Chapter 3

Solution of nonlinear equations

3.1 Introduction and types of methods
3.2 Bracketing methods: Bisection and Linear interpolation; Convergence analysis.
3.3 Open Methods and their Convergence analysis: Fixed Point, Newton-Raphson, Secant, Muller, Steffensen methods, Aitken extrapolation
3.4 Complex roots and multiple roots: Bairstow's method, modified open methods
3.5 Design of methods of arbitrary order
3.6 Introduction to system of non-linear equations
3.7 Solution Methods: Fixed Point, Newton Raphson, and Secant methods
3.8 Summary

3.1 Introduction

Quite often we come across problems involving solution of an equation of the form f(x)=0. For example, continuing with our *Batman* problem, the time when he sees the wheel is given by the equation $f(t) = \frac{1}{2}gt^2 - 666 = 0$. For this case, obtaining the value of t is a straightforward mathematical operation. If we further complicate the problem by specifying that *Joker* threw the wheel with an initial downward velocity of *u* m/s, the equation gets modified to $f(t) = ut + \frac{1}{2}gt^2 - 666 = 0$, which can be solved if one knows the relevant formula for solution of a quadratic equation. In both cases, the function *f* is a nonlinear function of *t* because of the presence of t^2 term. A solution is, however, readily obtained. We may not be so fortunate in most of the practical problems involving nonlinear functions. For example, we may be interested in finding out the time at which the distance travelled by the wheel (in m) is equal to the square root of the time (in seconds). The problem is formulated as $f(t) = ut + \frac{1}{2}gt^2 - \sqrt{t} = 0$ which may be manipulated to obtain a cubic equation, which can be solved analytically. However, the solution process is a little cumbersome and one may decide to use a numerical method to obtain the value of *t*. On the other hand, it is possible that an analytical solution is not obtainable (e.g., if $h=t^{1/4}$) and one has to use numerical methods. There are various other problems arising in different fields of engineering which require the solution of nonlinear equations. In this chapter, we would discuss some of the numerical methods which could be used for this purpose. To start with, we would assume that the nonlinear function, *f*, is a function of only one variable, *x*. In a latter chapter we would look at extension of some of the "single variable" techniques to functions of two or more variables giving rise to a system of nonlinear equations. Also, for most of the discussion we would assume that only real roots are desired. A brief description of methods which could be used for finding complex roots is presented in section 3.4.

As discussed in Chapter 1, any method which we use to solve *f(x)=0* (in other words, finding the *zeroes* or *roots* of the function) has to be judged on two important criteria: accuracy (how close is the computed root to the actual root) and efficiency (how much computing time it takes to obtain the root). A graphical method, in which the root is obtained by plotting the function and noting the point of intersection with the x-axis, may be very efficient in providing a "rough" estimate of the root but is not very accurate. Of course, the accuracy may be improved by successive "zoom-ins" in the vicinity of the root but then the efficiency suffers. In any case, since this is a book on Numerical Methods, we would simply state that graphical methods have become obsolete with the advent of high-speed computing! We must confess, however, that many a times a graph of the function provides tremendous insight into the behaviour of various numerical methods.

One may borrow the idea from the graphical method and devise a numerical method in which the function is evaluated at various points until there is a change in the sign of the function value. Since most practical problems involve *well-behaved* (which, in the present case, means *continuous* or *piecewise continuous*) functions, it may be safe to assume that a root (or to be more precise, *at least* one root) lies between the two points at which the function has opposite signs[1]. In other words, we have *bracketed* the root. The only thing which is left to do now is to shrink this bracket to the desired accuracy, which can be done in a number of ways thus resulting in various ***bracketing*** methods. Typically an incremental search technique is used to bracket the root and the choice of increment is purely problem-specific. A large increment may not be able to bracket a root (e.g., when there are two roots close to each other) and a small increment will increase the computational time of bracketing but will generally result in smaller computational effort in refining (because the initial bracket is smaller). In the following discussion on the bracketing methods, we will assume that the root has already been bracketed and will concern ourselves only with the refinement of the estimate of the root.

Example 3.1: Bracket the roots of the following equations, given that one of the roots is close to -1 and the other two close to 1.

(a) $x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$

(b) $x^3 - 1.25020000x^2 - 1.56249999x + 1.95343750 = 0$

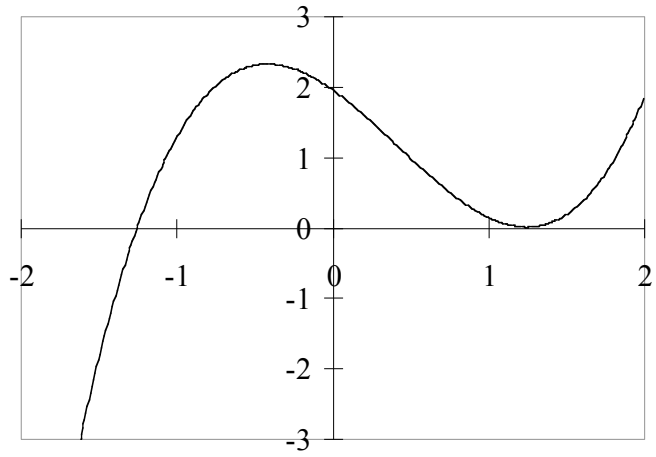
Solution: The following table shows the values of the functions at increments of 1, 0.1, and 0.01. The negative root is easily bracketed (between -2 and -1) for both the equations. With an increment of 1, the function values at x=1 are small but there is no change of sign. Even with an increment of 0.1, we see that the function values decrease till x=1.2 and starts increasing after that but there is again no change of sign. With an increment of 0.01, the two roots of the first function are bracketed, one between 1.24 and 1.25 and the other between 1.25 and 1.26. However, the roots of the second function are still not bracketed.

x	f_a	f_b	x	f_a	f_b	x	f_a	f_b
-2	-7.9218562	-7.92236252	1	0.14056880	0.14073751	1.20	0.00606380	0.00614951
-1	1.2656188	1.26573749	1.1	0.05281630	0.05294551	1.21	0.00387455	0.00395569
0	1.9530938	1.95343750	1.2	0.00606380	0.00614951	1.22	0.00216130	0.00223783

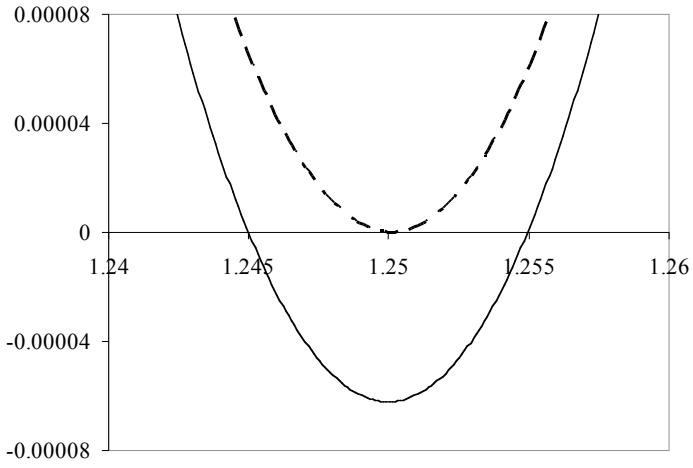
[1] Note that roots may exist between two points where the function has the same sign!

1	0.1405688	0.14073751	1.3	0.00631130	0.00634951	1.23	0.00093005	0.00100193
2	1.8280438	1.82763752	1.4	0.05955880	0.05954551	1.24	0.00018680	0.00025399
						1.25	-0.00006245	0.00000001
						1.26	0.00018830	0.00024599

The plots of the two functions are virtually identical (note that the functions have nearly equal coefficients) and show the presence of a double root near 1.25 (Fig. 3.1a). Although it is not apparent from the figure, there are two distinct roots at 1.245 and 1.255 for the first function and a double root at 1.25003 for the second function (Fig. 3.1b, which zooms in near the root). Thus a very small increment is needed to bracket the roots if there are two roots very close to each other and it would not be possible to bracket a double root (or any root of even multiplicity[1]).



(a)



(b)

[1] In general, we say that *f(x)* has a root, ξ, of multiplicity *m*, if there exists a nonzero real number, *c*, such that $\lim_{x\to\xi}\frac{|f(x)|}{|x-\xi|^m}=c$.

Figure 3.1 Plot of the functions of Example 3.1 (a) General trend (b) Near the root : Solid line - f_a; Dashed line - f_b.

Some times one may not want to spend the effort in bracketing the root. For example, if from the physics of the problem, it is obvious that the root is near a particular value, it may be desirable to take that value as our initial estimate of the root and then try to improve the estimate using numerical methods. These methods could be called ***open*** methods since we have not enclosed the root and, as we will see later, the successive approximations of the root may not lie within a closed interval.

