Chapter 7

The Solution of Nonlinear Equations

In 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

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

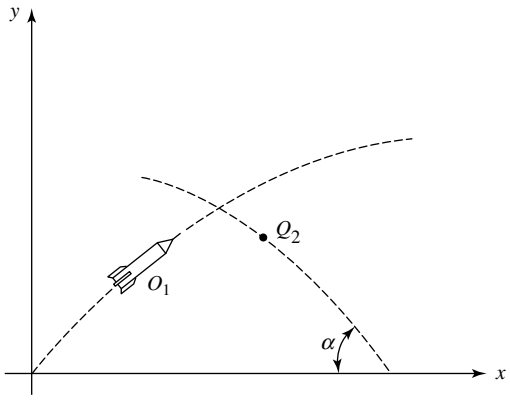

Figure 7.1
The missile-intercept problem.

Is it possible to choose a value for α 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 α 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

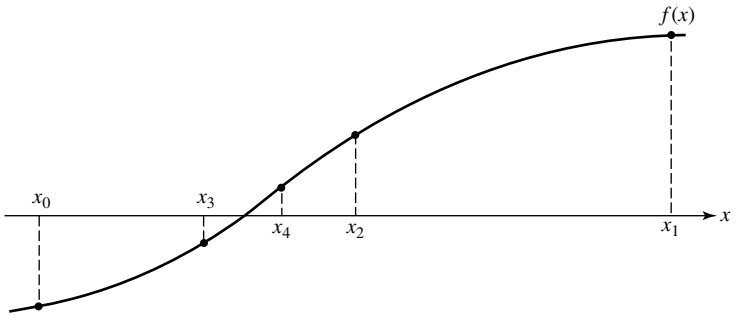

Figure 7.2
The bisection method. After three steps the root is known to lie in the interval $[x_3,\ x_4]$.

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 $x^{[0]},\ x^{[1]},\ x^{[2]},\ \ldots$. 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

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

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*.

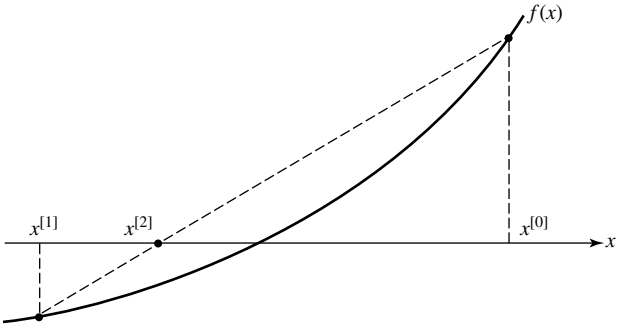

Figure 7.3
Approximating a root by linear interpolation.

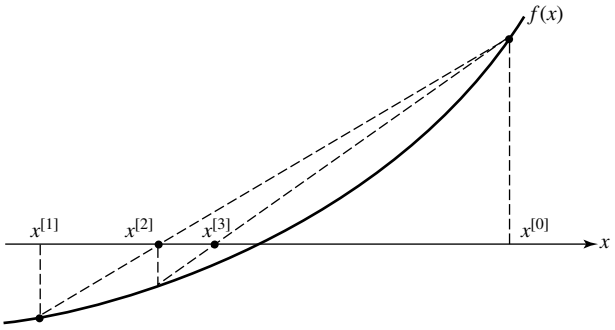

Figure 7.4
The method of false position. After the second iteration, the root is known to lie in the interval $(x^{[3]},\ x^{[0]})$.

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

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

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 $x^{[1]} = 1$,

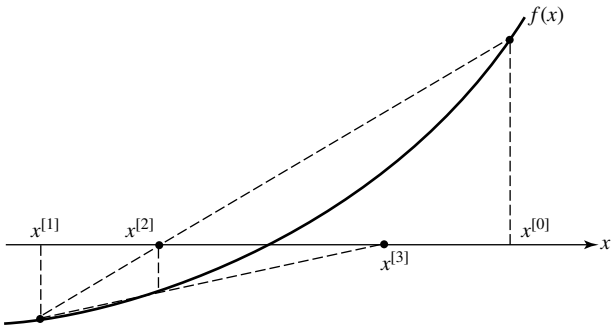

Figure 7.5
The secant method.

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^*,

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

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

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

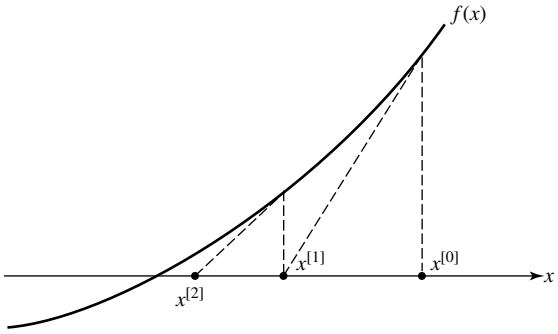

Figure 7.6
Two iterations of Newton's method.

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

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

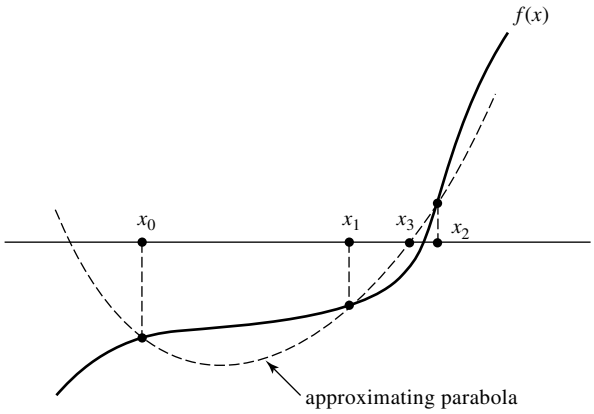

Figure 7.7
Approximating a root with quadratic interpolation.

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

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

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

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

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 α.

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 $x^{[0]}$, $x^{[1]}$, ... 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

$$\lim_{i \to \infty} x^{[i]} = x^*, \tag{7.9}$$

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

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

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 ε, 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,

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

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
$x^{[0]}$	1.315×10^{0}	--
$x^{[1]}$	8.282×10^{-1}	--
$x^{[2]}$	3.836×10^{-1}	1.66
$x^{[3]}$	7.532×10^{-2}	2.12
$x^{[4]}$	1.140×10^{-3}	2.57
$x^{[5]}$	1.001×10^{-7}	2.23
$x^{[6]}$	7.772×10^{-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

1. 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
$$x^{[0]} > x^{[1]} > x^{[2]} > \ \ldots\ .$$
2. Describe what happens with the method of false position under the conditions of Exercise 1.
3. 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}.
4. 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

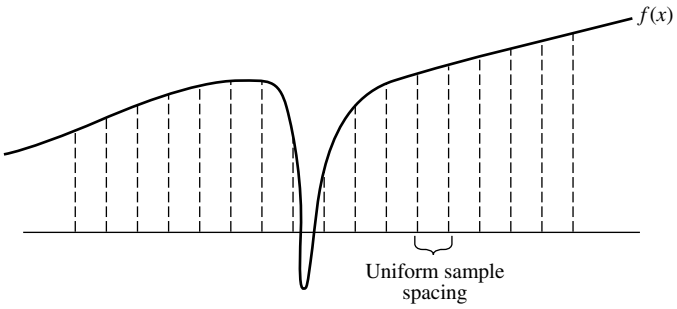

Figure 7.8
A case when two roots can be missed entirely by sampling.

