Chapter 4
The Roots of f(x)

4.1 Introduction

Perhaps one of the most fundamental of all basic mathematical tasks is the seemingly straight forward problem: determine the value(s) of x, say α, which solves the equation $f(x) = 0$. Specific values of α are called the roots or zeros of the function or solutions of the equation, in either case, $f(\alpha) = 0$. In the special cases of $f(x) = ax + b$ or $f(x) = ax^2 + bx + c$, the roots are well known from algebra. On the other hand, for many simple functions, say $f(x) = e^{-x} - x$, it is not possible to determine an algebraic expression for the solution of $e^{-x} - x = 0$.

The geometric interpretation of $f(\alpha) = 0$, with α real, should be clear. The graph of $f(x)$ either crosses the horizontal axis at $x = \alpha$ or the graph just touches the horizontal axis at $x = \alpha$. In the case of differentiable functions, the graph may have a horizontal tangent at the point $(\alpha, 0)$ on the horizontal axis. This indicates a multiple root.

Many root finding situations arise from physical problems. For example, consider a sphere of radius R and density δ lb/ft^3 floating at a depth H in water of density 62.4 lb/ft^3. H is the distance from the bottom of the sphere to the water line. Archimedes' principle states that the sphere will be submerged to a depth where the weight of the sphere is equal to the weight of the water displaced. Using calculus to find the displaced volume results in the equation

$$\frac{4}{3}\pi \mathrm{R}^3\delta = \pi(\mathrm{H}^2\mathrm{R} - \frac{1}{3}\mathrm{H}^3)\ 62.4. \tag{4.1}$$

Rearranging terms gives a cubic polynomial in H

$$\mathrm{H}^3 - 3\mathrm{H}^2\mathrm{R} + 4\mathrm{R}^3\delta/62.4\ = 0,\ \text{where } 0 < \mathrm{H} < 2\mathrm{R}. \tag{4.2}$$

There are formulas to find the roots of a cubic polynomial such as (4.2); however, the actual use of these formulas is very cumbersome.

Virtually all schemes to find the roots of a function begin with a good understanding of the problem. For cubic polynomials we expect three roots: either three real roots or one real root and two complex values. For this example, the relationship between the density of the sphere and the density of water is a critical issue which will influence the roots of the cubic polynomial. The physical implications are obvious $-$ float or sink!

As a general rule, the first step of a root finding problem is to obtain qualitative data suggesting possible values for the roots of a function. A graph or a table of values will frequently provide the data necessary to implement a particular numerical algorithm.

In the following sections we will consider some basic methods. Based on initial estimates, all of the methods compute a finite sequence of numbers. The last number in the sequence is used as an approximate value for the root. In an ideal case, the method produces an accurate approximation within a few iterations.

4.2 The Bisection Method

One of the simplest root finding methods has its foundation in the properties of continuous functions. The bisection method assumes that $f(x)$ is continuous on a closed interval $[\,a,\,b\,]$ and that

$$f(a)f(b) \; < 0. \tag{4.3}$$

In other words, the graph of $f(x)$ must cross the x-axis an odd number of times within the interval $[\;a,\;b\;]$ to insure that the product $f(a)f(b)$ is negative. A negative product guarantees at least one root in the interval.

As the name suggests, the method begins by finding the midpoint of $[\,a,\,b\,]$, $c_1 = (a+b)/2$, which bisects $[\,a,\,b\,]$ into two intervals $I_{\mathrm{L}} = [\,a,\,c_1]$ and $I_{\mathrm{R}} = [\,c_1, b\,]$. If $f(c_1) = f(\alpha) = 0$ then we have found the true root, α; however, in most cases α will be in either the left or right subinterval, I_{L} or I_{R}.

The basic premise of the bisection method, a sign change on a closed interval, is employed to determine the location of the root as follows:

If $f(a)f(c_1) \; < 0$ then α is in $I_{\mathrm{L}} = [\,a,\,c_1]$.

If $f(a)f(c_1) \; > 0$ then α is in $I_{\mathrm{R}} = [\,c_1,\,b\,]$.

The midpoint value c_1 is the first approximation to the root α. It is a simple task to bound the error in c_1, $|\,\alpha - c_1\,|$. Observe that $|\,\alpha - c_1\,|$ is less than or equal to $c_1 - a$ or $b - c_1$. In other words, the error is bounded by one-half the length of the original interval $[\,a,\,b\,]$. For example, the actual root, α, could be very close to b and $b - c_1 = \frac{1}{2}(b-a)$, thus

$$|\,\alpha - c_1\,| \;\le \frac{1}{2}(b-a). \tag{4.4}$$

To keep track of values, the new interval containing α may be identified as $[\;a_2,\,b_2\,]$. We continue the process by finding a second midpoint value, c_2, of either I_{L} or I_{R}, as appropriate. The error bound becomes, $|\;\alpha - c_2\;| \;\le \frac{1}{2}(b_2 - a_2) \le \frac{1}{2^2}(b-a)$. The data generated by the bisection method is a sequence of midpoint values, $\{c_i\}$, $i = 1, 2, \dots$ with error bounds given by

$$|\,\alpha - c_i| \;\le \frac{1}{2^i}(b-a). \tag{4.5}$$

A simple example will illustrate the steps in the bisection method. By inspection it is obvious that $f(x) = x^2 - 2$ has a root in the interval $[\,1,\,2\,]$ and $\alpha = \sqrt{2}$. The following table provides data for three iterations of the bisection method.

i	a_i	c_i	b_i	$f(a_i)$	$f(c_i)$	$f(b_i)$	*Error Bound*
1	1	1.5	2	-1	.25	2	.5
2	1	1.25	1.5	-1	$-.4375$	.25	.25
3	1.25	1.375	1.5	$-.4375$	$-.1094$	.25	.125

The last line shows $c_3 = 1.375$ with $|\alpha - c_3| \leq 0.125$. In this case the error bound is a rather poor estimate for the actual error, $|\sqrt{2} - c_3| = 0.039$.