3.2 Bracketing Methods

Assuming that we have bracketed the root within the interval $[x_l,x_u]$ defined by the lower and upper limits, x_l and x_u, a simple way of shrinking the bracket would be to evaluate f(x) at the midpoint of this interval (x_m), and then comparing its sign with $f(x_l)$ and $f(x_u)$ (for ease of notation, we denote these function values as f_m, f_l, and f_u). The bracket is then reduced by half by using x_m as x_l (if f_m and f_l have same sign) or x_u (if f_m and f_u have same sign). However, one obvious drawback of this method is that the magnitude of f(x) is given no consideration in shrinking the bracket. One would expect (and it is generally, *but not always*, true, see Fig. 3.2) that the root will be closer to the end where the function magnitude is smaller. Therefore a method may be devised which gives proper weight to the function values f_l and f_u in choosing the bracket at the next iteration. These ideas have been formalised in the subsequent subsections.

Figure 3.2 The root is closer to the end with larger magnitude of function

Bisection Method (Interval halving)

If one were given a bracket $[x_l,x_u]$ which contains the root (indicated by $f_l \cdot f_u < 0$) and asked to estimate the location of the root *without any regard to the function value at either ends* (which may work out to be a better option *in some cases*, see problem 1, Exercise 3.2), the logical choice would be the midpoint of the bracket, $x_m \left(= \frac{x_l + x_u}{2} \right)$. Now if the function value at x_m is zero, we have found the root. If not, we have narrowed down the bracket to $[x_l,x_m]$ or $[x_m,x_u]$ depending on whether $f_l \cdot f_m < 0$ or $f_m \cdot f_u < 0$. Thus at each step we reduce the bracketing *interval* to *half* by this *bisection*. The process is repeated till the desired accuracy is

achieved. In absence of the true value (ξ) of the root, the estimation of accuracy may be based on the approximate error

$$\varepsilon = \text{Current approximation} - \text{Previous approximation}$$

which would be equal to half of the bracket length at any iteration (the current approximation would be the midpoint of the bracket length and the previous approximation would be either the lower or the upper end of this bracket). Since the root lies *within* the bracket, the magnitude of the approximate error will always be greater than that of the true error. This is a desirable property since the iterations are stopped when the approximate error falls below a pre-decided tolerance thus ensuring that the true error is even smaller than the desired tolerance. It should be noted that we are talking in terms of the absolute error and not relative error, which, as mentioned in Chapter 1, should be preferred. However, in situations where we do have a general idea about the magnitude of the root beforehand, the absolute error criterion will be valuable.

If we denote the length of the initial bracket as $\Delta x^{(0)}$, the first estimate of the root[1] will be at its midpoint and, no matter which half is picked as the next bracket, the length of the new bracket would be $\frac{\Delta x^{(0)}}{2}$ and the magnitude of the approximate error for the second estimate would be $\frac{\Delta x^{(0)}}{4}$. Extending this argument, the magnitude of the approximate error for the n^{th} estimate of the root would be $\frac{\Delta x^{(0)}}{2^n}$ (this means that we can estimate *a priori* the number of iterations needed to reduce the approximate error to a specified tolerance). By using the nested interval theorem (Theorem 3.1), it is seen that the bracket will *ultimately* reduce to a single point, which will be the desired root. Also, the error at any iteration is equal to half of the error at the previous iteration, indicating a *linearly convergent* behaviour as described in the following paragraph which introduces the concept of *order of convergence* of an iterative method.

> If an iterative sequence $x^{(0)}$, $x^{(1)}$, $x^{(2)}$ … converges to the root ξ, the true error at iteration i is $e^{(i)} = \xi - x^{(i)}$ and $\lim_{i\to\infty} \frac{\left|e^{(i+1)}\right|}{\left|e^{(i)}\right|^p} = C$, then p is the **order of convergence** and C the **asymptotic error constant**. The convergence is called linear if p=1, quadratic if p=2, and cubic if p=3 (as we will see, p does not have to be an integer).

For the bisection method, if we use the approximate error (which is an upper bound of the true error and approaches the true error as the iterations converge to the root), we get p=1 (hence linearly convergent) and C= $\frac{1}{2}$ (indicating that the error is reduced by half in every iteration). Note that for linearly convergent methods, C has to be less than 1 to ensure that the method

[1] Throughout this book, we use the notation $x^{(i)}$ to represent the value of x at iteration i. While x^i or x_i would have been simpler, it may cause ambiguity at some places.

converges to the root. For superlinearly convergent methods (p>1), however, there is no such restriction. Also, it should be noted that, if the iterations are converging *quickly* towards the root, $x^{(i+I)}$ would be much closer to the root than $x^{(i)}$ and the approximate error at any iteration, $x^{(i+1)} - x^{(i)}$, is an indicator of the *true error* at the *previous iteration,* $\xi - x^{(i)}$, resulting in a conservative error estimate.

Theorem 3.1: **Nested Interval Theorem**

If $I_1 \supset I_2 \supset \ldots \supset I_n \supset \ldots$ is a sequence of nested, closed, bounded, nonempty intervals, then $\bigcap_{n=1}^{\infty} I_n$ is nonempty. Also, if length of the intervals approaches zero, then the intersection consists of a single point. The theorem is proved by writing the nth interval $I_n = [l_n, u_n]$ in terms of its lower and upper ends. Since the intervals are nested, $l_1 \le l_2 \le \ldots \le l_n \le u_n \ldots \le u_2 \le u_1$. Therefore, the l forms an increasing sequence bounded from above and u forms a decreasing sequence bounded from below. Since every monotone bounded sequence converges, l_n converges to, say, L, and u_n converges to, say, U, such that $l_n \le L$ and $u_n \ge U \ \forall n$. If $\lim_{n\to\infty} (u_n - l_n) = 0$, we have L=U and since $l_n \le u_n \ \forall n$, $L \in \bigcap_{n=1}^{\infty} I_n$, proving that the intersection consists of a single point.

Linear Interpolation Method (*Regula Falsi* or *False Position*)

As discussed in the previous section, the bisection method does not account for the magnitude of the function at the end points of the bracketing interval when estimating a *better* location of the root. A likely improvement, therefore, would be to estimate the location of the root in such a way that it is closer to the end which has a smaller function magnitude. The most common method of doing this is to use a **linear interpolation** between the two end points to estimate where the function will be zero (which indicates a **false position** of the root). Of course, one could use the function value at, say, the mid-point of the interval and perform a quadratic interpolation to locate the root. The additional function evaluation would increase the computational effort but may result in a faster convergence. In this section, we describe and analyse the Linear Interpolation method.

Assuming a linear variation of the function within the bracketing interval (x_l,x_u), an estimate of the location of the root could be made as $\tilde{x}$ given by

$$\tilde{x} = x_l - f_l \frac{x_u - x_l}{f_u - f_l} \tag{3.1}$$

As done earlier for the bisection method, $\tilde{x}$ will then replace x_l or x_u for the next iteration, depending on the value of $f(\tilde{x})$. If *f(x)* is uniformly concave or convex over the interval (x_l,x_u), it can be seen (Fig. 3.3) that one end of the interval remains fixed. If we take this fixed point as $x^{(0)}$, we have

$$x^{(i+1)} = x^{(0)} - f_0 \frac{x^{(i)} - x^{(0)}}{f_i - f_0} \tag{3.2}$$

in which f_0 and f_i represent the function values at $x^{(0)}$ and $x^{(i)}$, respectively.

Figure 3.3 Iterative sequence for linear interpolation for a uniformly concave function

To relate the error at iteration *i+1* with that at the i^{th} iteration, we introduce here the concept of errors associated with interpolation. A very brief description is given here since this topic will be dealt with in detail in Chapter 4.

Interpolation refers to approximating a given function (either continuous or in discrete form) by another function (generally polynomial) in such a way that the approximating function matches the given function at *some* selected points (assumed to be distinct). Denoting the given function by *f(x)*, the approximating polynomial by $\tilde{f}(x)$ and the grid of *match points* as $x_i (i = 0,1,2,...m)$, it is clear that $\tilde{f}(x)$ will be an m^{th}-degree polynomial, the coefficients of which may be obtained by the *m+1* equations representing the equality of *f(x)* and $\tilde{f}(x)$ at the grid points. However, at all other points, there would be some *remainder R(x)* (unless *f(x)* happens to be a polynomial of degree *m* or lower). The remainder at any point, *x*, may be written as (considering that the remainder has to be zero at all the grid points),

$$R(x) = f(x) - \tilde{f}(x) = A(x)(x - x_0)(x - x_1)(x - x_2)...(x - x_m) \tag{3.3}$$

where *A(x)* is a yet to be determined function of *x*. To obtain the expression for *A(x)*, we now define a function

$$F(\chi) = f(\chi) - \tilde{f}(\chi) - A(x)(\chi - x_0)(\chi - x_1)(\chi - x_2)...(\chi - x_m) \tag{3.4}$$

which is, clearly, zero when χ is a grid point since *f(x)* and $\tilde{f}(x)$ are matched at these points (note that the function definition in the above equation uses the function *A* at the point *x* and NOT χ. In other words, it could be treated as a constant with respect to χ). The important point to note is that eq. (3.3) indicates that *F(χ)* will also be zero when χ=*x*. Therefore, as the point χ moves over the interval containing $(x,x_0,x_1,x_2,...x_m)$ [1], *F(χ)* will become zero at (*m*+2) points which include the (*m*+1) grid points and the point *x*, at which we want to obtain the residual, *R(x)*. An application of Rolle's theorem[2] thus indicates that there are at least (*m*+1) points in the interval containing $(x,x_0,x_1,x_2,...x_m)$ at which the first derivative of the function *F* with respect to χ is zero. A further application of this theorem to the derivatives indicates that there are at least *m* points at which the second derivative of *F* is zero. Repeated applications of the theorem, therefore, lead to the fact that there must be a point $\hat{x} \in (x,x_0,x_1,x_2,...x_m)$ such

