather simple one that yields a small system, there is no obvious way to get the answers, or even to see if there is a solution. ■

The numerical solution of a system of nonlinear equations is one of the more challenging tasks in numerical analysis and, as we will see, no completely satisfactory method exists for it. To understand the difficulties, we start with what at first seems to be a rather easy problem, the solution of a single equation in one variable

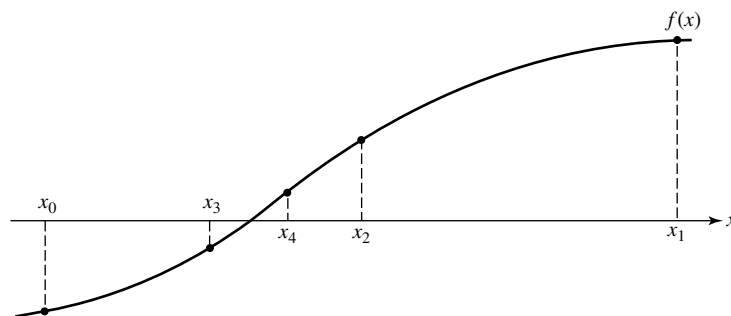
$$f(x) = 0. \tag{7.4}$$

The values of  $x$  that satisfy this equation are called the *zeros* or *roots* of the function  $f$ . In what is to follow we will assume that  $f$  is continuous and sufficiently differentiable where needed.

## 7.1 Some Simple Root-Finding Methods

To find the roots of a function of one variable is straightforward enough—just plot the function and see where it crosses the  $x$ -axis. The simplest

**Figure 7.2**  
The bisection method. After three steps the root is known to lie in the interval  $[x_3, x_4]$ .



methods are in fact little more than that and only carry out this suggestion in a systematic and efficient way. Relying on the intuitive insight of the graph of the function  $f$ , we can discover many different and apparently viable methods for finding the roots of a function of one variable.

Suppose we have two values,  $x_0$  and  $x_1$ , such that  $f(x_0)$  and  $f(x_1)$  have opposite signs. Then, because it is assumed that  $f$  is continuous, we know that there is a root somewhere in the interval  $[x_0, x_1]$ . To localize it, we take the midpoint  $x_2$  of this interval and compute  $f(x_2)$ . Depending on the sign of  $f(x_2)$ , we can then place the root in one of the two intervals  $[x_0, x_2]$  or  $[x_2, x_1]$ . We can repeat this procedure until the region in which the root is known to be located is sufficiently small (Figure 7.2). The algorithm is known as the *bisection method*.

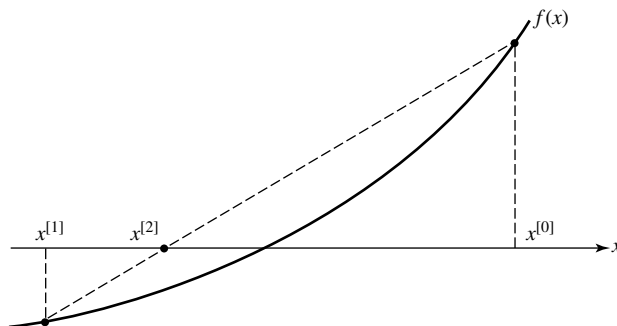
The bisection method is very simple and intuitive, but has all the major characteristics of other root-finding methods. We start with an initial guess for the root and carry out some computations. Based on these computations we then choose a new and, we hope, better approximation of the solution. The term *iteration* is used for this repetition. In general, an iteration produces a sequence of approximate solutions; we will denote these iterates by  $x^{[0]}, x^{[1]}, x^{[2]}, \dots$ . The difference between the various root-finding methods lies in what is computed at each step and how the next iterate is chosen.