The inequality, (4.5), may be employed to determine the number, k, of midpoint computations or bisections needed to reach a specified error tolerance with the relationship

$$|\,\alpha - c_k\,| \leq \frac{1}{2^k}(b-a) \leq \text{TOL}.$$

Solving this inequality for k gives

$$k \geq \frac{\ln(b-a) - \ln(\text{TOL})}{\ln(2)}. \tag{4.6}$$

For example, with $b - a = 1$ and TOL $= 1.0000e{-}006$, we find

$$k \geq \frac{\ln(1) - \ln(.000001)}{\ln(2)} \approx 19.9 \approx 20.$$

With $k \geq 20$, c_{20} will meet the error tolerance for an initial interval of length one.

From a mathematical point of view, the bisection method will always converge to α. On the other hand, from a computational viewpoint, an error tolerance should not be selected to exceed the limitations of the machine implementing the method. Although reliable, the bisection method is rather slow with the error bound dropping by a factor of two for each iteration. Other methods will reduce the error bound or error estimate at a much faster rate.

Construction of an M-file to perform the bisection computations is not difficult. The following M-file, bisect.m, will compute a specified number of bisections for a given function and interval. If there is no sign change on the interval, MATLAB's **error** command will display an informational message and terminate computations. The MATLAB help feature will provide information on **error**, **feval** and the two uses of the **if** command.

M-file bisect.m

```
function bisect(n,fct1,a,b)
%BISECT Bisection Method
fa = feval(fct1,a);
fb = feval(fct1,b);
if fa*fb > 0
  error('Root not in interval provided.')
end
% display column headings, adjust for format
disp('     i                    a(i)               c(i)                 b(i)            err bound')
for i=1:n
  c  = (a+b)/2;
  fc = feval(fct1,c);
  disp([i,a,c,b,(b-a)/2])
        if fa*fc < 0
                b  = c;
                fb = fc;
        else
            a  = c;
            fa = fc;
        end
    end
```

Example: If the density of a sphere is $\delta = 15.6$ lb/ft^3, determine the depth H at which a sphere of radius R $= 2$ ft floats in water. Substituting into (4.2) gives

$$H^3 - 6H^2 + 8 = 0. \tag{4.7}$$

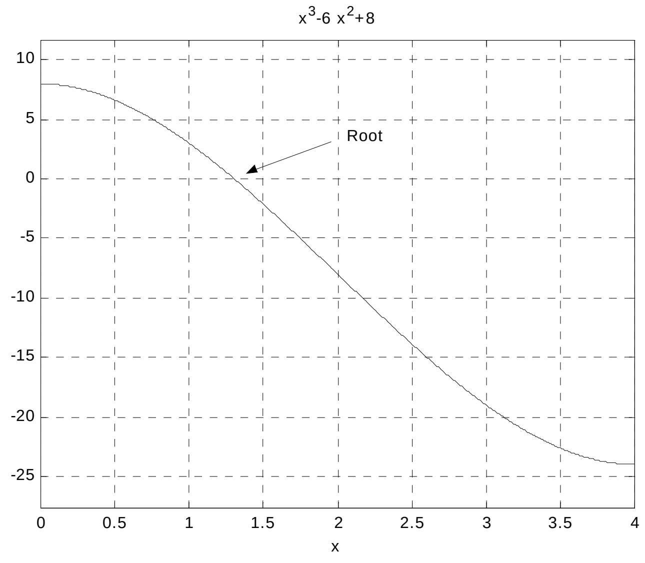

Figure 4.1 The Cubic Polynomial $H^3 - 6H^2 + 8$

Figure 4.1 reveals a root in the interval $[\,1,\ 1.5\,]$. With the polynomial (4.7) the active function in the M-file fct1.m, the command

```
>> bisect(10,'fct1',1,1.5);
```

will execute 10 iterations of the bisection method. The output of bisect.m is given in the following table. Note that **disp** can be used to display string data (the column headings).

```
      i             a(i)          c(i)          b(i)        err bound
1.0000e+000  1.0000e+000  1.2500e+000  1.5000e+000  2.5000e-001
2.0000e+000  1.2500e+000  1.3750e+000  1.5000e+000  1.2500e-001
3.0000e+000  1.2500e+000  1.3125e+000  1.3750e+000  6.2500e-002
4.0000e+000  1.2500e+000  1.2813e+000  1.3125e+000  3.1250e-002
5.0000e+000  1.2813e+000  1.2969e+000  1.3125e+000  1.5625e-002
6.0000e+000  1.2969e+000  1.3047e+000  1.3125e+000  7.8125e-003
7.0000e+000  1.3047e+000  1.3086e+000  1.3125e+000  3.9063e-003
8.0000e+000  1.3047e+000  1.3066e+000  1.3086e+000  1.9531e-003
9.0000e+000  1.3047e+000  1.3057e+000  1.3066e+000  9.7656e-004
1.0000e+001  1.3047e+000  1.3052e+000  1.3057e+000  4.8828e-004
```

Based on the data we can see that, $|\alpha - c_{10}| = |\alpha - 1.3052| \leq 0.000488$. In other words, $\alpha = 1.3052 \pm 0.000488$ so that c_{10} has three decimal places of accuracy. As you can see from this example, the bisection method slowly produces a small interval containing the root.