[1] Written compactly as $x^* \in (x,x_0,x_1,x_2,...x_m)$

[2] If f is continuous on [a, b] and differentiable on (a, b), and f(a) = f(b), then there exists a number c in (a, b) such that f'(c) = 0

that the $(m+1)^{th}$ derivative, $F^{m+1}(\hat{x}) = 0$. If we now assume that the given function, *f(x)*, has continuous derivative of at least $(m+1)^{th}$ order and using the fact that the $(m+1)^{th}$ derivative of $\tilde{f}(\chi)$ w.r.t. χ is zero and that of $(\chi - x_0)(\chi - x_1)(\chi - x_2)...(\chi - x_m)$ is (m+1)!, we get, from the definition of $F(\chi)$,

$$F^{m+1}(\chi) = f^{m+1}(\chi) - \tilde{f}^{m+1}(\chi) - A(x)\frac{d^{m+1}}{d(\chi)^{m+1}}(\chi - x_0)(\chi - x_1)(\chi - x_2)...(\chi - x_m) \tag{3.5}$$

$$\Rightarrow A(x) = \frac{f^{m+1}(\hat{x})}{(m+1)!} \quad \text{(on evaluating at } \chi = \hat{x})$$

resulting in

$$R(x) = \frac{f^{m+1}(\hat{x})}{(m+1)!}(x - x_0)(x - x_1)(x - x_2)...(x - x_m) \tag{3.6}$$

The location of the point $\hat{x}$, of course, depends on *x*.

For linear interpolation between points $x^{(0)}$ and $x^{(i)}$, we may, therefore, write

$$f(x) = f_0 + (x - x^{(0)})\frac{f_i - f_0}{x_i - x^{(0)}} + (x - x^{(0)})(x - x^{(i)})\frac{f''(\hat{x}_1)}{2} \tag{3.7}$$

where the first two terms on the right hand side represent the linear interpolating polynomial, $\hat{f}(x)$, the last term represents the remainder, and $x, \hat{x}_1 \in (x^{(0)}, x^{(i)})$. Applying this equation at the root, $x=\xi$ [such that $f(\xi) = 0$], substituting the value of f_0 from eq. (3.2) (from which $f_0 + (\xi - x^{(0)})\frac{f_i - f_0}{x^{(i)} - x^{(0)}} = (\xi - x^{(i+1)})\frac{f_i - f_0}{x^{(i)} - x^{(0)}}$), and using the derivative mean value theorem [implying $\frac{f_i - f_0}{x^{(i)} - x^{(0)}} = f'(\hat{x}_2)$ in which $\hat{x}_2 \in (x^{(0)}, x^{(i)})$], we get

$$0 = (\xi - x^{(i+1)})f'(\hat{x}_2) + (\xi - x^{(0)})(\xi - x^{(i)})\frac{f''(\hat{x}_1)}{2} \tag{3.8}$$

Thus

$$e^{(i+1)} = -\frac{f''(\hat{x}_1)}{2f'(\hat{x}_2)}e^{(0)}e^{(i)} \tag{3.9}$$

Assuming that the iterations converge to the root, $\lim_{i\to\infty}\frac{|e^{(i+1)}|}{|e^{(i)}|} = \left|\frac{f''(\hat{x}_3)e^{(0)}}{2f'(\hat{x}_4)}\right|$, where $\hat{x}_3, \hat{x}_4 \in (x^{(0)}, \xi)$. This indicates that, similar to the bisection method, the method of false position is also linearly convergent. Since the root is always bracketed, the method will converge. However, the asymptotic error constant, which had a constant value of ½ for the bisection method, is now dependent on the slope and curvature of the function (see problem 2, Exercise 3.2).

Example 3.2: Find a root of the following equation, to an accuracy of 1%, using the bisection and linear interpolation methods, given that the root has been bracketed in the interval (–2,–1).

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Solution:

The following table shows the steps of computation for the bisection method.

Iteration no.	x_l	x_u	f_l	f_u	$x_m = (x_l+x_u)/2$	ε_r (%)	f_m
1	-2	-1	-7.92186	1.265619	-1.5		-1.89062
2	-1.5	-1	-1.89062	1.265619	-1.25	20	5E-08
3	-1.5	-1.25	-1.89062	5E-08	-1.375	9.090909	-0.86132
4	-1.375	-1.25	-0.86132	5E-08	-1.3125	4.761905	-0.4104
5	-1.3125	-1.25	-0.4104	5E-08	-1.28125	2.439024	-0.20022
6	-1.28125	-1.25	-0.20022	5E-08	-1.26563	1.234568	-0.09888
7	-1.26563	-1.25	-0.09888	5E-08	**-1.25781**	0.621118	Not needed

Note that the function value at the midpoint, x_m, is nearly zero at the second iteration. It may sometimes be beneficial to put a stopping criterion based on the function value, e.g., we stop the iteration when the magnitude of the function value is less than a specified limit. Here we have stopped iterating when the approximate error becomes less than 1%. The reduction in the approximate error by a factor of 2 is readily seen in the table[1].

The linear interpolation method converges faster since the function is almost linear in the bracket (see Fig. 3.1).

Iteration no.	x_l	x_u	f_l	f_u	$\tilde{x}$ (Eq. 3.1)	ε_r (%)	$\tilde{f}$
1	-2	-1	-7.92186	1.265619	-1.13775		0.639949
2	-2	-1.13775	-7.92186	0.639949	-1.2022	5.360842	0.287417
3	-2	-1.2022	-7.92186	0.287417	-1.23013	2.270632	0.122192
4	-2	-1.23013	-7.92186	0.122192	**-1.24183**	0.941715	-

Note that one end of the bracket (in this case, the lower end) remains fixed since the function is convex upward in the entire bracket (the second derivative is negative). Thus, in Eq. (3.9), $e^{(0)}$ is equal to 0.75. The value of $\left(-f''/2f'\right)$ varies from about 0.5 to 0.8 over the interval $(-2,-1.25)$, indicating that the absolute true error at any iteration should be about 0.35 to 0.6 times the true error at the previous iteration. The table shows a factor of about 0.4 for the relative approximate error.

Other Possible Methods

Refinement of the bracket was done by interval halving and linear interpolation in the two methods described above. There are numerous other possibilities which could be used for this purpose. For example, instead of halving the interval, we may divide it into the golden ratio[2].

[1] Since we have computed the *relative* error, it is not exactly half of that at the previous iteration.

[2] The golden ratio is defined as the ratio in which a line is divided so that the ratio of the smaller part to the bigger part is equal to the ratio of the bigger part to the entire line. Its value is equal to 1.618 (=.618/.382=1/.618). It is also called the *divine proportion* due to its frequent occurrence in nature. For refining the bracket to find the

Or a quadratic interpolation[1] using the function values at the end points along with an additional function value at the mid-point (or, for that matter, any other point in the bracket!) could be used to estimate the location of the root (Problems 3,4 of Exercise 3.2). We may, in the linear interpolation method, devise some strategy to accelerate the rate of convergence (e.g., if one end of the bracket remains fixed for a few iterations, we use some fraction, usually ½, of the function value at that end for the interpolation at the next iteration, see Problem 5, Exercise 3.2). Or, we may manipulate the function values at the ends and at the midpoint in such a way that they fall on a straight line and then use linear interpolation (Ridder's method, see Problem 6, Exercise 3.2). Another alternative is to combine the linearly convergent bisection method and superlinearly convergent quadratic interpolation (Brent's method, see Problem 7, Exercise 3.2). These modifications generally lead to faster convergence but require either more function evaluations or more record-keeping and comparisons, and the improvement in efficiency may not be significant.

Remarks:

- The bracketing methods have guaranteed convergence with linear convergence rate
- Since one end of the bracket is common between two successive iterations, the function value at this point may be stored and need not be computed again. This will reduce the total number of function evaluation making the algorithm more efficient
- If the bracket contains more than one root, the bracketing method will find only one of them, say ξ. A common strategy to find other roots is to *factor out* $(x-\xi)$ from *f(x)* and find the roots of the function $f_1(x) = \dfrac{f(x)}{x-\xi} = 0$. This strategy will not work for bracketing methods, since the bracket obtained for *f(x)*=0 can no longer be used as a bracket for $f_1(x)=0$, as $f_1(x)$ will have the same sign at both ends. To overcome this difficulty, *if we know that there are more than one roots in the bracket*, we may try to locate another root by expanding the initial bracket and then factoring out two roots at a time (Problem 8, Exercise 3.2). However, the open methods described in the next section would be much better at locating all roots.
- If the bracket contains a multiple root (of odd multiplicity), bisection will have no difficulty in obtaining the root. However, the linear interpolation will show a very slow convergence (Problem 9, Exercise 3.2).
- Because of linear convergence properties, typically the bracketing methods are used only as *starting* methods. As we approach the root and are relatively confident that the iterations will not diverge, we would like to use a method which has a faster convergence even if it does not guarantee convergence (as the bracketing methods do). Such *open methods* are described in the next section.