Suppose we have two iterates  $x^{[0]}$  and  $x^{[1]}$  that enclose the root. We can then approximate  $f(x)$  by a straight line in the interval and find the place where this line cuts the  $x$ -axis (Figure 7.3). We take this as the new iterate

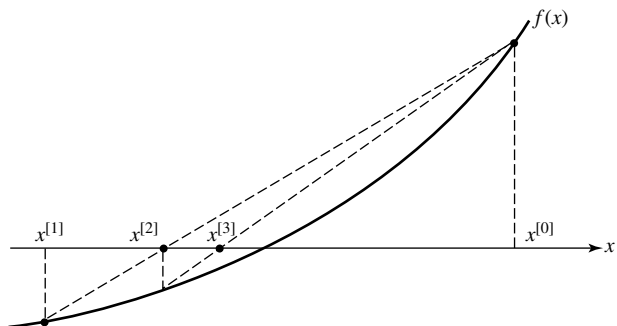
$$x^{[2]} = x^{[1]} - \frac{(x^{[1]} - x^{[0]})f(x^{[1]})}{f(x^{[1]}) - f(x^{[0]})}. \quad (7.5)$$

When this process is repeated, we have to decide which of the three points  $x^{[0]}$ ,  $x^{[1]}$ , or  $x^{[2]}$ , to select for starting the next iteration. There are two plausible choices. In the first, we retain the last iterate and one point from the previous ones so that the two new points enclose the solution (Figure 7.4). This is the *method of false position*.

**Figure 7.3**  
Approximating a root by linear interpolation.



**Figure 7.4**  
The method of false position. After the second iteration, the root is known to lie in the interval  $(x^{[3]}, x^{[0]})$ .



The second choice is to retain the last two iterates, regardless of whether or not they enclose the solution. The successive iterates are then simply computed by

$$x^{[i+1]} = x^{[i]} - \frac{(x^{[i]} - x^{[i-1]})f(x^{[i]})}{f(x^{[i]}) - f(x^{[i-1]})}. \quad (7.6)$$

This is the *secant method*. Figure 7.5 illustrates how the secant method works and shows the difference between it and the method of false position. From this example we can see that now the successive iterates are no longer guaranteed to enclose the root.

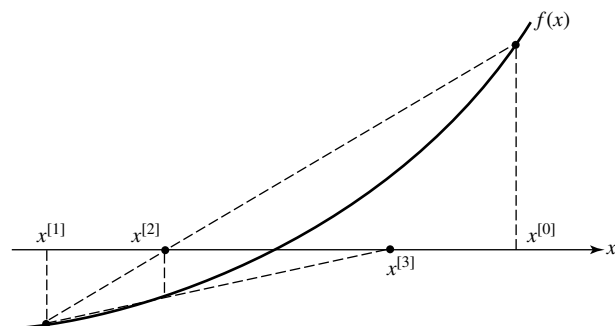
**Example 7.2**

The function

$$f(x) = x^2 e^x - 1$$

has a root in the interval  $[0, 1]$  since  $f(0)f(1) < 0$ . The results from the false position and secant methods, both started with  $x^{[0]} = 0$  and  $x^{[1]} = 1$ ,

**Figure 7.5**  
The secant method.



**Table 7.1**  
Comparison of the  
false position and  
secant methods.

Iterates	False position	Secant
$x^{[2]}$	0.3679	0.3679
$x^{[3]}$	0.5695	0.5695
$x^{[4]}$	0.6551	0.7974
$x^{[5]}$	0.6868	0.6855
$x^{[6]}$	0.6978	0.7012
$x^{[7]}$	0.7016	0.7035

are shown in Table 7.1. It appears from these results that the secant method gives the correct result  $x = 0.7035$  a little more quickly.