4.3 Newton's Method

Various methods to compute the roots of a function are based on approximating $f(x)$ with a line. For example, the so-called method of *false position* requires, just like bisection, a closed interval with a sign change; however, instead of using a midpoint computation to estimate the true root, false position computes the x-intercept of a line joining the end points $(a,\ f(a))$ and $(b,\ f(b))$. The x-intercept, x_1, is used to approximate α and to define $I_{\text{L}} = [a,\ x_1]$ and $I_{\text{R}} = [x_1,\ b]$. At this point the method parallels bisection with computations to determine if α is in I_{L} or I_{R}, followed by construction of a new line joining endpoints. A problem at the end of this chapter ask you to write an M-file to implement the method of false position.

Isaac Newton (1642-1727) is credited with a simple yet elegant procedure to approximate a root. *Newton's method* forgoes the need for a closed interval with a sign change and instead approximates $f(x)$ by a tangent line. All that is needed is a good initial estimate of α, say x_1. Newton's method begins by constructing a tangent line to the graph of $f(x)$ at the point $(x_1, f(x_1))$. Equation (3.1) with $a = x_1$ provides the standard form

$$f(x) \approx f(x_1) + f'(x_1)(x - x_1) \text{ for } x \text{ close to } x_1. \tag{4.8}$$

The x-intercept of the tangent line, $y = f(x_1) + f'(x_1)(x - x_1)$, is defined with coordinates $(x_2,\ 0)$. In other words

$$0 = f(x_1) + f'(x_1)(x_2 - x_1).$$

Solving for x_2 gives the basic structure of Newton's method

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \tag{4.9}$$

Figure 4.2 is a graphical representation of the first iteration of Newton's method.

The concept is easily replicated: construct a tangent line to the graph of $f(x)$ at the point $(x_2, f(x_2))$, define the x-intercept of the new tangent line as the point $(x_3,\ 0)$, and then solve for x_3,

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}.$$

In general terms, Newton's method becomes

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \text{ for } n = 1,\ 2,\ 3,\ \ldots\ . \tag{4.10}$$

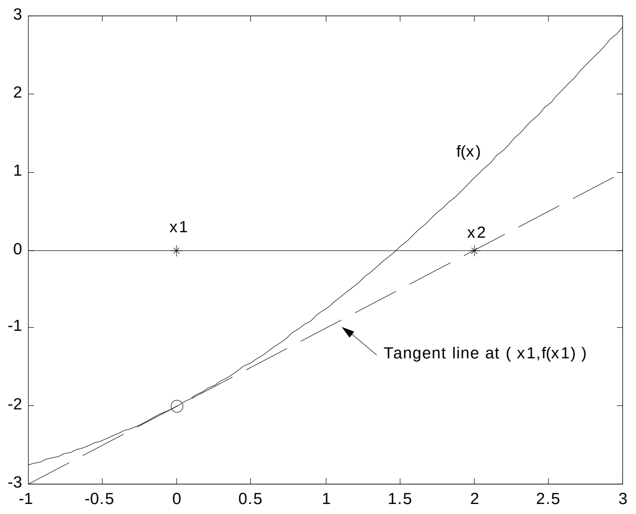

Figure 4.2 Newton's Method

Equation (4.10) produces a sequence of values $\{x_1,\ x_2,\ x_3,\ \ldots\ \}$ that continues until terminated by a method chosen by the user. Examples of termination criteria include: $1)$ stop at x_n for some specified n; $2)$ stop when x_n meets a specified error tolerance; or $3)$ stop if an unanticipated situation is encountered. As a practical matter, computer codes to implement Newton's method usually contain more than one termination criterion.

One source of difficulty with Newton's method is found in the denominator of the quotient $\frac{f(x_n)}{f'(x_n)}$. If $f'(x_n)$ happens to be zero for some n, the quotient is undefined and the procedure aborts. Geometrically, the graph of $f(x)$ has a horizontal tangent line at $(\ x_n, f(x_n))$ that will never intersect the x-axis.

Example: Find the roots of the cubic polynomial, (4.7), using Newton's method.

Recall $f(x) = x^3 - 6x^2 + 8$, so that $f'(x) = 3x^2 - 12x$. Substituting into (4.10) gives

$$x_{n+1} = x_n - \left[\frac{x_n^3 - \ 6x_n^2\ +\ 8}{3x_n^2 - 12x_n}\right] \text{ for } n = 1,\ 2,\ 3,\ \ldots\ . \tag{4.11}$$

Based on earlier computations with the bisection method we know that there is a root in the interval $[\,1,\ 1.5\,]$. Selecting $x_1 = 1.2$ as a starting value, the first iteration becomes

$$x_2 = x_1 - \left[\frac{x_1^3 - \ 6x_1^2\ +\ 8}{3x_1^2 - 12x_1}\right] = 1.2 - \left[\frac{(1.2)^3 - \ 6(1.2)^2 +\ 8}{3(1.2)^2 - 12(1.2)}\right] \approx 1.3079.$$

Likewise, the second iteration is

$$x_3 = x_2 - \left[\frac{x_2^3 - \ 6x_2^2 +\ 8}{3x_2^2 - 12x_2}\right] \approx 1.3079 - \left[\frac{(1.3079)^3 - \ 6(1.3079)^2 +\ 8}{3(1.3079)^2 - 12(1.3079)}\right] \approx 1.3054.$$