Exercise 3.2

(Note: for all root-finding problems, it would be a good idea to plot the function to get a feel about its nature).

---

root, bisection is the optimal strategy. However, for finding a minimum of the function within a bracket, the golden ratio works out better.

[1] We fit a parabola through the three function values to get a quadratic equation, which is solved to get the value of x at which the parabola crosses the x-axis.

1. Find a root of $f(x) = (2x)^{10} - 1.2689 = 0$ to an accuracy of 0.001 given that the root has been bracketed in [0,1]. Use the bisection method and the linear interpolation method and comment on their relative performance.
2. Solve the equation $x^7 + x^4 - 0.01 = 0$ using the linear interpolation method.
3. Solve the equation $x^3 - 3.6x^2 + 4.32x - 1.728 = 0$ using quadratic interpolation with three points located at the ends of the bracket and its midpoint.
4. Solve the above equation using quadratic interpolation with the third point located at a distance of one-fourth of the interval from the end having smaller function value.
5. Solve the equation $x^7 + x^4 - 0.01 = 0$ with modified linear interpolation method, reducing the function value by half if one end of the interval remains same for more than two iterations.
6. Solve the above equation using ***linear*** interpolation with midpoint using the function values multiplied by an exponential function (Ridder's method, see Box 3.1).
7. Solve the equation $x^3 - 3.6x^2 + 4.32x - 1.728 = 0$ using a quadratic inverse interpolation with three points[1] with the provision that if the estimated root falls outside the bracket, we switch to the bisection method (Brent's method).
8. The equation $x^5 - 4.3x^4 + 10.04x^3 - 19.972x^2 + 24.16x - 11.088 = 0$ has three roots between 1 and 2. Use linear interpolation with starting bracket of (1,2) to get a root. Change the bracket to (0,2) and apply linear interpolation method to get a (possibly different) root. Factor out these two roots to deflate the given polynomial and find the third root.
9. Solve $x^3 - 4.35x^2 + 6.3075x - 3.048625 = 0$ using the linear interpolation method. Note that there is a triple root at x=1.45.

Box 3.1: **Ridder's method**

In this method, we modify the given function in such a way that the resulting function values at the end points of the interval and at the mid-point lie in the same straight line. The modification is done through an exponential function of the form $e^{c\frac{x-x_l}{x_u-x_l}}$ and the coefficient c is chosen so as to satisfy the linearity condition in the interval (x_l,x_u) using the two end points and the midpoint (x_m). Or, equivalently, we look for a positive number, α, such that f_l/α, f_m, and αf_u are collinear. It is easy to show that this number is given by

$$\alpha = \frac{f_m + \text{sgn}(f_u)\sqrt{f_m^2 - f_l f_u}}{f_u}$$

[1] For the first iteration, we may take the three points as the two ends of the bracket and the estimated root obtained by linear interpolation. Using these 3 function values and corresponding x values, a quadratic inverse interpolation is written as $x=c_0+c_1f+c_2f^2$ and the value of c_0 provides the estimate of the root. For subsequent iterations, the three points are chosen such that the root is bracketed and the most recent estimate is included.

and the new estimate of root, obtained by linear interpolation between f_m and αf_u, is $\hat{x} = x_m - (x_u - x_m)\dfrac{\text{sgn}(f_u) f_m}{\sqrt{f_m^2 - f_l f_u}}$.

3.3 Open Methods

In the bracketing methods, to start the iterations we need to bracket the root, which may be time consuming. Once a bracket has been identified, the bracketing methods are guaranteed to converge to a root for continuous functions, but the rate of convergence is relatively slow. In this section we will look at some of the so called open methods, which need one or more (not necessarily bracketing the root) *points* to start the iterations and generally have superlinear convergence.

Fixed-point iteration (Successive Substitution OR One-point Iteration)

All equations of the form $f(x)=0$ can be written as $x=\phi(x)$. For example, one of our Batman equations $f(t) = ut + \frac{1}{2}gt^2 - \sqrt{t} = 0$ could be written in any of the following forms:

$$t = \frac{\sqrt{t} - \frac{1}{2}gt^2}{u}; \quad t = \left(ut + \frac{1}{2}gt^2\right)^2; \qquad t = \sqrt{\frac{2}{g}\left(\sqrt{t} - ut\right)}; \qquad t = (u+1)t + \frac{1}{2}gt^2 - \sqrt{t} \quad (3.10)$$

If a root of $f(x)=0$ is ξ then obviously $\xi=\phi(\xi)$ indicating that it is a **fixed point**[1] of $\phi(x)$. Now, starting from an initial guess of $x^{(0)}$, we obtain $\phi(x^{(0)})$ and compare with $x^{(0)}$. If their values are same (if we are so lucky, we may think of buying a few lottery tickets!) then $x^{(0)}$ is a root; if not, we take $x^{(1)}=\phi(x^{(0)})$ as our estimate of the root at the next iteration. Following this process of **successive substitution** of the argument of the function $\phi(x)$, we hope to converge to the root, i.e., reach a point when two successive estimates of the root are *nearly* identical. Since the iterations require only one point to compute the next estimate (as against the use of two points in the Secant method to be described later in this section), the scheme may also be called **single-point iteration** or **one-point iteration**. Sometimes the term *Picard iteration* is also used but it is more commonly used in connection with the solution of initial value problems of ordinary differential equations.

The sequence of iteration is written as $x^{(i+1)} = \phi(x^{(i)})$ and the iterations are stopped when the approximate error $\varepsilon^{(i+1)} = |x^{(i+1)} - x^{(i)}|$ or the relative approximate error $\varepsilon_r^{(i+1)} = \dfrac{\varepsilon^{(i+1)}}{|x^{(i+1)}|}$ falls below the desired error tolerance (sometimes $x^{(i)}$ is used in the denominator for the relative

[1] The function $\phi(x)$ may be thought of as mapping the point x to a new point, say, x^*. Generally this mapping will cause the points x and x^* to be some distance apart. If there is *no movement* while mapping some x, say, x_p, [i.e., $x_p = \phi(x_p)$], x_p is called a *fixed point* of the function $\phi(x)$.

error. If the iterations are converging to the root, both these values would be almost equal). Since the root is not bracketed between two successive estimates, there is no guarantee that the method will converge. We would now look at the conditions under which this method converges and find the rate of convergence.

The true error at iteration *i+1* could be written as

$$e^{(i+1)} = \xi - x^{(i+1)} = \phi(\xi) - \phi(x^{(i)}) \tag{3.11}$$

For the analysis of the rate of convergence, we assume that the function $\phi(x)$ is *differentiable* although it is not a necessary condition for convergence (the iterations may converge even when the function is not differentiable *and is only piecewise continuous* near the root). Using the derivative mean value theorem, we obtain

$$e^{(i+1)} = \phi'(\hat{x})(\xi - x^{(i)}) = \phi'(\hat{x})e^{(i)} \tag{3.12}$$

where $\hat{x} \in (\xi, x^{(i)})$. Thus the fixed point iteration method is linearly convergent[1] and, to ensure convergence, $|\phi'(\hat{x})|$ must be less than 1. In general, $\phi'(x)$ will be different at different points and it would be logical to distinguish between a near-convergence (i.e., when $x^{(i)}$ is *near* ξ) and a far-convergence (when $x^{(i)}$ is *far* from ξ). Since we do not bracket the root in the open methods, there is a chance that the initial (or a subsequent) estimate of the root may be far away from the true root. However, for our discussions on convergence, we will concentrate on the near-convergence only, i.e., we assume that the initial guess, x_0, is sufficiently close to the root, ξ. Following may be inferred from eq. 3.12:

- If $|\phi'(x)| < 1$ for all x, the fixed point iteration sequence will be guaranteed to converge[2] for all x_0 (Problem 1, Exercise 3.3)
- If $|\phi'(x)| < 1$ for all values of x such that $|\xi - x| \le \alpha$, the fixed point iteration sequence will converge to ξ if $x_0 \in (\xi-\alpha, \xi+\alpha)$. This is a sufficient but not necessary condition for convergence (Problem 2, Exercise 3.3).
- If $\phi'(\hat{x})$ is positive, the error at any iteration would have the same sign as that at the previous iteration (monotonic sequence). For negative values of the derivative, the error will oscillate between positive and negative values. (Figure 3.4)

---

[1] assuming that $\phi'(\xi) \neq 0$. If the first (p−1) derivatives of $\phi(x)$ at $x=\xi$ are zero, and the p^{th} derivative is not zero, the order of convergence will be *p*. This can be shown by expanding $\phi(x^{(i)})$ about ξ.

[2] We ignore the round-off errors here. There are some equations which are inherently ill-conditioned and all numerical schemes of root finding may show large errors. For example, the Wilkinson's polynomial given by $W(x) = \prod_{i=1}^{20}(x - i)$ is notoriously ill-conditioned.

Figure 3.4 Progress of fixed-point iteration scheme

Example 3.3: It is known that a root of the following equation lies close to x = −1.5

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the fixed-point iteration scheme.