A popular iteration method can be motivated by Taylor’s theorem. Suppose that  $x^*$  is a root of  $f$ . Then for any  $x$  near  $x^*$ ,

$$\begin{aligned} f(x^*) &= f(x) + (x^* - x)f'(x) + \frac{(x^* - x)^2}{2}f''(x) + \dots \\ &= 0. \end{aligned}$$

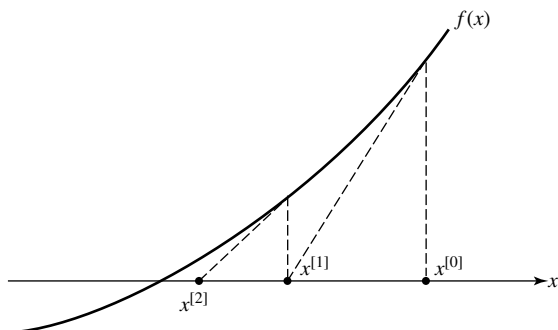
Neglecting the second order term on the right, we get

$$x^* \cong x - \frac{f(x)}{f'(x)}.$$

While this expression does not give the exact value of the root, it promises to give a good approximation to it. This suggests the iteration

$$x^{[i+1]} = x^{[i]} - \frac{f(x^{[i]})}{f'(x^{[i]})}, \tag{7.7}$$

**Figure 7.6**  
Two iterations of  
Newton’s method.



**Table 7.2**  
Results for  
Newton’s method.

Iterates	Newton
$x^{[0]}$	0.3679
$x^{[1]}$	1.0071
$x^{[2]}$	0.7928
$x^{[3]}$	0.7133
$x^{[4]}$	0.7036
$x^{[6]}$	0.7035

starting with some initial guess  $x^{[0]}$  (Figure 7.6). The process is called *Newton’s method*.

**Example 7.3**

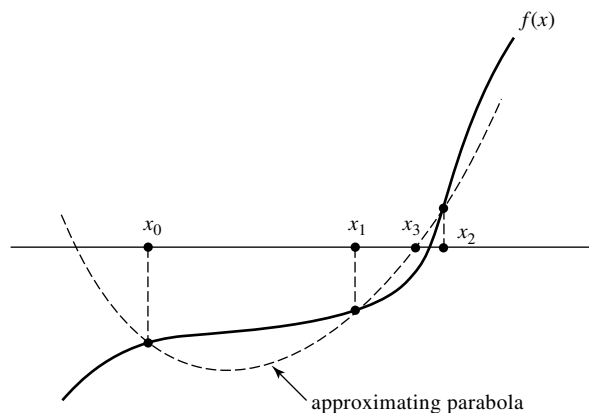
The equation in Example 7.2 was solved by Newton’s method, with the starting guess  $x^{[0]} = 0.3679$ . Successive iterates are shown in Table 7.2. The results, though somewhat erratic in the beginning, give the root with four-digit accuracy quickly.

The false position and secant methods are both based on linear interpolation on two iterates. If we have three iterates, we can think of approximating the function by interpolating with a second degree polynomial and solving the approximating quadratic equation to get an approximation to the root (Figure 7.7).

If we have three points  $x_0, x_1, x_2$  with the corresponding function values  $f(x_0), f(x_1), f(x_2)$ , we can use divided differences to find the second degree interpolating polynomial. From (4.12), with an interchange of  $x_0$  and  $x_2$ , this is

$$p_2(x) = f(x_2) + (x - x_2)f[x_2, x_1] + (x - x_2)(x - x_1)f[x_2, x_1, x_0].$$

**Figure 7.7**  
Approximating a root with quadratic interpolation.



If we write this in the form

$$p_2(x) = a(x - x_2)^2 + b(x - x_2) + c,$$

then

$$\begin{aligned} a &= f[x_2, x_1, x_0], \\ b &= f[x_2, x_1] + (x_2 - x_1)f[x_2, x_1, x_0], \\ c &= f(x_2). \end{aligned}$$

The equation

$$p_2(x) = 0$$

has two solutions

$$x = x_2 + \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Of these two we normally take the closest to  $x_2$  (Figure 7.7), which we can write as

$$x_3 = x_2 + \frac{-b + \text{sign}(b)\sqrt{b^2 - 4ac}}{2a}.$$

Since the numerator may involve cancellation of two nearly equal terms, we prefer the more stable expression

$$x_3 = x_2 - \frac{2c}{b + \text{sign}(b)\sqrt{b^2 - 4ac}}. \quad (7.8)$$

As with the linear interpolation method, we have a choice in what points to retain when we go from one iterate to the next. If the initial three points

bracket the solution, we can arrange it so that the next three points do so as well, and we get an iteration that always encloses the root. This has obvious advantages, but as with the method of false position, it can affect the speed with which the accuracy of the root improves. The alternative is to retain always the latest three iterates, an option known as *Muller’s method*.