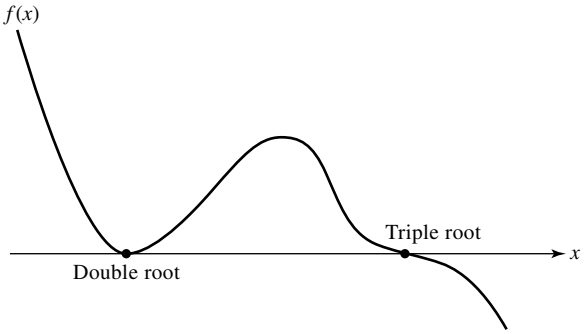

Figure 7.9
Two roots of higher multiplicity.

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

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

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

are not directly affected, but cannot get started for roots of even multiplicity. In either case, there is a problem of the accuracy with which a root can be obtained. To see this, consider what happens when we try to find a root on the computer. Because x^* is not necessarily a computer-representable number and the function f usually cannot be computed exactly, we can find only an approximation to the root. Even at the closest approximation to the root the value of f will not be zero. All we can say is that at the root, the value of f should be zero within the computational accuracy. If ε represents all the computational error, we can only agree that as long as

$$|f(\hat{x})| \leq \varepsilon \tag{7.13}$$

then $\hat{x}$ is an acceptable approximation to the root. Usually, there is a region around x^* where (7.13) is satisfied and the width of this region defines a limit on the accuracy with which one can approximate the root. From Taylor's theorem

$$f(\hat{x}) = f(x^*) + (\hat{x} - x^*)f'(x^*) + \frac{(\hat{x} - x^*)^2}{2}f''(x^*) + \ \ldots \ .$$

If x^* is a simple root, then it follows that, sufficiently close to the root,

$$|\hat{x} - x^*| \leq \frac{\varepsilon}{|f'(x^*)|},$$

so that the accuracy with which a simple root can be located is $O(\varepsilon)$. This makes finding a simple root a well-conditioned problem, unless $f'(x^*)$ is very close to zero.

When x^* is a zero of multiplicity two, then $f'(x^*) = 0$ and

$$|\hat{x} - x^*| \leq \sqrt{\frac{2\varepsilon}{|f''(x^*)|}},$$

so we can guarantee only an $O(\sqrt{\varepsilon})$ accuracy. In a similar way, we can show that if a root is of multiplicity p, then we can get its location only to an accuracy $O\left(\varepsilon^{1/p}\right)$. Finding roots of high multiplicity is an ill-conditioned problem.

EXERCISES

1. Show that the accuracy with which a root of multiplicity p can be found is $O(\varepsilon^{1/p})$.
2. Show that if f has a simple root at x^* then f^2 has a double root at that point.
3. Draw a graph that illustrates why Newton's method converges slowly to a double root.

4. Repeat the analysis leading to (7.11) to lend substance to the conjecture that Newton's method converges linearly to a double root.
5. Do you think that Newton's method tends to converge slower near a triple root than near a double root?
6. How do you think the false position method will behave near a triple root?
7. If we know that f has a root x^* of multiplicity p, then the following modification of Newton's method (7.6) will still have order two convergence:

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

Experimentally examine the claim that this modification of Newton's method has order two convergence by applying the function

$$f(x) = (x-1)^3 \sin(x)$$

with initial iterate $x^{[0]} = 2$.

7.4 The Solution of Simultaneous Equations by Newton's Method

Not all methods for a single equation carry over to systems of equations. For example, it is hard to see how one can adapt the bisection method to more than one equation. The one-dimensional method that most easily extends to several dimensions is Newton's method.

We consider the solution of a system of nonlinear equations

$$\begin{aligned} f_1(x_1,\ x_2,\ \ldots,\ x_n) &= 0, \\ f_2(x_1,\ x_2,\ \ldots,\ x_n) &= 0, \\ &\vdots \\ f_n(x_1,\ x_2,\ \ldots,\ x_n) &= 0, \end{aligned} \tag{7.14}$$

which we write in vector notation as

$$\mathbf{F}(\mathbf{x}) = 0. \tag{7.15}$$

We will assume that the number of unknowns and the number of equations are the same.

Using a multi-dimensional Taylor expansion, one can show that in n dimensions (7.7) becomes

$$\mathbf{x}^{[i+1]} = \mathbf{x}^{[i]} - \mathbf{J}^{-1}(\mathbf{x}^{[i]})\mathbf{F}(\mathbf{x}^{[i]}), \tag{7.16}$$

where $\mathbf{J}$ is the *Jacobian*

$$\mathbf{J}(\mathbf{z}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\mathbf{x}=\mathbf{z}}.$$

For practical computations, we do not usually use (7.16) but prefer the form

$$\mathbf{x}^{[i+1]} = \mathbf{x}^{[i]} + \Delta^{[i]}, \tag{7.17}$$

where $\Delta^{[i]}$ is the solution of

$$\mathbf{J}(\mathbf{x}^{[i]})\Delta^{[i]} = -\mathbf{F}(\mathbf{x}^{[i]}). \tag{7.18}$$

Each step in the multi-dimensional Newton's method involves the solution of an n-dimensional linear system. In addition, in each step we have to evaluate the n^2 elements of the Jacobian. It can be shown that, as in the one-dimensional case, the order of convergence of the multi-dimensional Newton's method is two, provided we have a good starting guess.

Example 7.5 Consider the solution of Example 7.1. Setting the x and y coordinates equal to each other, we get a system in two unknowns

$$\begin{aligned} 1 - \cos(\alpha)t - t &= 0, \\ \sin(\alpha)t - 0.1t^2 - 1 + e^{-t} &= 0. \end{aligned}$$

Using (7.16) with initial guess $(t,\ \alpha) = (1,\ 1)$, Newton's method converges to (0.6278603030418165, 0.9363756944918756) in four iterations. This can be shown to agree with the true solution to thirteen significant digits. The 2-norm of the error in successive iterates and the estimates for the order of convergence k are shown in Table 7.4.

But things do not always work as smoothly as this. Even for some small, innocent-looking systems we may get into some difficulties.

Example 7.6 For the system

$$\begin{aligned} x_1^2 + 2\sin(x_2) + x_3 &= 0, \\ \cos(x_2) - x_3 &= 2, \\ x_1^2 + x_2^2 + x_3^2 &= 2, \end{aligned}$$

Table 7.4
Estimate of order of convergence for Newton's method in calculating the solution of Example 7.1 using (7.12).