Solution:

For the fixed-point iteration schemes, we have to decide on how to write the given equation in the functional form, x=ϕ(x). If we write

$$x = \left(x^3 - 1.2500000x^2 + 1.9530938\right)/1.5625250 \qquad (3.13)$$

it is easy to see that $\phi'(x) = \left(3x^2 - 2.5x\right)/1.562525$ will become larger than 1 for x values near −1, possibly leading to divergence. On the other hand, writing

$$x = \left(1.2500000x^2 + 1.5625250x - 1.9530938\right)^{1/3}$$

we have $\phi'(x) = \left(\frac{1}{3}\right)\left(1.2500000x^2 + 1.5625250x - 1.9530938\right)^{-2/3}\left(2.5x + 1.562525\right)$ which has a smaller magnitude near the root. The following table shows the iteration using

$$x^{(i+1)} = \phi\left(x^{(i)}\right) = \left[1.2500000\left(x^{(i)}\right)^2 + 1.5625250x^{(i)} - 1.9530938\right]^{1/3}$$

Iteration no.	x	ε_r (%)	ϕ (x)
0	-1.5		-1.14073
1	-1.14073	31.49506	-1.28239
2	-1.28239	11.04697	-1.23882
3	-1.23882	3.517025	-1.25368
4	-1.25368	1.185173	-1.24877
5	-1.24877	0.393456	-1.25041
6	-1.25041	0.131321	-1.24986
7	-1.24986	0.043754	-1.25005
8	-1.25005	0.014586	-1.24998
9	**-1.24998**	0.004862	-

Note the linear convergence of the iterations as the error reduces by a factor of nearly 3 at each iteration[1]. It would be instructive to try different starting values in the table above (e.g., −2, −1, and, especially, 0 or −2.5). If we use Eq. (3.13) as the iterative scheme starting with an initial guess of −1.5, the iterations quickly diverge and if we use a starting value of −1, the iterations move towards the positive roots (in fact, it is nearly impossible to obtain the negative root of this function). In general, if we have no idea about the root, it would be difficult to decide which alternative expression for $\phi(x)$ would be better. The fixed point theorem[2] may be helpful under some circumstances.

---

[1] It is easy to see that near the root, $\phi'(x) = -1/3$.

[2] If $\phi(x)$ is continuous on [*a*,*b*] and $\phi(x) \in [a,b] \quad \forall \quad x \in [a,b]$ then $\phi(x)$ has a fixed point in [*a*,*b*].

While the fixed point iteration scheme does not require bracketing, it is still only linearly convergent. It is, therefore, desirable to explore some other open methods.

Newton-Raphson Method (or Tangent Method)

In the linear interpolation method, we approximated the function by a straight line within the bracketing interval. This line was obtained by joining the function values at either end of the interval. If we have a single point and wish to approximate the function by a straight line in the neighbourhood of this point, a natural choice would be the **tangent** at that point. Thus, starting from an initial estimate of the root, we approximate the function by the tangent at this point, and obtain a new estimate as the point where this linear approximation has a root (Figure 3.5). The method was developed by **Newton**[1] and, about 20 years later but independently, by **Raphson**.

Figure 3.5 Estimate of root in the Newton-Raphson scheme

The linear approximation of the function *f(x)* near the point x_i is thus given by

$$\tilde{f}(x) = f_i + (x - x^{(i)}) f'(x^{(i)}) \tag{3.14}$$

and the new estimate of the root, which is a zero of $\tilde{f}(x)$, is given by

$$x^{(i+1)} = x^{(i)} - \frac{f_i}{f'(x^{(i)})} \tag{3.15}$$

Assuming that f(x) is *twice differentiable*, we may write the Taylor series expansion about $x^{(i)}$ as

$$f(x) = f(x^{(i)}) + (x - x^{(i)}) f'(x^{(i)}) + \frac{(x - x^{(i)})^2}{2} f''(\hat{x}) \tag{3.16}$$

Moreover, if $0 < |\phi'(x)| < 1 \quad \forall \quad x \in [a,b]$ then $\phi(x)$ has a *unique* fixed point in [*a*,*b*].

[1] Heron of Alexandria was probably the earliest to use a similar method in 1st century and al-Tusi of Iran in 12th century and Viete of France in 16th century also used the method. However, we would follow the most commonly used nomenclature of Newton's method or Newton-Raphson method. Sometimes, the iterative technique applied to a single nonlinear equation is called the Newton's method and to a set of simultaneous nonlinear equations is called the Newton-Raphson method.

where $\hat{x} \in (x, x^{(i)})$. Letting x approach the root, ξ, and assuming that ξ is a simple root (so that $f'(x) \neq 0$ for all x close to ξ), we get

$$0 = f_i + (\xi - x^{(i)}) f'(x^{(i)}) + \frac{(\xi - x^{(i)})^2}{2} f''(\hat{x}) \tag{3.17}$$

and

$$e^{(i+1)} = \xi - x^{(i+1)} = \xi - \left(x^{(i)} - \frac{f_i}{f'(x^{(i)})} \right) = -\frac{(\xi - x^{(i)})^2}{2 f'(x^{(i)})} f''(\hat{x}) = -\frac{f''(\hat{x})}{2 f'(x^{(i)})} \left(e^{(i)} \right)^2 \tag{3.18}$$

in which $\hat{x} \in (\xi, x^{(i)})$. If the iterations converge to the root, both $x^{(i)}$ and $\hat{x}$ will approach ξ and $\lim_{i \to \infty} \frac{\left| e^{(i+1)} \right|}{\left| e^{(i)} \right|^2} = \left| \frac{f''(\xi)}{2 f'(\xi)} \right|$. Thus the Newton-Raphson method is *quadratically convergent*[1].

If we assume that for all points x_a and x_b "near" the root, $\left| \frac{f''(x_a)}{2 f'(x_b)} \right|$ has an upper bound of M, we can write $\left| M \cdot E_{i+1} \right| \leq \left(M \cdot E_i \right)^2$. Therefore, if we assume that the initial guess $x^{(0)}$ is sufficiently near the root **and** $\left| M \cdot E_0 \right| < 1$, the iterations will converge and $\left| E_i \right| \leq \frac{\left(M \cdot E_0 \right)^{2^i}}{M}$. However, since the root is not known beforehand it is difficult to use this criterion.

A more usable criterion for convergence is (Figure 3.6):

> If there is an interval $[x_l, x_u]$ such that f_l and f_u have opposite signs; $f'(x)$ is neither zero nor does it change sign in the interval; and $\left| \frac{f'(x)}{f(x)} \right|$ at both x_l and x_u is less than $(x_u - x_l)$; the Newton-Raphson method will converge from any initial guess within the interval.

Figure 3.6 Convergence criterion for Newton-Raphson scheme

[1] The quadratic convergence may also be shown by writing the Newton iteration scheme as a fixed point scheme with $\phi\left(x^{(i)}\right) = x^{(i)} - \frac{f\left(x^{(i)}\right)}{f'\left(x^{(i)}\right)}$ and showing that $\phi'(\xi) = 0$.

Example 3.4: It is known that a root of the following equation lies close to x=1.

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the Newton-Raphson scheme.

Solution:

The function derivative is easily obtained in this case. The table below shows the iterations -

Iteration no.	$x^{(i)}$	f_i	f_i'	$x^{(i+1)}$ (Eq. 3.15)	ε_r (%)
0	1	0.140569	-1.06253	1.132297	11.68394
1	1.132297	0.032945	-0.54698	1.192528	5.050672
2	1.192528	0.008007	-0.27748	1.221383	2.362551
3	1.221383	0.001962	-0.14065	1.235334	1.129262
4	1.235334	0.000473	-0.07271	1.241832	0.523306
5	1.241832	0.000104	-0.04066	1.24439	0.205515
6	1.24439	1.62E-05	-0.02798	1.244969	0.046522
7	1.244969	8.33E-07	-0.02511	**1.245002**	0.002666

Note the quadratic convergence as x approaches the root, the error being proportional to the square of the error at previous iteration[1].

Example 3.5: It is known that a root of the following equation lies close to x=1.

$$x^3 - 1.25020000x^2 - 1.56249999x + 1.95343750 = 0$$

Obtain this root, to an accuracy of 0.01%, using the Newton-Raphson scheme.

Solution:

The table below shows the iterations -

Iteration no.	$x^{(i)}$	f_i	f_i'	$x^{(i+1)}$ (Eq. 3.15)	ε_r (%)
0	1	0.140738	-1.0629	1.132409	11.69268
1	1.132409	0.032999	-0.54693	1.192745	5.058566
2	1.192745	0.008036	-0.27692	1.221763	2.375112
3	1.221763	0.001985	-0.13928	1.236013	1.152908
4	1.236013	0.000493	-0.06984	1.243076	0.56821
5	1.243076	0.000123	-0.03497	1.246593	0.282081
6	1.246593	3.07E-05	-0.0175	1.248347	0.140516
7	1.248347	7.67E-06	-0.00876	1.249222	0.070078
8	1.249222	1.91E-06	-0.00439	1.249658	0.034894
9	1.249658	4.75E-07	-0.00221	1.249874	0.017214
10	1.249874	1.16E-07	-0.00113	**1.249976**	0.008175

[1] Strictly speaking, since Eq. (2.18) uses the absolute true error, the true error at any iteration should be equal to a constant times the square of the true error at the previous iteration, the constant being equal to $-f''/2f'$ at the root (for this example, this value is close to 100 at x=1.245). However, since the magnitude of the root is close to 1, and the asymptotic error constant is close to 100, it turns out that the percent relative approximate error at any iteration is roughly equal to the square of that at the previous iteration.