Obviously one can use even more points and higher degree interpolating polynomials. But this is of little use because closed form solutions for the roots of polynomials of degree higher than two are either very cumbersome or not known. Furthermore, as we will discuss in the next section, Muller’s method is only slightly better than the secant method so little further improvement can be expected.

## EXERCISES

1. How many iterations of bisection are needed in Example 7.2 to get 4-digit accuracy?
2. Use the bisection method to find a root of  $f(x) = 1 - 2e^x$  to two significant digits.
3. Use Newton’s method to find the root in Exercise 2 to six significant digits.
4. Give a graphical explanation for the irregularity of the early iterates in the secant and Newton’s methods observed in Examples 7.2 and 7.3.
5. Consider the following suggestion: We can modify the bisection method to get a *trisection* method by computing the value of  $f$  at the one-third and two-thirds points of the interval, then taking the smallest interval over which there is a sign change. This will reduce the interval in which the root is known to be located by a factor three in each step and so give the root more quickly. Is there any merit to this suggestion?
6. Suppose that a computer program claims that the root it produced has an accuracy of  $10^{-6}$ . How do you verify this claim?
7. Use Muller’s method to get a rough location of the root of a function  $f$  whose values are tabulated as follows.

$x$	$f(x)$
0	1.20
0.5	0.65
1.0	-0.50

8. Find the three smallest positive roots of

$$x - \cot(x) = 0$$

to an accuracy of  $10^{-4}$ .

9. In the system (7.2), use the first equation to solve for  $\cos(\alpha)$  in terms of  $t$ . Substitute this into the second equation to get a single equation in the unknown  $t$ . Use the secant method to solve for  $t$  and from this get a value for  $\alpha$ .

## 7.2 Convergence Rates of Root-Finding Methods

To put these preliminary results into perspective, we need to develop a way of comparing the different methods by how well they work and how quickly they will give a desired level of accuracy.

### Definition 7.1

Let  $x^{[0]}, x^{[1]}, \dots$  be a sequence of iterates produced by some root-finding method for the equation  $f(x) = 0$ . Then we say that the method produces *convergent* iterates (or just simply that it is convergent) if there exists an  $x^*$  such that

$$\lim_{i \rightarrow \infty} x^{[i]} = x^*, \quad (7.9)$$

with  $f(x^*) = 0$ . If a method is convergent and there exists a constant  $c$  such that

$$|x^{[i+1]} - x^*| \leq c|x^{[i]} - x^*|^k \quad (7.10)$$

for all  $i$ , then the method is said to have *iterative order of convergence*  $k$ .

For a first-order method, convergence can be guaranteed only if  $c < 1$ ; for methods of higher order the error in the iterates will decrease as long as the starting guess  $x^{[0]}$  is sufficiently close to the root.

An analysis of the bisection method is elementary. At each step, the interval in which the root and the iterate are located is halved, so we know that the method converges and the error is reduced by about one-half at each step. The bisection method is therefore a first-order method. To reduce the interval to a size  $\varepsilon$ , and thus guarantee the root to this accuracy, we must repeat the bisection process  $k$  times such that

$$2^{-k}|x^{[0]} - x^{[1]}| \leq \varepsilon,$$

or

$$k \geq \log_2 \frac{|x^{[0]} - x^{[1]}|}{\varepsilon}.$$

If the original interval is of order unity, it will take about 50 bisections to reduce the error to  $10^{-15}$ .