Although algebra may be used to simplify the right side of (4.11), further iterations with manual computations become tedious. An M-file to implement Newton's method will be presented later in this section. For the moment, symbolic computations with MATLAB may be used to implement Newton's method. The symbolic derivative of a function may be computed using MATLAB's **diff**. See >> **help sym/diff**. The following MATLAB commands illustrate how to reproduce the computations of x_2 and x_3 above. The variable **newt** represents the right hand side of Newton's algorithm. The **subs** command will allow us to replace the symbolic x in **newt** with a numerical value.

```
>> syms x
>> f = x^3 - 6*x^2 + 8;
>> df = diff(f);
>> newt = x - f/df
newt =
x-(x^3-6*x^2+8)/(3*x^2-12*x)
>> x2 = subs(newt,x,1.2)
x2 =
  1.3079e+000
>> x3=subs(newt,x,x2)
x3 =
  1.3054e+000
```

As we have seen, bounding the errors in the bisection method is a simple task. Although we cannot bound the errors in the case of Newton's method, it is possible to estimate the error at each iteration. Using Taylor's Theorem, see (3.7) and (3.8), with $n = 0$ gives

$$f(x) = f(\mathrm{a}) + f'(c)(x - \mathrm{a}) \text{ where } c \text{ is between } x \text{ and a.}$$

Replacing x by the true root, α, and a by x_n results in

$$f(\alpha) = 0 = f(x_n) + f'(c)(\alpha - x_n) \text{ where } c \text{ is between } \alpha \text{ and } x_n. \tag{4.12}$$

This equation contains the error in x_n: $\alpha - x_n$. Solving (4.12) for $\alpha - x_n$

$$\alpha - x_n = -\frac{f(x_n)}{f'(c)}. \tag{4.13}$$

If the sequence of Newton values is converging, the error $\alpha - x_n$ will be approaching zero. With c somewhere between α and x_n, we may assume that c is very close to x_n. (c is also close to α; however, α is unknown.) Using $c \approx x_n$ in (4.13) gives, with (4.10),

$$\alpha - x_n \approx -\frac{f(x_n)}{f'(x_n)} = x_{n+1} - x_n. \tag{4.14}$$

In other words, the error $\alpha - x_n$ is approximately equal to the difference $x_{n+1} - x_n$.

An M-file, newton.m, to implement Newton's method is given below. The program will terminate when the error estimate, $\alpha - x_n \approx x_{n+1} - x_n$, is less than a prescribed error tolerance. Since the error estimate for x_n depends on the next iteration x_{n+1}, we must use care in coding the procedure by introducing vectors to save the data. In addition, Newton's method requires the derivative of $f(x)$ which should be entered in an M-file, say fct1p.m.

M-file newton.m

```
function  r = newton(tol,fun,funp,x1)
%NEWTON Newton's algorithm
%      Tolerance, function, derivative,
%       and initial value must be given.
%      Maximum 20 iterations displayed.
max = 21; x(1) = x1;
y(1) = feval(fun,x(1));
yp(1) = feval(funp,x(1));
for i = 2:max
   x(i)  = x(i-1) - y(i-1)/yp(i-1);
   y(i)  = feval(fun,x(i));
   yp(i) = feval(funp,x(i));
   esterr(i-1) = abs(x(i)-x(i-1));
         if  esterr(i-1) < tol
                  fprintf('Error tolerance met at i = %2.0f\n',i-1)
                  imax = i; break;
         end
   imax = i;
end
disp('      i              x(i)              f(x(i))          x(i+1)-x(i)')
for k = 1:imax - 1
         disp([k,x(k),y(k),esterr(k)])
end
r = x(imax);
fprintf('Root is approximately  % 12.6e \n',r)
```

There are two new MATLAB commands in newton.m which should be explored using the help feature. They are **break** and **fprintf**. When the error tolerance is met the program terminates, prints results and reports the next Newton iteration as the root.

The previous example, $f(x) = x^3 - 6x^2 + 8$ and $f'(x) = 3x^2 - 12x$, with an error tolerance of 0.00001 and an initial value to 1.2 gives

```
>> r = newton(.00001,'fct1','fct1p',1.2);

Error tolerance met at i = 3
       i                  x(i)             f(x(i))        x(i+1)-x(i)
  1.0000e+000  1.2000e+000  1.0880e+000  1.0794e-001
  2.0000e+000  1.3079e+000 -2.6703e-002  2.5280e-003
  3.0000e+000  1.3054e+000 -1.3284e-005  1.2589e-006
Root is approximately  1.305407e+000
```

The output data shows that $x(3)$ satisfies the prescribed error tolerance. In other words, $|\alpha - x_3| \approx |x_4 - x_3| \approx 0.0000012589 \leq 0.00001$.

Since the numerical value of $x(4)$ is available from the error computations, $x(4)$ is reported as the approximate root with more accuracy expected. The M-file displays six decimal places for the final estimate. The rapid decrease in the error estimates, 10^{-1}, 10^{-3}, 10^{-6}, is indicative of the rate at which Newton's method converges. We will investigate the rate of convergence below.