Note the ***linear convergence*** as x approaches the root (the error is nearly half of the error at previous iteration). The reason for reduction in the order of convergence from quadratic to linear is the presence of a double root. We will discuss more about it in section 3.4.

The faster convergence of the Newton-Raphson method comes with the price one has to pay in terms of additional computations for obtaining the derivative of the function. The additional computational effort depends on the type of function and, for a polynomial, is almost equal to that involved in computing the function. We may define an *Efficiency parameter*, η, for an iterative scheme as follows:

> Let p be the order of convergence and C the asymptotic error constant for the iterative scheme, which requires a computational effort of κ times that required for evaluating the function $f(x)$. For example, for the Newton-Raphson scheme applied to a polynomial $f(x)$: p=2, $C=\left|\frac{f''(\xi)}{2f'(\xi)}\right|$ and $\kappa\approx 2$. If we assume that the iterative scheme converges to the root ξ (within a specified tolerance of δ) in N iterations, and the asymptotic rate of convergence is applicable throughout the iterative sequence, we get
>
> $$\delta \approx e^{(N)} = C\left|e^{(N-1)}\right|^p = C\left|C\left|e^{(N-2)}\right|^p\right|^p = C^{1+p}\left|e^{(N-2)}\right|^{p^2} \ldots = C^{1+p+p^2+\ldots p^{N-1}}\left|e^{(0)}\right|^{p^N} = C^{\frac{p^N-1}{p-1}}\left|e^{(0)}\right|^{p^N}$$
>
> from which the value of N could be obtained. The computational effort would then be proportional to $N\kappa$ and the efficiency is inversely related to the effort. From the above equation, $N = N\left(e^{(0)}, \delta, C, p\right)$, i.e., a function of the initial error, desired error, asymptotic error constant and the order of convergence. To simplify the analysis, *treating the term* $C^{\frac{p^N-1}{p-1}}$ *as a constant*, for a given initial error and tolerance, we obtain $N \propto \frac{1}{\ln p}$ and, therefore, $N\kappa \propto \frac{1}{\ln p^{1/\kappa}}$. Thus $p^{1/\kappa}$ is a reasonably good indicator of η, the efficiency parameter of the iterative scheme.

For polynomial $f(x)$, the efficiency parameter of the Newton-Raphson method $= 2^{1/2} = 1.414$. However, for some types of functions, the evaluation of derivative may require more effort than that required in the function evaluation. Therefore, some times a modified Newton-Raphson method is used in which the derivative is not evaluated at every iteration but is updated at every, say, fifth iteration. Also, for some functions, it may be very difficult to obtain the derivative. Therefore, it is advisable to look for other methods which do not involve the computation of the derivative but still provide a superlinear convergence. One option is to obtain the derivative at any point numerically by evaluating the function at another neighbouring point and then using the divided difference. However, it is generally not recommended since there would be large round-off errors and, as discussed in the next section, it would be less efficient than the *Secant method*.

Secant Method

As seen in the previous section, the Newton-Raphson method approximates the function near any point by the tangent at that point. In the Secant method, the approximating straight line is taken as the **Secant** (a straight line that intersects a curve at two or more points). Thus, starting from any two points (*not necessarily bracketing the root*) on the function curve, we approximate the function by the Secant passing through these points, and obtain an estimate of the root as the point where this linear approximation has a root (Figure 3.7). For the next iteration, this estimate of the root replaces *one of the previous two* points (Generally the earlier iteration is discarded. However, an alternative would be to discard the point at which the function has a larger magnitude, see Problem 3, Exercise 3.3). Starting from two initial values, $x^{(0)}$ and $x^{(1)}$, the iterative sequence is given by

$$x^{(i+1)} = x^{(i)} + \frac{x^{(i)} - x^{(i-1)}}{f_i - f_{i-1}}(-f_i) \tag{3.19}$$

Figure 3.7 Estimation of root in Secant method

Note that Eq. (3.19) is similar to Eq. (3.1) used in the linear interpolation method, which is not surprising since both of these are based on approximating the function by a straight line and obtaining its point of intersection with the axis. The only difference is that in linear interpolation method, the root is always bracketed between the estimates at two successive iterations. In the Secant method, it is not necessary that the root lies between $x^{(i)}$ and $x^{(i+1)}$. An analysis similar to that performed for the Linear Interpolation method (Eq. 3.9) results in the error of the Secant method as

$$e^{(i+1)} = -\frac{f''(\hat{x}_1)}{2f'(\hat{x}_2)} e^{(i)} e^{(i-1)} \tag{3.20}$$

where $\hat{x}_1 \in (\xi, x^{(i)}, x^{(i-1)})$ and $\hat{x}_2 \in (x^{(i)}, x^{(i-1)})$. Assuming that the iterations converge to the root ξ, we get, $\lim_{i\to\infty} \left|e^{(i+1)}\right| = \left|\frac{f''(\xi)}{2f'(\xi)}\right| \left|e^{(i)}\right| \left|e^{(i-1)}\right|$. Thus, using the relationships $e^{(i+1)} = C\left|e^{(i)}\right|^p$ and $e^{(i-1)} = \left|\frac{e^{(i)}}{C}\right|^{1/p}$, the order of the Secant method and its asymptotic error constant are obtained by $p^2 = p+1$ and $C = \left|\frac{f''(\xi)}{2f'(\xi)}\right|^{1/p}$ implying that the order is 1.618 (better than Bisection and False Position but not as good as Newton Raphson). As far as efficiency is concerned, if the computational effort in evaluating the derivative of a function is more than 0.44 times

($2^{1/1.44}$=1.618) that required for a function evaluation, the Secant method is likely to be more efficient than the Newton-Raphson method.

Example 3.6: It is known that a root of the following equation lies close to x=1.

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the Secant method.

Solution:

We start the iterations with the points x=0 and x=1 and use the two most recent estimates at each iteration. The table below shows the iterations.

Iteration no.	$x^{(i)}$	f_i	$x^{(i+1)}$ (Eq. 3.19)	ε_r (%)
	0	1.953094		
0	1	0.140569	1.077554	7.197238
1	1.077554	0.069158	1.152661	6.515953
2	1.152661	0.022705	1.189371	3.08653
3	1.189371	0.008906	1.213064	1.953109
4	1.213064	0.003299	1.227003	1.136029
5	1.227003	0.001248	1.235487	0.686695
6	1.235487	0.000461	1.240463	0.401159
7	1.240463	0.000164	1.243215	0.221335
8	1.243215	5.25E-05	1.244507	0.103843
9	1.244507	1.3E-05	1.24493	0.03399
10	1.24493	1.8E-06	**1.244999**	0.005494

The asymptotic error constant, *C*, is equal to $\left|\frac{f''(\xi)}{2f'(\xi)}\right|^{0.618}$ which is close to 17, and the order of convergence, *p*, is 1.618. The absolute true error[1] at iterations 8, 9, and 10 is 0.000495, 0.0000717, and 0.00000334, respectively, and it is rather straightforward to see that these values are consistent with the computed values of *C* and *p*.

As seen above, the Secant method has a lower order of convergence than the Newton-Raphson method. In fact, it has been shown to be generally true that any iterative method which uses only one additional function evaluation at each iteration cannot have quadratic convergence. In order to improve upon the order of convergence, without using derivatives, Muller's method and Steffensen's method could be used as described next.

Muller's Method

In this method, instead of a linear interpolation using the function values at $x^{(i-1)}$ and $x^{(i)}$, the function value at $x^{(i-2)}$ is also utilized to perform a quadratic interpolation (A side benefit of using a quadratic interpolation is that we may obtain complex roots also, but that will be discussed later!). Another option is to use *inverse interpolation*, i.e., expressing x as a quadratic function of *f*, say *g(f)*, and then obtaining the root as *g(0)*. However, this will not

[1] The True Value of the root being 1.245002.