Iterates	Errors	k
0	3.78×10^{-1}	--
1	4.55×10^{-2}	--
2	6.77×10^{-4}	1.99
3	3.90×10^{-7}	1.77
4	8.98×10^{-14}	2.05

the Jacobian is

$$\mathbf{J} = \begin{bmatrix} 2x_1 & 2\cos(x_2) & 1 \\ 0 & -\sin(x_2) & -1 \\ 2x_1 & 2x_2 & 2x_3 \end{bmatrix}.$$

When started with $\mathbf{x}^{[0]} = (1.1,\ 0.1,\ -0.9)$ the iterates produced by Newton's method were

$$\mathbf{x}^{[1]} = (1.021674086,\ -0.022927241,\ -0.992723588),$$
$$\mathbf{x}^{[2]} = (1.000739746,\ -0.000641096,\ -0.999751903),$$
$$\mathbf{x}^{[3]} = (1.000000714,\ -0.000000544,\ -0.999999795),$$

and the next iterate agrees with the root $\mathbf{x}^* = (1,\ 0,\ -1)$ to more than ten significant digits.

However, with the starting value $\mathbf{x}^{[0]} = (1,\ 1,\ 0)$ we obtained the following iterates

$$\mathbf{x}^{[1]} = (0.3053,\ 1.6947,\ -2.0443),$$
$$\mathbf{x}^{[2]} = (-0.9085,\ 1.0599,\ -1.4936),$$
$$\mathbf{x}^{[3]} = (-0.3404,\ 0.8060,\ -1.2896).$$

No clear pattern has emerged in these first approximations and no convergence was observed after 20 iterations.

This example is typical of Newton's method and illustrates its main features. When the method works, it works exceedingly well. The second order of convergence allows us to get very accurate results with just a few iterations. On the other hand, unless we have a good starting value, the whole process may fail spectacularly and we may do a lot of work without achieving anything.

■

Newton's method has been studied extensively because theoretically it is very attractive. It has second-order convergence and one can establish

theorems that tell us exactly how close the first guess has to be to assure convergence. From a practical point of view, there are some immediate difficulties. First, we have to get the Jacobian which requires explicit expressions for n^2 partial derivatives. If this proves too cumbersome, we can use numerical differentiation but, as suggested in Chapter 6, this will lead to some loss of accuracy. Also, each step requires the solution of a linear system and this can be expensive. There are ways in which some work can be saved, say, by using the same Jacobian for a few steps before recomputing it. This slows the convergence but can improve the overall efficiency. The main difficulty, though, with Newton's method is that it requires a good starting guess. When we do not have a good starting vector, the iterates can behave quite erratically and wander around in n-space for a long time. This problem can be alleviated by monitoring the iterates and restarting the computations when it looks like convergence has failed, but on the whole, there is no entirely satisfactory solution to the starting problem. In some applications, the physical origin of the system might suggest a good initial value. In other cases, we may need to solve a sequence of nonlinear problems that are closely related, so one problem could suggest starting values for the next. In these situations, Newton's method can be very effective. In general, though, locating the solution roughly is much harder than to refine its accuracy. This is the main stumbling block to the effective solution of (7.15) by Newton's method.

EXERCISES

1. Use the Taylor expansion in two dimensions to derive (7.16) for $n = 2$.

2. Why is $\mathbf{x}^{[0]} = (0,\ 0,\ 0)$ not a good starting vector for Example 7.6?

3. Investigate whether Example 7.5 has any other solutions.

4. Find all the solutions of the system

$$x_1 + 2x_2^2 = 1,$$
$$|x_1| - x_2^2 = 0.$$

5. What can you expect from Newton's method near a multiple root?

6. An interesting biological experiment concerns the determination of the maximum water temperature at which various species of hydra can survive without shortened life expectancy. The problem can be solved by finding the weighted least-squares approximation for the model $y = a/(x - b)^c$ with the unknown parameters a, b, and c, using a collection of experimental data. The x refers to water temperature, and y refers to the average life expectancy at that temperature. More precisely, a, b, and c are chosen to minimize

$$\sum_{i=1}^{n} \left[y_i w_i - \frac{a}{(x_i - b)^c} \right]^2.$$

(a) Show that $a,\ b,\ c$ must be a solution of the following nonlinear system:

$$a = \Big(\sum_{i=1}^{n} \frac{y_i w_i}{(x_i - b)^c}\Big) \Big/ \Big(\sum_{i=1}^{n} \frac{1}{(x_i - b)^{2c}}\Big),$$

$$0 = \sum_{i=1}^{n} \frac{y_i w_i}{(x_i - b)^c} \cdot \sum_{i=1}^{n} \frac{1}{(x_i - b)^{2c+1}} - \sum_{i=1}^{n} \frac{y_i w_i}{(x_i - b)^{c+1}} \cdot \sum_{i=1}^{n} \frac{1}{(x_i - b)^{2c}},$$

$$0 = \sum_{i=1}^{n} \frac{y_i w_i}{(x_i - b)^c} \cdot \sum_{i=1}^{n} \frac{\log_e(x_i - b)}{(x_i - b)^{2c}} - \sum_{i=1}^{n} \frac{y_i w_i \log_e(x_i - b)}{(x_i - b)^{c+1}} \cdot \sum_{i=1}^{n} \frac{1}{(x_i - b)^{2c}}.$$

(b) Solve the nonlinear system for the species with the following data. Use the weights $w_i = \log_e(y_i)$.

i	x_i	y_i
1	30.2	21.6
2	31.2	4.75
3	31.5	3.80
4	31.8	2.40

7. Find a nonzero solution for the following system to an accuracy of at least 10^{-12}, using Newton's method.

$$\begin{aligned} x_1 + 10x_2 &= 0, \\ x_3 - \sqrt{|x_4|} &= 0, \\ (x_2 - 2x_3)^2 - 1 &= 0, \\ x_1 - x_4 &= 0. \end{aligned}$$

7.5 The Method of Successive Substitution*

Note one difference between Newton's method and the secant method: Newton's method obtains the next iterate $x^{[i+1]}$ from $x^{[i]}$ only, while the secant method needs both $x^{[i]}$ and $x^{[i-1]}$. We say that Newton's method is a *one-point* iteration, while the secant method is a *two-point* scheme. The analysis for one-point methods is straightforward and considerably simpler than the analysis for multipoint methods.

We begin by rewriting (7.4) as

$$x = G(x) \tag{7.19}$$

in such a way that x is a solution of (7.4) if and only if it satisfies (7.19). There are many ways such a rearrangement can be done; as we will see, some are more suitable than others.

Equation (7.19) can be made into a one-point iterative scheme by substituting $x^{[i]}$ into the right side to compute $x^{[i+1]}$; that is

$$x^{[i+1]} = G(x^{[i]}). \tag{7.20}$$

We start with some $x^{[0]}$ and use (7.20) repeatedly with $i = 0,\ 1,\ 2,\ \ldots$. This approach is sometimes called the *method of successive substitution*, but is actually the general form of any one-point iteration. The obvious question is what happens as $i \to \infty$.

Let x^* satisfy equation (7.19). Such a point is called a *fixed point* of the equation. Then

$$\begin{aligned} x^{[i+1]} - x^* &= G(x^{[i]}) - x^* \\ &= G(x^{[i]}) - G(x^*). \end{aligned}$$

Using a Taylor expansion, we get

$$x^{[i+1]} - x^* = G'(x^*)(x^{[i]} - x^*) + \frac{1}{2}G''(x^*)(x^{[i]} - x^*)^2 + \ \ldots\ . \tag{7.21}$$

If $G'(x^*) \neq 0$, each iteration multiplies the magnitude of the error roughly by a factor $|G'(x^*)|$. The method is therefore of order one and converges if $|G'(x^*)| < 1$. If $|G'(x^*)| > 1$ the iteration may diverge.