Further Error Analysis

A three term Taylor representation centered at x_n will allow us to see how errors in Newton's algorithm are related . Recall

$$f(x) = f(x_n) + f'(x_n)(x - x_n) + \frac{1}{2} f''(c)(x - x_n)^2,$$

where again c is between x and x_n. Replacing x by α, the actual root, gives

$$f(\alpha) = 0 = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2} f''(c)(\alpha - x_n)^2$$

Assuming that $f'(\alpha)$ is not zero we may divide by $f'(x_n)$ to obtain

$$0 = \frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) + \frac{1}{2}\frac{f''(c)}{f'(x_n)}(\alpha - x_n)^2.$$

The first term on the right hand side looks familiar. Using Newton's method, (4.10), the previous expression becomes

$$0 = x_n - x_{n+1} + (\alpha - x_n) + \frac{1}{2}\frac{f''(c)}{f'(x_n)}(\alpha - x_n)^2$$

or

$$0 = \alpha - x_{n+1} + \frac{1}{2}\frac{f''(c)}{f'(x_n)}(\alpha - x_n)^2. \tag{4.15}$$

Equation (4.15) contains two error terms, $(\alpha - x_n)$ and $(\alpha - x_{n+1})$. Solving for the latter we find

$$\alpha - x_{n+1} = -\frac{1}{2}\frac{f''(c)}{f'(x_n)}(\alpha - x_n)^2. \tag{4.16}$$

This equation shows that the error term on the left, $\alpha - x_{n+1}$, is proportional to the square of the error term $\alpha - x_n$. For example, if the error $\alpha - x_n$ at step n is $1.0000e{-}003$ we expect that the error at step $n+1$, $\alpha - x_{n+1}$, will be approximately $1.0000e{-}006$. Recall our polynomial example. Equation (4.16) is the source of the statement that Newton's method is *quadratically* convergent.

The Secant Method

Another scheme, closely related to Newton's method, is the *secant method*. Instead of using the tangent line at x_1, the secant method begins with two estimates for α, say x_1 and x_2, and constructs a line between the points $(x_1,\ f(x_1))$ and $(x_2,\ f(x_2))$. Denoting the x-intercept of the secant line as x_3, it is an algebra task to show that the intercept is

$$x_3 = x_2 - f(x_2) \cdot \left[\frac{x_2 - x_1}{f(x_2) - f(x_1)}\right].$$

In general terms, the secant method is defined by the formula

$$x_{n+1} = x_n - f(x_n) \cdot \left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right] \text{ for } n = 2,\ 3,\ 4,\ \ldots \qquad (4.17)$$

The secant method does not converge as fast as Newton's method; however, the derivative of $f(x)$ is not needed. With proper programming, only one new function evaluation will be required at each iteration whereas Newton's method requires two function evaluations, $f(x_n)$ and $f'(x_n)$, per iteration. Using newton.m as a model, you should be able to write an M-file, say secant.m, to implement the secant method.

4.4 MATLAB Methods

MATLAB contains two commands to compute the roots of functions. The first, **roots**, is specifically written to find the roots of a polynomial. The actual method is beyond the scope of an introductory text. The second command, **fzero**, is designed to find the roots of $f(x)$ provided the function actually crosses the x-axis at the root. **fzero** uses a combination of methods: bisection, the secant method and inverse quadratic interpolation. Inverse linear interpolation will be discussed in Chapter 6.

roots

The coefficients of the various powers are the critical data in a polynomial. For example, the coefficients of the cubic polynomial $5x^3 - 8x^2 + 11$ may be entered in a row vector $[5\quad 8\quad 0\quad 11]$ where the coefficients are in descending order, beginning with the highest power, with a zero included for the missing linear term. This representation is a MATLAB convention for polynomials. The roots of this cubic polynomial may be found as follows:

```
>> roots([5,-8,0,11])
ans =
  1.2661e+000 +8.7024e-001i
  1.2661e+000 -8.7024e-001i
 -9.3212e-001
```

The coefficients may be entered as either a row or column vector. For a second example recall the cubic polynomial from the floating sphere example, $H^3 - 6H^2 + 8$.

```
>> c = [1,-6,0,8];
>> r = roots(c)
r =
  5.7588e+000
  1.3054e+000
 -1.0642e+000
```

Acceptable solutions are in the interval $0 < \mathrm{H} < 2\mathrm{R}$ or, with $\mathrm{R} = 2$, the interval $0 < \mathrm{H} < 4$.

To verify that the second value in the vector **r** is indeed the approximate root it is necessary to evaluate the polynomial defined by $c = [\,1, -6, 0\,, 8\,]$. Polynomials may be evaluated using the MATLAB command **polyval**. Inputs to **polyval** are the coefficient vector and the value(s) of the independent variable. Testing the second value, 1.3054, in the solution vector **r** gives

```
>> polyval(c,r(2))
ans =
 -8.8818e-015
```

Within the accuracy of MATLAB, 15 decimal places, the value **r(2)** ≈ 1.3054 is very close to the actual root.

As an aside, the MATLAB command **poly** will construct a polynomial coefficient vector from a vector of roots. By default, the coefficient of the highest degree will be one. Consider $(2x-3)(x+5) = 2x^2 + 7x - 15$ and observe the results of the MATLAB commands

```
>> r = roots([2 7 -15])
r =
 -5.0000e+000
  1.5000e+000
>> poly(r)
ans =
  1.0000e+000  3.5000e+000 -7.5000e+000
```

As expected, **roots** returns the values $3/2$ and -5. Constructing a polynomial with the same roots using **poly** gives $x^2 + 3.5x - 7.5$. The roots of this polynomial are identical to the roots of $2x^2 + 7x - 15$.

Every student of calculus knows that derivatives of polynomials are easy to compute. The command **polyder** will operate on a coefficient vector to produce a new vector containing the coefficients of the derivative of the polynomial. For example, the leading term of the polynomial represented by $[\,5 \quad 8 \quad 0 \quad 11\,]$ is $5x^3$ with a derivative of $5 \cdot 3\,x^2$. Convince yourself that the following is correct.

```
>> polyder([5,-8,0,11])
ans =
    15   -16     0
```

fzero

MATLAB's root finding procedure, **fzero**, may be used to compute the zeros of non-polynomial functions. The usual syntax is **fzero('fct1', initial value)** where the function, $f(x)$, is entered in an M-file. As an alternative, enter the function directly as a string.

```
>> fzero('x^2-2',1)
Zero found in the interval: [0.54745, 1.4525].
ans =
  1.4142e+000
```

Keep in mind that roots of even multiplicity, wherein the graph of $f(x)$ does not actually cross the x-axis, will cause problems for **fzero**. Multiple roots will be considered in the next section.