work in case of complex roots (see Problem 4, Exercise 3.3). Starting from three points, $x^{(0)}$, $x^{(1)}$, and $x^{(2)}$, the iterative sequence is written as

$$x^{(i+1)} = x^{(i)} + \frac{\pm\sqrt{b^2 - 4ac} - b}{2a} \tag{3.21}$$

where a, b, and c are obtained from the quadratic interpolation using function values at $x^{(i-2)}$, $x^{(i-1)}$, and $x^{(i)}$ with the stipulation that the interpolating parabola attains a zero value at the point $x^{(i+1)}$. The analysis is simplified by shifting the origin of the coordinate system to the point $x^{(i)}$. In this new system, x_*, the quadratic polynomial is written as $f(x_*) = ax_*^2 + bx_* + c$ and matching the function values at the three grid points leads to $a = f\left[x^{(i)}, x^{(i-1)}, x^{(i-2)}\right]$ $b = f\left[x^{(i)}, x^{(i-1)}\right] + \left(x^{(i)} - x^{(i-1)}\right) f\left[x^{(i)}, x^{(i-1)}, x^{(i-2)}\right]$ $c = f_i$. Here, the square brackets indicate the divided differences [1] defined by $f\left[x^{(i)}, x^{(i-1)}\right] = \frac{f_i - f_{i-1}}{x^{(i)} - x^{(i-1)}}$ and $f\left[x^{(i)}, x^{(i-1)}, x^{(i-2)}\right] = \frac{f\left[x^{(i)}, x^{(i-1)}\right] - f\left[x^{(i-1)}, x^{(i-2)}\right]}{x^{(i)} - x^{(i-2)}}$. Since the form of eq (3.21) is subject to severe round-off errors (section 1.3), an alternative form is used for the iterative sequence as

$$x^{(i+1)} = x^{(i)} - \frac{2c}{b \pm \sqrt{b^2 - 4ac}} \tag{3.22}$$

The sign of the square root is chosen in such a way as to make the magnitude of the denominator large (for $b^2 - 4ac < 0$, i.e., complex denominator, both the signs will give the same magnitude, for real values we choose the negative sign if b is negative and vice versa), thereby choosing $x^{(i+1)}$ closer to $x^{(i)}$ (see Fig. 3.8. An alternative would be to choose the value of $x^{(i+1)}$ which gives smaller function magnitude but it would require more function computations, see Problem 5, Exercise 3.3). As in the Secant method, we again have a choice in discarding one of the three points, $x^{(i-2)}$, $x^{(i-1)}$, and $x^{(i)}$, for the next iteration. We may discard $x^{(i-2)}$, or the point farthest from $x^{(i+1)}$, or the point at which the function has the largest magnitude (see Problem 6, Exercise 3.3). Performing an analysis of interpolation error, similar to that done for the Secant method, we obtain

$$\lim_{i\to\infty} e^{(i+1)} \approx -\frac{f'''(\xi)}{6f'(\xi)} e^{(i-2)} e^{(i-1)} e^{(i)} \tag{3.23}$$

If p denotes the order of the Muller method, we get $p^3 = p^2 + p + 1$ implying that the order is 1.839 (better than Secant but not as good as Newton Raphson). The asymptotic error constant is given by $C = \left|\frac{f''(\xi)}{6f'(\xi)}\right|^{(p-1)/2}$.

[1] Detailed discussion of divided difference is available in Chapter 4.

Figure 3.8 Choosing the more appropriate root for Muller's method

Example 3.7: It is known that a root of the following equation lies close to x=1.

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the Muller's method starting with the points 0, 0.5, and 1 to start the iterations.

Solution:

We use the three most recent estimates at each iteration. The table below shows the iterations.

Iteration no.	$x^{(i)}$	f_i	$x^{(i+1)}$ (Eq. 3.22)	ε_r (%)
	0	1.953094		
	0.5	0.984331		
0	1	0.140569	1.091296	
1	1.091296	0.058912	1.181861	7.662871
2	1.181861	0.01123	1.226125	3.61012
3	1.226125	0.00135	1.241139	1.209691
4	1.241139	0.000133	1.244831	0.296539
5	1.244831	4.35E-06	1.245002	0.013728
6	1.245002	1.25E-08	**1.245002**	4.01E-05

The order of convergence is 1.839 and the asymptotic error constant is about 4.4. Computing the absolute true errors, one can easily verify that these values hold good as the iterations approach the root. Had we taken the starting points as 0, 1, and 2, we would have encountered complex values which is desirable if we know that the function has complex roots. In this case, however, it unnecessarily complicates the computation.

Steffensen's Method

As discussed earlier, the Newton-Raphson method has quadratic convergence but requires computation of the derivative of the function. The Secant method, on the other hand, may be thought of as approximating the derivative of the function by a secant. It does not require derivative evaluation but has sub-quadratic convergence (p=1.618). In the Steffensen's method, a judicious choice of the secant is made for the approximation of the derivative, such that a quadratic convergence is achieved. The iterative scheme is written as

$$x^{(i+1)} = x^{(i)} - \frac{f_i}{\dfrac{f\left(x^{(i)} + f_i\right) - f_i}{f_i}} \tag{3.24}$$

which indicates that the function is approximated by the secant[1] passing through $x^{(i)}$ and $x^{(i)}+f_i$. By expanding $f\left(x^{(i)}+f_i\right)$ in a Taylor's series about the point $x^{(i)}$, and using the Taylor's series expansion of $f(x)$ about the root, ξ, it can be shown that

$$\lim_{i\to\infty} e^{(i+1)} = -\frac{f''(\xi)}{2f'(\xi)}\left[1+f'\left(\xi\right)\right]\left[e^{(i)}\right]^2 \tag{3.25}$$

establishing the quadratic convergence of the Steffensen's method[2].

Example 3.8: It is known that a root of the following equation lies close to x=1.

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the Steffensen's method.

Solution:

The table below shows the iterations.

Iteration no.	$x^{(i)}$	f_i	f(x+f)	$x^{(i+1)}$ (Eq. 3.24)	ε_r (%)
0	1	0.140569	0.028568	1.176423	
1	1.176423	0.013075	0.008869	1.217067	3.33948
2	1.217067	0.002614	0.002209	1.233916	1.365463
3	1.233916	0.000581	0.000535	1.241335	0.597706
4	1.241335	0.000125	0.000119	1.24425	0.234291
5	1.24425	2.01E-05	1.96E-05	1.244954	0.056542
6	1.244954	1.2E-06	1.17E-06	**1.245002**	0.003814

The order of convergence is 2 and the asymptotic error constant is about 97, both of which may be verified by computing the absolute true errors as the iterations approach the root. Had we taken the starting point as 2, we would have reached the other root at 1.245 in 13 iterations. Starting from an initial guess of 0 requires more than 2000 iterations to reach the correct root 1.245, although the relative approximate error is very small in the initial iterations because $f\left(x^{(i)}+f_i\right)$ is of much larger magnitude than f_i. We should, however, not stop the iteration even if the approximate error is small because the function value stays very large. The table below shows an extract of these computations.

Iteration no.	$x^{(i)}$	f_i	f(x+f)	$x^{(i+1)}$ (Eq. 3.24)	ε_r (%)
0	0	1.953094	1.58334	10.31653	
1	10.31653	950.7924	8.87E+08	10.31551	0.009884
2	10.31551	950.4948	8.86E+08	10.31449	0.009888
3	10.31449	950.1971	8.85E+08	10.31347	0.009892
.	.	.	.	.	.
2475	1.25531	8.05E-06	8.27E-06	1.255007	0.024099

[1] In the next subsection, we will see another interpretation of the Steffensen method.

[2] In fact, all schemes of the form $x^{(i+1)} = x^{(i)} - \dfrac{f_i^2}{cf_i^2 + f\left(x^{(i)}+f_i\right) - f_i}$ would have quadratic convergence. Steffensen's methods is a special case with c=0.

2476	1.255007	2.36E-07	2.42E-07	1.254998	0.00075

Aitken Extrapolation

Another alternative to accelerate the convergence of a linearly converging iterative sequence is to use the Aitken extrapolation. For a sequence of iterations converging to the root, ξ, if the differences between successive iterates are approximated as forming a geometric series, i.e., $x^{(i)} - x^{(i-1)} = k(x^{(i-1)} - x^{(i-2)})$, k being the multiplying factor or common ratio, the root can be obtained by extrapolation as

$$\xi = \lim_{n\to\infty} x^{(n)} = x^{(i)} + \sum_{j=1}^{\infty}\left(x^{(i+j)} - x^{(i+j-1)}\right) = x^{(i)} + \left(x^{(i)} - x^{(i-1)}\right)\sum_{j=1}^{\infty} k^j = x^{(i)} + \left(x^{(i)} - x^{(i-1)}\right)\frac{k}{1-k} \quad (3.26)$$

Thus the extrapolated sequence, $x_*^{(i)}$, may be written as

$$x_*^{(i)} = x^{(i)} + \left(x^{(i)} - x^{(i-1)}\right)\frac{\dfrac{x^{(i)} - x^{(i-1)}}{x^{(i-1)} - x^{(i-2)}}}{1 - \dfrac{x^{(i)} - x^{(i-1)}}{x^{(i-1)} - x^{(i-2)}}} = x^{(i)} - \frac{\left(x^{(i)} - x^{(i-1)}\right)^2}{x^{(i)} - 2x^{(i-1)} + x^{(i-2)}} \quad (3.27)$$

and will generally converge faster to the root.

Example 3.9: It is known that a root of the following equation lies close to $x = -1.5$

$$x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938 = 0$$

Obtain this root, to an accuracy of 0.01%, using the fixed-point iteration scheme with Aitken extrapolation (see Box 3.2)

$$x_{i+1} = \phi(x_i) = \left(1.2500000x_i^2 + 1.5625250x_i - 1.9530938\right)^{1/3}$$

Solution:

The following table shows the iteration using the Aitken extrapolation at every third step (since we need three successive iterates to apply Eq. 3.27) to refine the estimate of the root. Note that the *extrapolated* value becomes the iterate at the next step (shown by italics) and also note that at every third step $\phi(x)$ does not need to be computed (shown by a dash).