Example 7.7 Both equations $x = 1 - \frac{1}{4}\sin\pi x$ and $x = 1 - \sin\pi x$ have a solution $x^* = 1$. In Table 7.5, the results of the first ten iterates for both these equations are shown, using (7.20) with the initial guess $x^{[0]} = 0.9$. Clearly the iterations converge in the second column of Table 7.5, although not very quickly. It is easy to verify that $|G'(x^*)| = \frac{\pi}{4} < 1$. On the other hand, the third column in the table reveals a divergent sequence. This is because $|G'(x^*)| = \pi > 1$. ■

Since Newton's method is a single-point iteration, (7.21) is applicable with

$$G(x) = x - \frac{f(x)}{f'(x)}$$

Table 7.5 Iterates for equations in Example 7.7.

Iterates	$x = 1 - 0.25 \sin \pi x$	$x = 1 - \sin \pi x$
$x^{[0]}$	0.9000	0.9000
$x^{[1]}$	0.9227	0.6910
$x^{[2]}$	0.9399	0.1747
$x^{[3]}$	0.9531	0.4784
$x^{[4]}$	0.9633	0.0023
$x^{[5]}$	0.9712	0.9928
$x^{[6]}$	0.9774	0.9773
$x^{[7]}$	0.9823	0.9288
$x^{[8]}$	0.9861	0.7782
$x^{[9]}$	0.9891	0.3582
$x^{[10]}$	0.9914	0.0976

and

$$G'(x) = \frac{f(x)f''(x)}{(f'(x))^2}.$$

Now $G'(x^*) = 0$ and the situation is changed. Provided $f'(x^*) \neq 0$,

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

and the magnitude of the error is roughly squared in each iteration; that is, the method has order of convergence two.

The condition $f'(x^*) \neq 0$ means that x^* is a simple root, so second order convergence holds only for that case. For roots of higher multiplicity, we reconsider the analysis. Suppose that f has a root of multiplicity two at x^*. From Taylor's theorem we know that

$$f(x) = \frac{1}{2}(x - x^*)^2 f''(x^*) + \ \ldots$$
$$f'(x) = (x - x^*)f''(x^*) + \ \ldots$$

so that

$$\lim_{x \to x^*} \frac{f(x)f''(x)}{(f'(x))^2} \cong \frac{1}{2}.$$

We expect then that near a double root, Newton's method will converge with order one, and that the error will be reduced by a factor of one half on each iteration.

It can also be shown (in Exercise 5 at the end of this section) that for roots of higher multiplicity we still get convergence, but that the rate gets slower as the multiplicity goes up. For roots of high multiplicity, Newton's method becomes quite inefficient.

These arguments can be extended formally to nonlinear systems. Using vector notation, we write the problem in the form

$$\mathbf{x} = \mathbf{G}(\mathbf{x}).$$

Then, starting with some initial guess $\mathbf{x}^{[0]}$, we compute successive values by

$$\mathbf{x}^{[i+1]} = \mathbf{G}(\mathbf{x}^{[i]}). \tag{7.22}$$

From (7.22), we get

$$\mathbf{x}^{[i+1]} - \mathbf{x}^{[i]} = \mathbf{G}(\mathbf{x}^{[i]}) - \mathbf{G}(\mathbf{x}^{[i-1]}),$$

and from the multi-dimensional Taylor's theorem,

$$\mathbf{x}^{[i+1]} - \mathbf{x}^{[i]} = \mathbf{G}'(\mathbf{x}^{[i]})(\mathbf{x}^{[i]} - \mathbf{x}^{[i-1]}) + O(||\mathbf{x}^{[i]} - \mathbf{x}^{[i-1]}||^2),$$

where $\mathbf{G}'$ is the Jacobian associated with $\mathbf{G}$. If $\mathbf{x}^{[i+1]}$ and $\mathbf{x}^{[i]}$ are so close that we can neglect higher order terms,

$$||\mathbf{x}^{[i+1]} - \mathbf{x}^{[i]}|| \;\cong\; ||\mathbf{G}'(\mathbf{x}^{[i]})||\;||\mathbf{x}^{[i]} - \mathbf{x}^{[i-1]}||. \tag{7.23}$$

If $||\mathbf{G}'(\mathbf{x}^{[i]})|| < 1$, then the difference between successive iterates will diminish, suggesting convergence.

It takes a good bit of work to make this into a precise and provable result and we will not do this here. The intuitive rule of thumb, which can be justified by precise arguments, is that if $||\mathbf{G}'(\mathbf{x}^*)|| < 1$, and if the starting value is close to a root x^*, then the method of successive substitutions will converge to this root. The order of convergence is one. The inequality (7.23) also suggests that the error in each iteration is reduced by a factor of $||\mathbf{G}'(\mathbf{x}^{[i]})||$, so that the smaller this term, the faster the convergence.

The simple form of the method of successive substitution, as illustrated in Example 7.7, is rarely used for single equations. However, for systems there are several instances where the approach is useful. For one of them we need a generalization of (7.22). Suppose that $\mathbf{A}$ is a matrix and we want to solve

$$\mathbf{A}\mathbf{x} = \mathbf{G}(\mathbf{x}). \tag{7.24}$$

Then, by essentially the same arguments, we can show that the iterative process

$$\mathbf{A}\mathbf{x}^{[i+1]} = \mathbf{G}(\mathbf{x}^{[i]}) \tag{7.25}$$

converges to a solution of (7.24), provided that $||\mathbf{A}^{-1}||\ ||\mathbf{G}'(\mathbf{x}^*)|| < 1$ and $\mathbf{x}^{[0]}$ is sufficiently close to x^*.

Example 7.8 Find a solution of the system

$$\begin{aligned} x_1 + 3x_2 + 0.1x_3^2 &= 1, \\ 3x_1 + x_2 + 0.1\sin(x_3) &= 0, \\ -0.25x_1^2 + 4x_2 - x_3 &= 0. \end{aligned}$$

We first rewrite the system as

$$\begin{bmatrix} 1 & 3 & 0 \\ 3 & 1 & 0 \\ 0 & 4 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 - 0.1x_3^2 \\ -0.1\sin(x_3) \\ 0.25x_1^2 \end{bmatrix}$$

and carry out the suggested iteration. As a starting guess, we can take the solution of the linear system that arises from neglecting all the small nonlinear terms. The solution of

$$\begin{bmatrix} 1 & 3 & 0 \\ 3 & 1 & 0 \\ 0 & 4 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

is $(-0.1250,\ 0.3750,\ 1.5000)$ which we take as $\mathbf{x}^{[0]}$. This gives the iterates

$$\begin{aligned} \mathbf{x}^{[1]} &= (-0.1343,\ 0.3031,\ 1.2085), \\ \mathbf{x}^{[2]} &= (-0.1418,\ 0.3319,\ 1.3232), \\ \mathbf{x}^{[3]} &= (-0.1395,\ 0.3215,\ 1.2808). \end{aligned}$$

Clearly, the iterations converge, although quite slowly. ■

The above example works, because the nonlinear part is a small effect compared to the linear part. This is not uncommon in practice where a linear model can sometimes be improved by incorporating small nonlinear perturbations. There are other special instances from partial differential equations where the form of the equations assures convergence. We will see some of this in Chapter 13.