As another example, the problem noted at the beginning of this chapter, $f(x) = e^{-x} - x$, may be solved with **fzero**. Visualize the graph of the decaying exponential function, $y = e^{-x}$, and the graph of the line, $y = x$. A reasonable guess for the intersection of these two graphs might be $x = 0.5$. The MATLAB results are

```
>> fzero('exp(-x)-x',.5)
Zero found in the interval: [0.42, 0.58].
ans =
  5.6714e-001
```

As an alternative, with $e^{-x} - x$, the active function in fct1.m

```
>> fzero('fct1',.5)
Zero found in the interval: [0.42, 0.58].
ans =
  5.6714e-001
```

4.5 Root Finding Difficulties

As noted earlier, the bisection method will converge slowly to the desired root within the accuracy of the machine implementing the algorithm. On the other hand, Newton's method may generate non convergent or slowly convergent data as a result of the nature of the function and the initial value.

To illustrate a slowly convergent sequence of Newton iterates, consider the numerical task of finding the root of sin(x) at zero. If Newton's method is started with $x_1$very close to 1.16556, the first iteration will give a numerical value of $x_2 \approx -1.16556$. With symmetry, we should expect $x_3 \approx 1.16556$, $x_4 \approx -1.16556$ and so forth. In other words, Newton's method is in a loop which may eventually converge, albeit very slowly, or may never converge. We have previously seen how quickly Newton's method converged

Using newton.m on $f(x) = \sin(x)$ with an initial value close to 1.16556 gives

```
>> r = newton(.000000001,'fct1','fct1p',1.165561185);
Error tolerance met at i = 17
      i                 x(i)              f(x(i))        x(i+1)-x(i)
  1.0000e+000  1.1656e+000  9.1901e-001  2.3311e+000
  2.0000e+000 -1.1656e+000 -9.1901e-001  2.3311e+000
  3.0000e+000  1.1656e+000  9.1901e-001  2.3311e+000
  4.0000e+000 -1.1656e+000 -9.1901e-001  2.3311e+000
  5.0000e+000  1.1656e+000  9.1901e-001  2.3311e+000
  6.0000e+000 -1.1656e+000 -9.1901e-001  2.3311e+000
  7.0000e+000  1.1656e+000  9.1901e-001  2.3311e+000
```

Although only the first seven values are shown, the procedure does converge eventually to meet the error tolerance in 17 iterations. Observe the alternating values of $x(i)$ in the output and the lack of quadratic convergence.

Figure 4.3 illustrates the predicament described in the data and shows how Newton's method may be trapped in a slowly convergent loop where the iterates change very little from step to step.

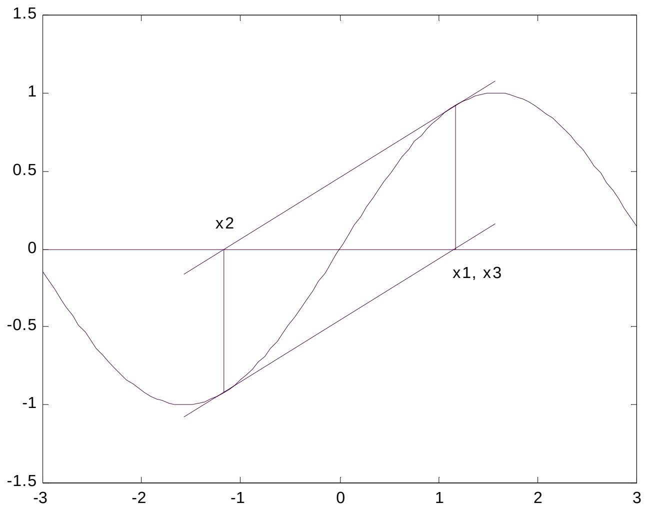

Figure 4.3 Newton's Method for sin$(x), x1 = a$

How did such an example occur? The problem is a poor choice of an initial guess resulting in a pair of parallel tangent lines around the symmetric function sin(x). To see where the initial guess $x_1$came from, we construct the tangent line to the graph of sin(x) at $x =$ a with $0 < a < \frac{\pi}{2}$ as follows:

$$y - \sin(a) = \cos(a)(x - a). \tag{4.18}$$

Since the graph of sin(x) is symmetric to the origin, we attempt to find the x-intercept of the tangent line (4.18) at the point $(-a, 0\,)$. In other words

$$0 - \sin(a) = \cos(a)(-a - a)$$

or $$\tan(a) = 2a. \tag{4.19}$$

fzero may be used to solve this nonlinear equation.

```
>> format long e
>> fzero('tan(x)-2*x',1)
Zero found in the interval: [0.77373, 1.2263].
ans =
   1.165561185207211e+000
```

This is the initial guess x_1 that resulted in the slowly convergent sequence of Newton iterates shown in Figure 4.3.