To see how Newton’s method works, let us examine the error in successive iterations,

$$\varepsilon_i = x^* - x^{[i]}.$$

From (7.6), we get that

$$x^* - x^{[i+1]} = x^* - x^{[i]} + \frac{f(x^{[i]}) - f(x^*)}{f'(x^{[i]})},$$

and expanding the last term on the right by Taylor’s theorem,

$$x^* - x^{[i+1]} = x^* - x^{[i]} + \frac{(x^{[i]} - x^*)f'(x^{[i]})}{f'(x^{[i]})} + \frac{(x^{[i]} - x^*)^2 f''(x^{[i]})}{2f'(x^{[i]})} + \dots$$

After canceling terms, this gives that, approximately,

$$\varepsilon_{i+1} \cong c\varepsilon_i^2, \tag{7.11}$$

where  $c = f''(x^{[i]})/2f'(x^{[i]})$ . If  $c$  is of order unity, then the error is roughly squared on each iteration; to reduce it from 0.5 to  $10^{-15}$  takes about 6 or 7 iterations. This is potentially much faster than the bisection method.

This discussion suggests that Newton’s method has second-order convergence, but the informality of the arguments does not quite prove this. To produce a rigorous proof is a little involved and is of no importance here. All we need to remember is the somewhat vague, but nevertheless informative statement that Newton’s method has iterative order of convergence two, provided the starting value is sufficiently close to a zero. Arguments can also be made to show that the secant method has an order of convergence of about 1.62 and Muller’s method an order approximately 1.84. While this makes Muller’s method faster in principle, the improvement is not very great. This often makes the simplicity of the secant method preferable. The method of false position, on the other hand, has order of convergence one and can be quite slow.

The arguments for establishing the convergence rates for the secant method and for Muller’s method are quite technical and we need not pursue them here. A simple example will demonstrate the rate quite nicely. If a method has order of convergence  $k$ , then

$$\varepsilon_{i+1} \cong c\varepsilon_i^k.$$

A little bit of algebra shows that

$$k \cong \frac{\log \varepsilon_{i+1} - \log \varepsilon_i}{\log \varepsilon_i - \log \varepsilon_{i-1}}. \tag{7.12}$$

**Table 7.3**  
Estimation of the order of convergence for Newton’s method using (7.12).

Iterates	Errors	$k$
$x^{[0]}$	$1.315 \times 10^0$	--
$x^{[1]}$	$8.282 \times 10^{-1}$	--
$x^{[2]}$	$3.836 \times 10^{-1}$	1.66
$x^{[3]}$	$7.532 \times 10^{-2}$	2.12
$x^{[4]}$	$1.140 \times 10^{-3}$	2.57
$x^{[5]}$	$1.001 \times 10^{-7}$	2.23
$x^{[6]}$	$7.772 \times 10^{-16}$	2.00

For examples with known roots, the quantity on the right can be computed to estimate the convergence rate.

**Example 7.4** The function

$$f(x) = e^{x^2} - \frac{5}{e^{2x}}$$

has a known positive root  $x^* = \sqrt{1 + \log_e(5)} - 1$ . Using Newton’s method, with the starting guess  $x^{[0]} = -0.7$ , errors in successive iterates and the estimates for the order of convergence  $k$  are shown in Table 7.3. ■

It is possible to construct methods that have iterative order larger than two, but the derivations get quite complicated. In any case, second-order methods converge so quickly that higher order methods are rarely needed.

## EXERCISES

- Suppose that  $f$  has a root  $x^*$  in some interval  $[a, b]$  and that  $f'(x) > 0$  and  $f''(x) > 0$  in this interval. If  $x^{[0]} > x^*$ , show that convergence of Newton’s method is monotone toward the root; that is

$$x^{[0]} > x^{[1]} > x^{[2]} > \dots$$

- Describe what happens with the method of false position under the conditions of Exercise 1.
- Give a rough estimate of how many iterations of the secant method will be needed to reduce the error from 0.5 to about  $10^{-15}$ .
- Give an estimate of how many iterations with Muller’s method are needed under the conditions of the previous exercise.