EXERCISES

1. Use the method of successive substitution to find the positive root of

$$x^2 - e^{0.1x} = 0$$

to three-digit accuracy. Are there any negative roots?

2. Estimate the number of iterations required to compute the solution in Example 7.8 to four significant digits.

3. Rewrite the equations in the form (7.22) and iterate to find a solution for

$$10x_1^2 + \cos(x_2) = 12,$$

$$x_1^4 + 6x_2 = 2,$$

to three significant digits.

4. Use the method suggested in Example 7.8 to find a solution, accurate to 10^{-2}, for the system of equations

$$10x_1 + 2x_2 + \tfrac{1}{10}x_3^2 = 1,$$

$$2x_1 + 3x_2 + x_3 = 0,$$

$$\tfrac{1}{10}x_1 + x_2 - x_3 = 0.$$

Estimate how many iterations would be required to get an accuracy of 10^{-6}.

5. Show that for $f(x) = x^n$, $n > 2$, Newton's method converges for x near zero with order one. Show that the error is reduced roughly by a factor of $\frac{n-1}{n}$ in each iteration.

6. Consider the nonlinear system

$$3x_1 - \cos(x_2x_3) - \frac{1}{2} = 0,$$

$$x_1^2 - 81(x_2 + 0.1)^2 + \sin x_3 + 1.06 = 0,$$

$$e^{-x_1x_2} + 20x_3 + \frac{10\pi - 3}{3} = 0.$$

(a) Show that the system can be changed into the following equivalent problem

$$x_1 = \frac{1}{3}\cos(x_2x_3) + \frac{1}{6},$$

$$x_2 = \frac{1}{9}\sqrt{x_1^2 + \sin x_3 + 1.06} - 0.1,$$

$$x_3 = -\frac{1}{20}e^{-x_1x_2} - \frac{10\pi - 3}{60}.$$

(b) Use the method given by (7.25) to find a solution for the system in (a), accurate to 10^{-5}.

7.6 Minimization Methods*

A conceptually simple way of solving the multi-dimensional root-finding problem is by *minimization*. We construct

$$\Phi(x_1,\ x_2,\ \ldots,\ x_n) = \sum_{i=1}^{n} f_i^2(x_1,\ x_2,\ \ldots,\ x_n) \tag{7.26}$$

and look for a minimum of Φ. Clearly, any solution of (7.15) will be a minimum of Φ.

Optimization, with the special case of minimization, is an extensive topic that we consider here only in connection with root-finding. As with all nonlinear problems, it involves many practical difficulties. The main obstacle to minimization is the distinction between a *local* minimum and a *global* minimum. A global minimum is the place where the function Φ takes on its smallest value, while a local minimum is a minimum only in some neighborhood. In Figure 7.10, we have a function with one global and several local minima. In minimization it is usually much easier to find a local minimum than a global one. In root-finding, we unfortunately need the global minimum and any local minimum, at which Φ is not zero is of no interest.

To find the minimum of a function, we can take its derivative and then use root-finding algorithms. But this is of limited use and normally we approach the problem more directly. For minimization in one variable, we can use search methods reminiscent of bisection, but now we need three points at each step. The process is illustrated in Figure 7.11. We take three points, perhaps equally spaced, and compute the value of Φ. If the minimum value is at an endpoint, we proceed in the direction of the smallest value until

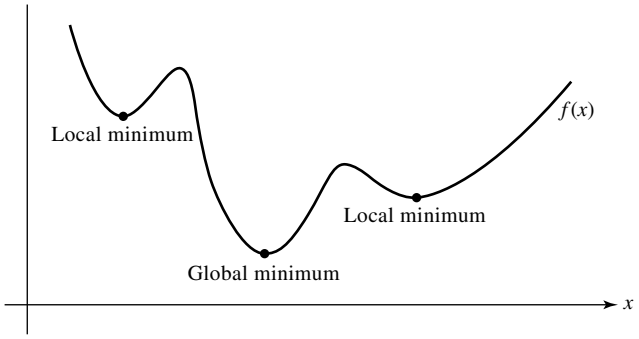

Figure 7.10
Global and local minima of a function.

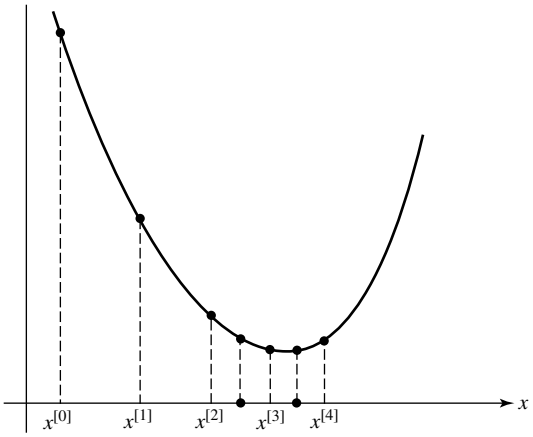

Figure 7.11 One-dimensional minimization.

the minimum occurs at the interior point. When this happens, we reduce the spacing and search near the center, continuing until the minimum is located with sufficient accuracy. Alternatively, if we have three trial points we can fit a parabola to these points and find its minimum. This will give us an approximation which can be improved by further iterations.

The main use of one-dimensional minimization is for the solution of problems in many dimensions. To find a minimum in several dimensions, we use an iterative approach in which we choose a direction, perform a one-dimensional minimization in that direction, then change direction, continuing until the minimum is found. The point we find is a local minimum, but because of computational error this minimum may be attainable only within a certain accuracy. The main difference between minimization algorithms is the choice of the direction in which we look. The process is easily visualized in two dimensions through contour plots.

In one approach we simply cycle through all the coordinate directions $x_1,\ x_2,\ \ldots,\ x_n$ in some order. This gives the iterates shown in Figure 7.12.

Example 7.9 Find a minimum point of the function

$$f(x,\ y) = x^2 + y^2 - 8x - 10y + e^{-xy/10} + 41,$$

using the strategy of search in the coordinate axes directions.

For the lack of any better guess, we start at $(0,\ 0)$ and take steps of unit length in the y-direction. After a few steps, we find

$$f(0,\ 4) = 18,\ f(0,\ 5) = 17,\ f(0,\ 6) = 18,$$

indicating a directional minimum in the interval $4 \le y \le 6$. We fit the three points with a parabola and find its vertex, which in this case is at $y = 5$.