In this example we know the solution is zero; however, in a more complicated problem, an unfortunate choice of an initial value may produce unexpected results. The selection of an initial value other than $x_1 = 1.165561185$ gives dramatically different results as shown in the following data.

```
>> r = newton(.000000001,'fct1','fct1p',1);

Error tolerance met at i =  5
     i              x(i)           f(x(i))        x(i+1)-x(i)
  1.0000e+000  1.0000e+000  8.4147e-001  1.5574e+000
  2.0000e+000 -5.5741e-001 -5.2899e-001  6.2334e-001
  3.0000e+000  6.5936e-002  6.5889e-002  6.6032e-002
  4.0000e+000 -9.5722e-005 -9.5722e-005  9.5722e-005
  5.0000e+000  2.9236e-013  2.9236e-013  2.9236e-013
Root is approximately  0
```

Multiple Roots

Another source of root finding difficulties occurs in the case of multiple roots. We say that $f(x)$ has a root of multiplicity k at α if $f(x)$ has the form

$$f(x) = (x-\alpha)^k g(x)\ , \quad \text{where } g(\alpha) \neq 0 \text{ and } k \text{ is a positive integer.}$$

Functions with even multiplicity will not cross the x-axis. Without a sign change the bisection method and MATLAB's **fzero** will be unable to locate the root. Although Newton's algorithm will usually converge, the rate will be slow.

An example of multiplicity two will illustrate the difficulties. Consider the simple function, $f(x) = (x-3.2)^2 e^{-x}$. Since **fzero** attempts to find an interval with a sign change, values close to the root at 3.2 provide interesting results.

```
>> fzero('fct1', 5)
Exiting fzero: aborting search for an interval containing a sign change
   because NaN or Inf function value encountered during search
```

```
(Function value at -814.2 is Inf)
Check function or try again with a different starting value.
ans =
  NaN
```

An inappropriate starting value will also give unexpected results.

```
>> fzero('fct1', 100)
Zero found in the interval: [-624.0773, 824.0773].
ans =
  8.2408e+002
```

Clearly, 824.08 is not a root of $f(x)$ although MATLAB will confirm the result. Why?

```
>> fct1(ans)
ans =
    0
```

Both examples illustrate the need for good qualitative analysis when a multiple root is suspected.

The following data illustrates what may occur when Newton's method is applied to a function with a multiple root. Consider $f(x) = x^2\cos(x - \frac{\pi}{2})$. Clearly f has a multiple root at zero. Changing the maximum number of iterations in newton.m yields the following data.

```
>> r = newton(.00001,'fct1','fct1p',.5);
Error tolerance met at i = 25
     i              x(i)            f(x(i))        x(i+1)-x(i)
  1.0000e+000  5.0000e-001  1.1986e-001  1.7151e-001
  2.0000e+000  3.2849e-001  3.4811e-002  1.1083e-001
  3.0000e+000  2.1765e-001  1.0230e-002  7.2936e-002
         ⋮
  1.3000e+001  3.7565e-003  5.3011e-008  1.2522e-003
  1.4000e+001  2.5044e-003  1.5707e-008  8.3479e-004
  1.5000e+001  1.6696e-003  4.6539e-009  5.5652e-004
         ⋮
  2.3000e+001  6.5144e-005  2.7646e-013  2.1715e-005
  2.4000e+001  4.3429e-005  8.1913e-014  1.4476e-005
  2.5000e+001  2.8953e-005  2.4270e-014  9.6510e-006
Root is approximately 1.930194e-005
```

Based on the data, it appears that both **x(i)** and **f(x(i))** are converging to zero; however, the normal quadratic rate of convergence is missing. Slow convergence or lack of quadratic convergence in Newton's method is a warning sign for the possibility of a multiple root.

It is possible to show that the rate of change of x_{n+1} with respect to x_n, in other words the derivative, $\frac{dx_{n+1}}{dx_n}$, is related to the multiplicity of the root. A useful relationship may be derived from Newton's method (4.10) and is given by

$$\frac{dx_{n+1}}{dx_n} \approx 1 - 1/k = \frac{k-1}{k}. \tag{4.20}$$

It is important to keep in mind that the approximation in (4.20) assumes that Newton's method is converging. The derivative on the right hand side may be approximated by a difference quotient using three consecutive Newton iterates as follows

$$\frac{dx_{n+1}}{dx_n} \approx \frac{x_{n+1} - x_n}{x_n - x_{n-1}} \approx \frac{k-1}{k}. \tag{4.21}$$

Using x_{13}, x_{14}, and x_{15} from the example data, the ratio in (4.21) becomes

$$\frac{x_{15} - x_{14}}{x_{14} - x_{13}} \approx \frac{1.6696 - 2.5044}{2.5044 - 3.7565} \approx 0.67.$$

With $k = 3$, the ratio $\frac{k-1}{k}$ in (4.21) is $\frac{2}{3}$; thus, $f(x) = x^2\cos(x - \frac{\pi}{2})$ has a root of multiplicity three at zero. Two of the roots come from x^2 and the third from $\cos(-\frac{\pi}{2})$.

If the multiplicity is unknown, the relationship (4.21) may be used to compute an estimate of the multiplicity. With an estimate of k, we may differentiate the function $k-1$ times to obtain a new function with a simple root. If k is small, perhaps 2 or 3, this may be a viable approach. As an example, recall $f(x) = (x-3.2)^2e^{-x}$ with $k = 2$. The first derivative is $f'(x) = 2(x-3.2)e^{-x} - (x-3.2)^2e^{-x}$. Factoring gives $f'(x) = (x-3.2)e^{-x}(5.2 - x)$. In other words, $f'(x)$ has a simple zero at 3.2.

With the multiplicity, k, known the so-called modified Newton's method, will demonstrate quadratic convergence. The modified method includes the factor of k as shown in equation (4.22).

$$x_{n+1} = x_n - k\frac{f(x_n)}{f'(x_n)} \quad \text{for } n = 1,\ 2,\ 3,\ \ldots. \tag{4.22}$$

4.4 Problems

4-1. Given $f(x) = e^{-x} - x$ on the interval $[\,0,\ 1\,]$.
 a. Use the bisection method to compute c_4 by hand. What is the error bound for c_4?
 b. Determine the number of bisections needed to result in an error of .00001.
 c. Verify your solution of part b) with bisect.m .