5. Roughly how many iterations would you expect a third-order method to take to reduce the error from 0.5 to  $10^{-15}$ ? Compare this with the number of iterations from Newton’s method.
6. Draw a graph to argue that the false position method will not work very well for finding the root of

$$f(x) = e^{-20|x|} - 10^{-4}.$$

7. A common use of Newton’s method is in algorithms for computing the square root of a positive number  $a$ . Applying the method to the equation

$$x^2 - a = 0,$$

show that the iterates are given by

$$x^{[i+1]} = \frac{1}{2} \left( x^{[i]} + \frac{a}{x^{[i]}} \right).$$

Prove that, for any positive  $x^{[0]}$ , this sequence converges to  $\sqrt{a}$ .

8. Use Newton’s method to design an algorithm to compute  $\sqrt[3]{a}$  for positive  $a$ . Verify the second-order convergence of the algorithm.
9. Use the equation in Example 7.4 to investigate the convergence rate for Muller’s method.

### 7.3 Difficulties with Root-Finding

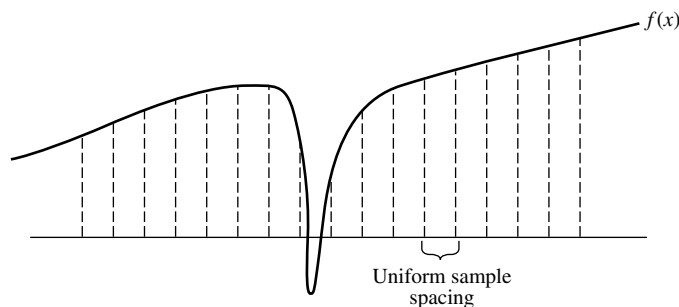
Most root-finding methods for a single equation are intuitive and easy to apply, but they are all based on the assumption that the function whose roots are to be found is well behaved in some sense. When this is not the case, the computations can become complicated. For example, both the bisection method and the method of false position require that we start with two guesses that bracket a root; that is, that

$$f(x^{[0]})f(x^{[1]}) \leq 0,$$

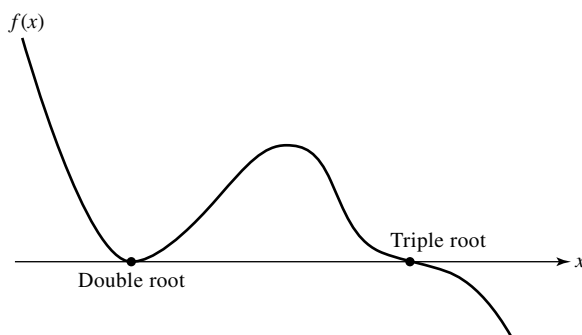
while for Newton’s method we need a good initial guess for the root. We can try to find suitable starting points by a sampling of the function at various points, but there is a price to pay. If we sample at widely spaced points, we may miss a root; if our spacing is very close we will have to expend a great deal of work. But even with very close sampling we may miss some roots. A situation like that shown in Figure 7.8 defeats many algorithms.

This situation illustrates how difficult it is to construct an automatic root-finder. Even the most sophisticated algorithm will occasionally miss if it is used without an analysis that tells something about the structure of the function. This situation is not unique to root-finding, but applies to many other numerical algorithms. Whatever method we choose, and no

**Figure 7.8**  
A case when two roots can be missed entirely by sampling.



**Figure 7.9**  
Two roots of higher multiplicity.



matter how carefully we implement it, there will always be some problems for which the method fails.

Newton’s method, in addition to requiring a good initial approximation, also requires that  $f'(x^{[i]})$  does not vanish. This creates a problem for the situation shown in Figure 7.9, where  $f'(x^*) = 0$ . Such roots require special attention.

**Definition 7.2**

A root  $x^*$  of  $f$  is said to have a multiplicity  $p$  if

$$f(x^*) = f'(x^*) = \dots = f^{(p-1)}(x^*) = 0, \\ f^{(p)}(x^*) \neq 0.$$

A root of multiplicity one is called a *simple* root, a root of multiplicity two is a *double* root, and so on.

It can be shown that Newton’s method still converges to roots of higher multiplicity, but the order of convergence is reduced. Methods like bisection

















