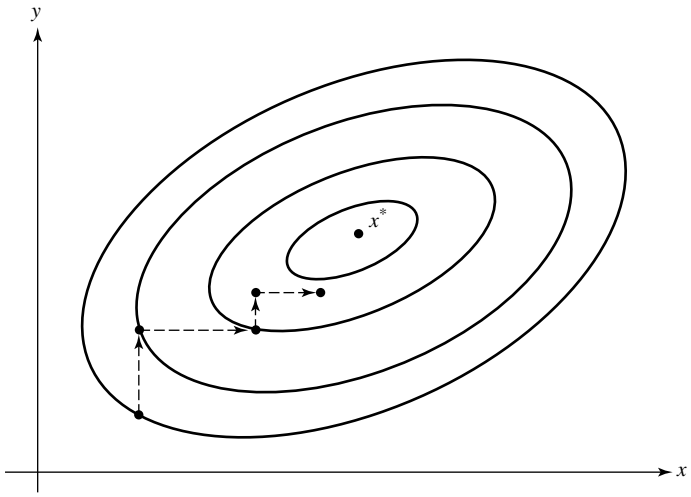

Figure 7.12
Minimization in direction of the coordinate axes.

We now change direction and continue the search. After a few more computations, we find that

$$f(3,\ 5) = 1.2231,\ f(4,\ 5) = 0.1353,\ f(5,5) = 1.0821,$$

with a resulting directional minimum at $x = 4.0347$.

Returning once more to the y-direction, we conduct another search, now with a smaller step size. We get

$$f(4.0347,\ 4.5) = 0.4139,\ f(4.0347,\ 5) = 0.1342,\ f(4.0347,\ 5.5) = 0.3599.$$

The computed directional minimum is now at $y = 5.0267$. Obviously, the process can be continued to get increasingly better accuracy.

If we stop at this point, the approximation for the minimal point is $(4.0347,\ 5.0267)$. This compares well with the more accurate result[1] $(4.0331, 5.0266)$.

In another way we proceed along the gradient

$$\Delta\Phi = \begin{bmatrix} \dfrac{\partial\,\Phi}{\partial\,x_1} \\ \dfrac{\partial\,\Phi}{\partial\,x_2} \\ \vdots \\ \dfrac{\partial\,\Phi}{\partial\,x_n} \end{bmatrix}.$$

[1]This result was produced by the MATLAB function fmins.

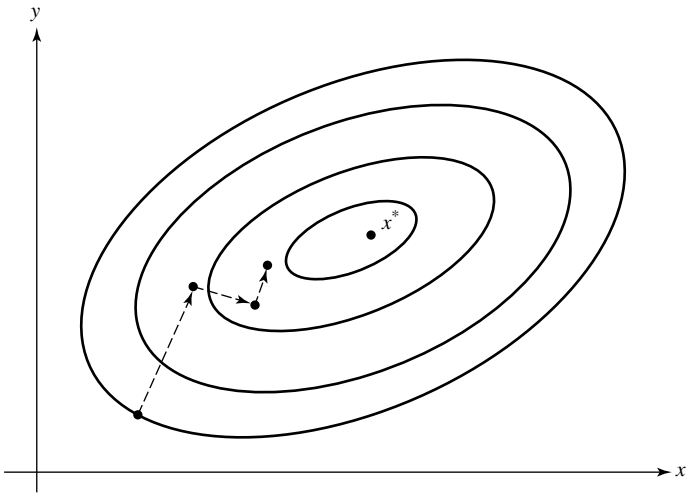

Figure 7.13
Minimization by steepest descent.

This is called the *steepest descent method* because moving along the gradient reduces Φ locally as rapidly as possible (Figure 7.13).

In a more brute force search we systematically choose a set of points and use the one where Φ is smallest for the next iteration. If triangles (or simplexes in more than two dimensions) are used, we can compute the value of Φ at the corners and at the center. If the minimum occurs at a vertex, we take this as the new center. We proceed until the center has a smaller value than all the vertices. When this happens, we start to shrink the triangle and repeat until the minimum is attained. (Figure 7.14).

There are many other more sophisticated minimization methods that one can think of and that are sometimes used. But, as in root-finding, there are always some cases that defeat even the best algorithm.

Minimization looks like a safe road toward solving nonlinear systems, and this is true to some extent. We do not see the kind of aimless wandering about that we see in Newton's method, but a steady progress toward a minimum. Unfortunately, minimization is not a sure way of getting the roots either. The most obvious reason is that we may find a local minimum instead of a global minimum. When this happens, we need to restart to find another minimum. Whether or not this eventually gives a solution of (7.15) is not clear; in any case, a lot of searching might have to be done. A second problem is that at a minimum all derivatives are zero, so it acts somewhat like a double root. When the minimum is very flat, it can be hard to locate it with any accuracy.

The simplex search illustrated in Figure 7.14 is easy to visualize, but not very efficient because each iteration requires $n + 1$ new values of Φ. In practice we use more sophisticated strategies that reduce the work considerably. For a simple discussion of this, see the elementary treatment of Murray [19]. For a more up-to-date and thorough analysis of the methods and difficulties in optimization, see Dennis and Schnabel [7].

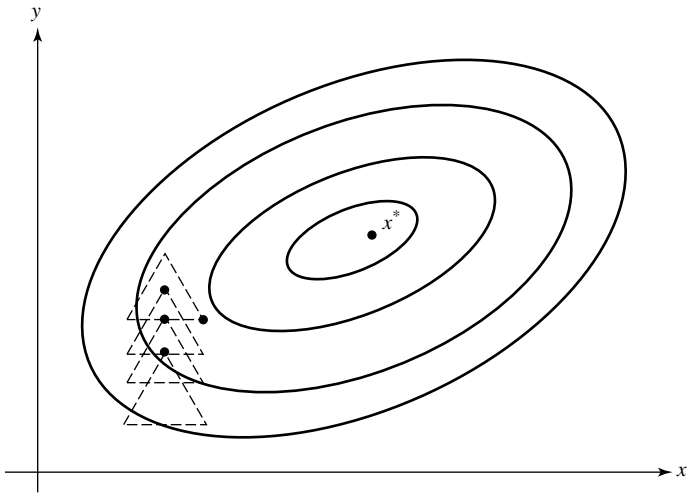

Figure 7.14
Minimization using a simplex.

EXERCISES

1. Set up the system in Exercise 3, Section 7.5 as a minimization problem of the form given by (7.26). Plot the contour graph for the function to be minimized.

2. Starting at (0, 0), visually carry out three iterations with the method of steepest descent for the problem in the previous exercise.

3. Starting at (0, 0), visually carry out three iterations with the descent along the coordinate axes for the problem in Exercise 1.

4. Elaborate a strategy for using the simplex-searching method for the minimization problem.

5. Assume that for all i, $f_i'(x^*) \neq 0$ at the solution x^* of (7.15). Suppose that the inherent computational errors are of order $O(\varepsilon)$. What is the order of accuracy with which we can expect to locate the minimum of Φ in (7.26)?

6. Instead of minimizing the Φ in (7.26), we could minimize other functions such as

$$\Phi(x_1,\ x_2,\ \ldots,\ x_n) = \sum_{i=1}^{n} |f_i(x_1,\ x_2,\ \ldots,\ x_n)|.$$

Do you think that this is a good idea?

7. Set up the system in Example 7.1 as a minimization problem. Then, starting from (0, 0), carry out four steps of minimization in the direction of the coordinate axes.

8. Repeat Exercise 7, using four steps with the steepest descent method.

9. Locate an approximate root of the system

$$\begin{aligned} 10x + y^2 + z^2 &= 1, \\ x - 20y + z^2 &= 2, \\ x + y + 10z &= 3. \end{aligned}$$

Find a good first guess for the solution, then use minimization to improve it.