4-2. Repeat Problem 4-1 with $f(x) = \tan(x) - \sqrt{x}$ on the interval $[\,0.5,\ 1\,]$. Use radians.

4-3. Consider $f(x) = 1 + x^2 - \tan(x)$.
 a. Use **ezplot** to graph $1 + x^2$ and $\tan(x)$ in the same figure window. See >> **help hold**.
 b. Select an interval containing the first positive root and use the bisection method to compute c_4 by hand.
 c. For your choice of an interval what is the error bound for c_4?
 d. Determine the number of bisections needed to result in an error of 0.00001.

e. Verify your solution of part d with bisect.m .

4-4. Repeat Problem 4-3 seeking the negative root closest to zero.

4-5. Show that the formula for the x-intercept of a line joining the points $(a, f(a))$ and $(b, f(b))$ is given by

$$c = \frac{af(b)-bf(a)}{f(b)-f(a)}.$$

4-6. The result given in Problem 4-5 is used in the method of false position.
a. Use false position to compute c_4 for $f(x) = x^2 - 2$ on the interval $[0, 2]$.
b. Modify bisect.m, call it falsep.m, to implement the method of false position. Delete the error bound term $(b-a)/2$ in the output display.
c. Using your M-file, falsep.m, determine the root of $f(x) = e^{-x} - x$ on the interval $[0, 1]$. Compute ten iterations.

4-7. Use Newton's method to estimate $\sqrt[5]{31}$ by finding the root of $x^5 - 31 = 0$. Compute x_5 using five decimal places beginning with $x_1 = 2.5$ and hand computations. Compare the estimated error in x_4, $x_5 - x_4$, with the actual error in x_4, $\sqrt[5]{31} - x_4$.

4-8. Repeat Problem 4-7 using the symbolic capability of MATLAB.

4-9. Consider $f(x) = 1 + x^2 - \tan(x)$. See Problem 4-3.
a. Use Newton's method to approximate the first positive root. Begin with $x_1 = 0.1$ and compute x_4 using hand computations. Estimate the error in x_4.
b. Repeat part a with $x_1 = 1$.
c. Use newton.m to determine how many iterations are needed to approximate the root with an approximate error of 0.0001 for the initial values of parts a and b.

4-10. Use **ezplot** to estimate the positive root of $2\sin(x) - x = 0$. You may wish to use MATLAB's **grid** to fine tune your initial estimate. Use newton.m with an error tolerance of 0.000005 to compute the root.

4-11. Consider $f(x) = 1 + x^2 - \tan(x)$. See Problem 4-3.
a. Use the secant method to approximate the first positive root with $x_1 = 0$ and $x_2 = 1.5$. Iterate three times to compute x_5.
b. Repeat part a with $x_1 = 1$ and $x_2 = 1.5$. Iterate three times to compute x_5.

4-12. Modify newton.m to implement the secant method. Test your M-file on Problem 4-11.

4-13. Derive Newton's method using the first two terms of a Taylor approximation centered at the value x_n.

4-14. One half of the thickness of an airfoil is given by the equation

$t = 2.969\sqrt{x} - 1.26x - 3.516x^2 + 2.84x^3 - 1.015x^4, [0,\ 1].$

Use Newton's method with an error tolerance of 0.0001, to locate the value of x where the thickness is a maximum. Hint: To find the maximum value ...

4-15. Recall the equation for the floating sphere in equation (4.2).

$$\mathrm{H}^3 - 3\mathrm{H}^2\mathrm{R} + 4\mathrm{R}^3\delta/62.4 = 0, \text{ where } 0 < \mathrm{H} < 2\mathrm{R}.$$

Use **roots** and then **fzero** to determine H for a sphere of radius 5 ft given each of the densities $\delta = 20$, 45 and 70 lb/ft^3. Are your answers reasonable? Explain.

4-16. In an effort to determine the optimum damping ratio of a spring-mass-damper system designed to minimize the transmitted force when an impact is applied to the mass the following equation must be solved for ζ

$$\cos[\,4\,\zeta\,\sqrt{(1-\ \zeta^{\,2})}\ ] = -1 + 8\zeta^2 - 8\zeta^4$$

Use a root finding method to compute the solution to this problem.
To read more about this problem see the article by Peters in SIAM Review, V.39, No. 1, pp 119-122, March 1997.

4-17. Use Newton's algorithm to estimate the multiplicity of the root at zero for the function $f(x) = x\,\sin(x^2)\tan(x)$.

4-18. Using the results of the previous problem, verify quadratic convergence using the modified Newton algorithm, (4.22), on $f(x) = x\,\sin(x^2)\tan(x)$.

4-19. Each of the following functions has a root or roots with multiplicity greater than one. Determine the multiplicity in each case. Explain your reasoning. Finally, use one of the methods of this chapter to approximate the root or roots.
a. $f(x) = e^x - x - 1$
b. $f(x) = x^3 + 3.3x^2 + 3.63x + 1.331$
c. $f(x) = x^6 - 6x^4 + 12x^2 - 8$
d. $f(x) = \cos(x + \sqrt{2}) + x(x/2 + \sqrt{2})$

4-20. Without thinking Newton's algorithm might be used on $f(x) = x^2 + 1$. Find an initial value which will result in an alternating sequence of values.

4-21. Find a Taylor expansion of $\cos(x - \frac{\pi}{2})$ centered at $a = 0$. Use your result to show that $f(x) = x^2\cos(x - \frac{\pi}{2})$ has a root of multiplicity three at zero.