10. Implement a search procedure for finding the minimum of a function of n variables, analogous to the method illustrated in Figure 7.14, but using the 2^n corners of a n-dimensional cube as sample points. Select a set of test examples to investigate the effectiveness of this algorithm. Does this method have any advantages or disadvantages compared to fmins?

11. Use the program created in the previous exploration to implement an n-dimensional root-finder.

7.7 Roots of Polynomials*

In our discussion of Gaussian quadrature, we encountered the need for finding roots of orthogonal polynomials. This is just one of many instances in numerical analysis where we want algorithms for approximating the roots of polynomials. These roots can of course be found by the standard methods, but there are several reasons why we need to pay special attention to polynomials. On one hand, because of the simple nature of polynomials, some of the difficulties mentioned in Section 7.3 can be overcome and we can find generally useful and efficient algorithms. A little bit of analysis often gives us information that is not always available for general functions. For example, it is possible to locate roots roughly. If we write the polynomial equation in the form

$$x^n = -\frac{a_{n-1}}{a_n}x^{n-1} - \frac{a_{n-2}}{a_n}x^{n-2} - \ \ldots \ - \frac{a_0}{a_n}, \tag{7.27}$$

then it follows easily that, provided $|x| \geq 1$,

$$|x| \ \leq \ \sum_{i=0}^{n-1} \left| \frac{a_i}{a_n} \right|. \tag{7.28}$$

This shows that the roots must be confined to some finite part of the complex plane, eliminating much of the need for extensive searching.

Also, once a root, say z, has been found, it can be used to simplify the finding of the other roots. The polynomial $p_n(x)/(x - z)$ has all the roots of $p_n(x)$ except z. This lets us reduce the degree of the polynomial every

time a root is found. Reduction of the degree, or *deflation*, can be done by *synthetic division*, which can be implemented recursively by

$$\frac{p_n(x)}{x-z} = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \ \ldots\ + b_0,$$

where the b_i are computed by

$$\begin{aligned} b_{n-1} &= a_n, \\ b_{n-i} &= a_{n-i+1} + zb_{n-i+1}, \ \ i = 2, 3, \ \ldots\ , \ n. \end{aligned} \tag{7.29}$$

On the other hand, polynomial root-finding has special needs. In many applications where polynomials occur we are interested not only in the real roots, but all the roots in the complex plane. This makes it necessary to extend root-finding methods to the complex plane.

While some of the methods we have studied can be extended to find complex roots, it is not always obvious how to do this. An exception is Muller's method if we interpret all operations as complex.

Example 7.10 Find all five roots $z_1, \ z_2, \ \ldots\ , \ z_5$ of the polynomial

$$p_5(x) = x^5 - 2x^4 - \frac{15}{16}x^3 + \frac{45}{32}x^2 + x + \frac{3}{16}.$$

We use equation (7.8) with a stopping criteria $|x_3 - x_2| < 10^{-8}$ and initial points $x^{[0]} = -s$, $x^{[1]} = 0$, and $x^{[2]} = s$. Here $s = 5.53125$ is the right side of equation (7.28). Muller's method then produces the following sequence that converges to a complex root:

$$\begin{aligned} x^{[3]} &= -0.00020641435017, \\ x^{[4]} &= -0.00489777122243 + 0.04212476080079\ i, \\ x^{[5]} &= -0.01035051976510 + 0.06760459238812\ i, \\ x^{[6]} &= -0.31296331041835 + 0.08732823246456\ i, \\ x^{[7]} &= -0.34558627541781 + 0.13002738461112\ i, \\ x^{[8]} &= -0.36010841180409 + 0.15590007220620\ i, \\ x^{[9]} &= -0.35624447399641 + 0.16301914734518\ i, \\ x^{[10]} &= -0.35606105305313 + 0.16275737859175\ i, \\ x^{[11]} &= -0.35606176171231 + 0.16275838282328\ i, \\ x^{[12]} &= -0.35606176174733 + 0.16275838285138\ i. \end{aligned}$$

We take the last iteration as the first root.[2] Applying the synthetic division procedure given in (7.29) with $z_1 = x^{[12]}$, the resulting deflated polynomial is:

$$\begin{aligned} p_4(x) = x^4 &+ (-2.35606176174733 + 0.16275838285138i)x^3 \\ &+ (-0.12508678951512 - 0.44142083877717i)x^2 \\ &+ (1.52263356452234 + 0.13681415794943i)x \\ &+ 0.43558075942153 + 0.19910708652543i. \end{aligned}$$

Repeating the application of Muller's method, now using initial points $x_0 = -s$, $x_1 = 0$, and $x_2 = s$, with $s = 4.82817669929469$, we get the second root after nine steps at

$$z_2 = 1.97044607872988.$$

Again the deflated polynomial for z_2 is

$$\begin{aligned} p_3(x) = x^3 &+ (-0.38561568301745 + 0.16275838285138i)x^2 \\ &+ (-0.88492170001360 - 0.12071422150726i)x \\ &- 0.22105692925244 - 0.10104670646647i. \end{aligned}$$

Continuing in the same fashion, we find

$$z_3 = -0.35606176174735 - 0.16275838285137i.$$

After deflation, we obtain the quadratic equation:

$$p_2(x) = x^2 - 0.74167744476478x - 0.62083872238239.$$

The two roots for $p_2(x)$ can then be obtained by applying the quadratic formula. They are

$$z_4 = 1.24167744476478$$

and

$$z_5 = -0.5.$$

■

Many special methods have been developed for the effective computation of the roots of a polynomial, but this is a topic that is technically difficult and we will not pursue it. Polynomial root finding was a popular topic in early numerical analysis texts. We refer the interested reader to

[2] We could use a shortcut here. The theory of polynomials tells us that for polynomials with real coefficients the roots occur in complex conjugate pairs. Thus, we can claim that another root is $z_3 = -0.35606176174735 - 0.16275838285137i$.

Blum[5], Isaacson and Keller[12], or similar books written in the 1960s. We describe only the *companion matrix* method, which is easy to understand and can be used if we need to deal with this special problem.

Consider the polynomial

$$p_n(\lambda) = \lambda^n + a_{n-1}\lambda^{n-1} + \ \ldots\ + a_1\lambda + a_0. \tag{7.30}$$

Then the companion matrix of this polynomial is

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & \\ 0 & 0 & 0 & 1 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 \\ -a_0 & -a_1 & \cdots & & -a_{n-2} & -a_{n-1} \end{bmatrix}. \tag{7.31}$$

One can then show that the determinant

$$\det(\mathbf{C} - \lambda\,\mathbf{I}) = (-1)^n p_n(\lambda\,), \tag{7.32}$$

so that the roots of the polynomial in (7.30) are the eigenvalues of its companion matrix. Anticipating results in the next chapter, we know that there are effective methods to solve this eigenvalue problem. The companion matrix method therefore gives us a convenient way to get the complex roots of any polynomial.

Now, as we will also see in the next chapter, finding the eigenvalues of a matrix often involves computing the roots of a polynomial. So it seems we have cheated, explaining how to find the roots of a polynomial by some other method that does the same thing! But from a practical viewpoint this objection has little force. Programs for the solution of the eigenvalue problem are readily available; these normally use polynomial root-finders (using methods we have not described here), but this is a matter for the expert. If, for some reason, we need to find the roots of a polynomial, the companion matrix is the most convenient way to do so.

Example 7.11 The Legendre polynomial of degree four is

$$P_4(x) = \frac{35x^4 - 30x^2 + 3}{8}.$$

Its companion matrix is

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -3/35 & 0 & 30/35 & 0 \end{bmatrix}.$$

Using the techniques for matrix eigenvalue problems that we will discuss in the next chapter, we find that the eigenvalues of $\mathbf{C}$ are ± 0.8611 and ± 0.3400. These are then the roots of P_4, which, as expected, are all real. They are also the quadrature points for a four-point Gauss-Legendre quadrature over $[-1,\ 1]$.

EXERCISES

1. By finding bounds on the derivative, show that the polynomial

$$p_3(x) = x^3 - 3x^2 + 2x - 1$$

has no real root in $[-0.1,\ 0.1]$. Can you extend this result to a larger interval around $x = 0$?

2. Prove that the polynomial in Exercise 1 has at least one real root.

3. Let z be a root of the polynomial

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ \ldots\ + a_0.$$

Show that the recursive algorithm suggested in (7.29) produces the polynomial

$$p_{n-1}(x) = \frac{p_n(x)}{x - z}.$$

4. Suppose that in Exercise 3, z is complex; the process then creates a polynomial p_{n-1} with complex coefficients. How does this affect the usefulness of deflation in root-finding?

5. Use Newton's method to find a real root of the polynomial in Exercise 1. Then use deflation to find the other two roots.

6. Prove (7.32).

7. Use the companion matrix method to check the results of Example 7.10.

8. Consider the ellipse

$$x^2 + 2(y - 0.1)^2 = 1$$

and the hyperbola

$$xy = \frac{1}{a}.$$

(a) For $a = 10$, find all the points at which the ellipse and the hyperbola intersect. How can you be sure that you have found all the intersections?

(b) For what value of a is the positive branch of the hyperbola tangent to the ellipse?

7.8 Selecting a Root-Finding Method*

Even though there are many methods for solving nonlinear equations, selecting the most suitable algorithm is not a simple matter. To be successful one has to take into account the specific problem to be solved and choose or modify the algorithm accordingly. This is especially true for systems of equations.

For a single equation, one can write fairly general software that deals effectively with simple roots and functions that do not have the pathological behavior exhibited in Figure 7.8. Here is some of the reasoning we might use in designing such software: We assume that to use this software we need only to supply the name of the function, a rough location of the root of interest, and an error tolerance, and that the program returns the root to an accuracy within this tolerance. The first step is to decide on the basic root-finding method. Most likely, our choice would be between Newton's method and the secant method. The method of false position often converges very slowly, therefore we rule it out for general use. Newton's method converges faster than the secant method, but requires the computation of the derivative, so each step of the secant method takes less work. If the derivative is easy to evaluate we might use Newton's method, but in a general program we cannot assume this. Also, because the error in two applications of the secant method is reduced roughly by

$$\begin{aligned} e_i &= e_{i-1}^{1.62} \\ &= (e_{i-2}^{1.62})^{1.62} \\ &= e_{i-2}^{2.6}, \end{aligned}$$

we see that two iterations of the secant method are better than one application of Newton's method. Since the secant method additionally relieves

the user of having to supply the derivative, we might go with it. To ensure convergence of the method, we will design the program so that the iterates never get outside a region known to contain a root. We can ask that the user supply two guesses that enclose a root or just one guess of the approximate solution. In the latter case, we need to include in our algorithm a part that searches for a sign change in the solution. Once we have a starting point and an enclosure interval, we apply the secant method repeatedly to get an acceptable answer. The first problem is that the method may not converge and give iterates that are outside the enclosure interval. To solve this problem, we test the prospective iterate and, if it is outside the interval, apply one bisection. After that, we revert to the secant method. In this way we always have two points that bracket the solution, and the process should converge. The next issue is when to terminate the iteration. A suggestion is to compare two successive iterates and stop if they are within the error tolerance. This, however, does not guarantee the accuracy unless the two last iterates bracket the solution; what we really want is a bracketing interval within the tolerance level. To achieve this we might decide to apply the secant method until two iterates are within half the tolerance, and if the two last iterates do not bracket the solution, take a small step (say the difference between the last two iterates) in the direction toward the root (Figure 7.15). This often resolves the situation, although we still have to decide what to do if it does not.

A number of widely available programs, including MATLAB's `fzero` have been designed in such a manner. Most of them combine several simple ideas in a more sophisticated way than what we have described here.

For systems of equations, writing general purpose software is much more difficult and few attempts have been made. Most software libraries just give the basic algorithms, Newton's method, minimization, or other algorithms, leaving it to the user to find good starting points and to evaluate the results. If we want to write a more automatic program, we need safeguards against

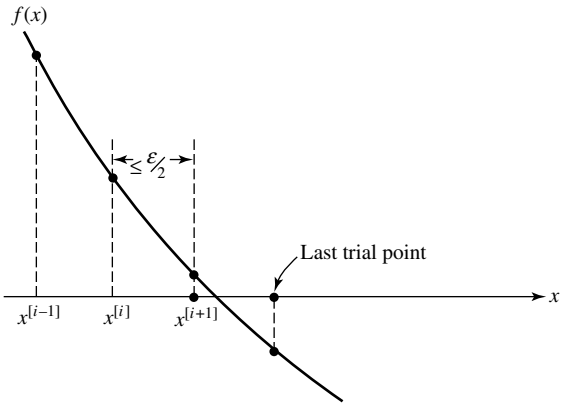

Figure 7.15
Using the secant method with bracketing of the root.

iterating without any progress. Typically, we can do two things. The first is to check that we are getting closer to a solution. We can do this by accepting the next iteration only if

$$||\mathbf{F}(\mathbf{x}^{[i+1]})|| < ||\mathbf{F}(\mathbf{x}^{[i]})||.$$

If we are using Newton's method, the second thing we can do is put a limit on the number of iterations. If the method converges, it does so quite quickly; a large number of iterations is usually a sign of trouble. For minimization on the other hand, a fair number of iterations is not unusual. To be effective for a large number of cases, multi-dimensional root-finders have to have very sophisticated search strategies. Combining a good strategy with knowledge of the specific case often gives some solutions of nonlinear systems. But unless the problem has some illuminating origin, we generally will not know that we have found all solutions. This is the harder problem for which we have no good answer.

Actually, the last statement has to be qualified. There are indeed methods that, in principle, can give us a complete solution to the nonlinear problem. These methods are based on an interval arithmetic approach, constructing functions whose domains and ranges are intervals. Interval programs have been produced that work quite well in three or four dimensions but, unfortunately, they become impractical for larger n. This is still a matter of research, but for the moment the problem is still not solved.

Exercises

1. Under what conditions are two iterates of the secant method on the same side of the root?
2. Can you think of a situation where the algorithm for solving a single equation sketched in this section might fail?
3. Is it possible for Newton's method to be on a convergent track, yet require a large number of iterations to get high accuracy?