Iteration no.	x	ϕ (x)	ε_r (%)	x_* (Eq. 3.27)
0	-1.5	-1.14073		
1	-1.14073	-1.28239	31.49506	
2	-1.28239	-	11.04697	*-1.24233*
3	*-1.24233*	-1.25254	3.224835	
4	-1.25254	-1.24915	0.814837	
5	-1.24915	-	0.270859	*-1.24999*
6	*-1.24999*	-1.25	0.067391	
7	-1.25	**-1.25**	0.000553	

Comparing with Example 3.3, we note that the convergence is faster (7 iterations instead of 9) and the error is also much smaller.

Box 3.2: **Steffensen's method from Aitken's extrapolation**

Application of Aitken's extrapolation to a fixed point iteration scheme, written as $x_{new} = x_{old} + f\left(x_{old}\right)$, would give rise to the Steffensen's method discussed in the previous subsection.

Substituting $x^{(i+1)} = x^{(i)} + f_i$ and $x^{(i+2)} = x^{(i+1)} + f\left(x^{(i)} + f_i\right) = x^{(i)} + f_i + f\left(x^{(i)} + f_i\right)$ in Eq. (3.27) written in terms of $x^{(i)}, x^{(i+1)}$, and $x^{(i+2)}$, we get

$$\begin{aligned} x_{new} &= x^{(i+2)} - \frac{\left(x^{(i+2)} - x^{(i+1)}\right)^2}{x^{(i+2)} - 2x^{(i+1)} + x^{(i)}} \\ &= x^{(i)} + f_i + f\left(x^{(i)} + f_i\right) - \frac{\left[f\left(x^{(i)} + f_i\right)\right]^2}{f\left(x^{(i)} + f_i\right) - f_i} \\ &= x^{(i)} - \frac{f_i^2}{f\left(x^{(i)} + f_i\right) - f_i} \end{aligned}$$

which is the same as Eq. (3.24).

We may apply the extrapolation to other linearly convergent methods like bisection, false position, or the Newton-Raphson method near a multiple root (see Example 3.5), also.

Exercise 3.3

1. Solve the equation $x - \cos\left(x/2\right) = 0$ by using the fixed-point iteration scheme, $x = \cos\left(x/2\right)$. Since the scheme will always converge (why?), try very different initial guesses of the root, e.g., 0, −100, 100.
2. An iterative scheme to find the square root of a number, c, may be written as $x_{i+1} = \phi\left(x_i\right) = 0.5\left(x_i + c/x_i\right)$. Find the square root of 19 with an initial guess of 4.5. It can be shown that $\left|\phi'\left(x\right)\right|$ is greater than 1 for $x < \sqrt{c/3}$. Use a starting guess of 1 and show that convergence is achieved (why?)
3. Solve the problem discussed in Example 3.6 using the secant method but instead of keeping the two most recent estimates, discard the point where the magnitude of the function is larger.
4. Find a root of $f\left(x\right) = 1.5\,x^2 - 5.5\,x - 6 = 0$ by obtaining the function values at x=2,3, and 4, expressing x as a quadratic function *g(f)*, and then putting f=0 in the inverse function, g. Use the three most recent points to repeat till convergence. Try the same technique with −6 replaced by +6 in the equation.

5. In Example 3.7, Muller's method was used with the sign of the square root in Eq. 3.22 chosen to make the "correction" in the estimate of the root small. Re-do this example but choosing the sign of the radical which will give the smaller value of the function at the new estimate of the root.
6. In Example 3.7, Muller's method was used with the three most recent estimates of the root. Re-do this example but discarding the point at which the function has the largest magnitude. Try another option in which the point farthest from the most recent estimate is discarded.
7. An equation with infinite roots is $f(x) = x - \tan x = 0$. What can be said about the approximate location of the roots? Find the first three positive roots by using any of the methods described in this section.

3.4 Complex roots and Multiple roots

Although most physical problems involve the determination of real roots, sometimes it is desired to obtain complex roots also. For example, eigenvalues of a real matrix (see chapter 2) may be complex. If we use complex arithmetic in the computer program (which most scientific programming languages are capable of) the open methods[1] described for obtaining the real roots may be easily extended to equations involving complex roots. Obviously, if the equation involves a polynomial with only real coefficients, we will have to start with an initial guess which is complex to reach the complex root (except the fixed-point method and Muller's method, Eq. 3.22, which may lead to a complex root even with real starting values). Sometimes, however, *f(x)* may involve terms which lead the iterations to the complex root with real starting guess.

Example 3.10: Find one of the complex roots of the equation

$$x^3 - 2.850x^2 + 3.910x - 2.121 = 0$$

to an accuracy of 0.01%, using the fixed-point, Newton-Raphson, and Muller's methods.

Solution:

The following table shows an extract of the iterations using the fixed point iteration written as

$$x = \sqrt{\left(x^3 + 3.910x - 2.121\right)/2.850}$$

to create the possibility of getting a complex root. The error is based on the magnitude of the complex number.

Iteration no.	x	ε_r (%)	ϕ (x)
0	1		0.98924
1	0.98924	1.087667	0.976028
2	0.976028	1.353694	0.959726
.	.	.	.

[1] Bracketing methods will not work as the function values will be complex and it would not be possible to bracket the root. There is an algorithm proposed by Lehmer in 1961 that may be used to "encircle" the root, but it will not be discussed here.

11	0.384786	54.938583	0 + 0.443082 i
12	0 + 0.443082 i	13.157071	0.314402 + 0.918182 i
13	0.314402 + 0.918182 i	54.345855	0.569453 + 0.951449 i
.	.	.	.
38	0.898444 + 1.100135 i	0.016361	0.898761 + 1.099786 i
39	**0.898761 + 1.099786 i**	0.004948	-

The iterations approach the root 0.9+1.1 i. The other complex root would be its conjugate.

The Newton-Raphson method must start from a complex initial guess to obtain the complex root. The table below shows the iterations -

Iteration no.	$x^{(i)}$	f_i	f_i'	$x^{(i+1)}$ (Eq. 3.15)	ε_r (%)
0	i	0.729 + 2.910 i	0.910 – 5.700 i	0.478 + 0.796 i	7.7252
1	0.478 + 0.796 i	0.103 + 0.985 i	-0.029 – 2.254 i	0.631 + 0.876 i	14.0383
2	0.631 + 0.876 i	0.197 + 0.648 i	-0.795 – 1.675 i	0.730 + 0.913 i	7.6342
.	.	.	.	.	.
24	0.898 + 1.101 i	0.006 - 0.002 i	-2.426 – 0.346 i	0.898 + 1.100 i	0.0248
25	0.898 + 1.100 i	0.005 + 0.000 i	-2.421 – 0.344 i	**0.898 + 1.100 i**	0.0090

For Muller's method, we use the three most recent estimates at each iteration and also use the negative sign in the denominator of Eq. 3.22 (with positive sign, the iterations converge to the real root). The table below shows the iterations (the numbers are shown up to 3 decimal places only but the error computation is done with the more precise values).

Iteration no.	$x^{(i)}$	f_i	$x^{(i+1)}$ (Eq. 3.22)	ε_r (%)
	0.000	-2.121		
	1.000	-0.061		
0	2.000	2.299	-13.761	85.466
1	-13.761	-3201.390	2.170	534.271
2	2.170	3.159	2.806	22.694
3	2.806	8.509	3.705	24.257
4	3.705	24.108	1.768-0.617 i	97.908
5	1.768-0.617 i	0.476-1.744 i	1.483-0.766 i	12.182
6	1.483-0.766 i	-0.266-1.124 i	1.157-0.863 i	15.629
7	1.157-0.863 i	-0.326-0.506 i	0.903-1.018 i	6.082
8	0.903-1.018 i	-0.032-0.176 i	0.892-1.098 i	3.818
9	0.892-1.098 i	0.017-0.009 i	0.900-1.100 i	0.476
10	0.900-1.100 i	0.000-0.000 i	**0.900-1.100 i**	0.007

While these methods do provide the complex roots, the use of complex arithmetic leads to much larger computation times than that for real arithmetic. A frequent occurrence of complex roots is in the computation of eigenvalues of a real matrix. Since the equation to be solved is a polynomial with all real coefficients, the complex roots will occur in conjugate pairs. Therefore, instead of finding one root at a time, it would be more practical to find a quadratic factor (which corresponds to a conjugate pair) of the polynomial. This computation would require only real arithmetic and the complex roots, if present, are readily obtained

using the formula for the quadratic equation. A recursive algorithm using this idea was first suggested by Bairstow (and later modified by Lin and various other researchers).

Bairstow method

Before describing the Bairstow method, it is helpful to look at its counterpart for obtaining a single root of a polynomial (see Box 3.3 for a quick review of characteristics of the roots of a polynomial). Expressing the polynomial as[1] $f(x) = \sum_{j=0}^{n} c_j x^j$, one way of finding a root is to divide it by the term (x–r) and obtain a remainder, R, which would be a function of r. If R is zero, r would be a root of the polynomial. If not, we could try changing r by an amount Δr in such a way as to make R equal to zero. Writing the quotient as $\sum_{j=0}^{n-1} d_{j+1} x^j$ and the residual as d_0 (in order to enable us to write a recursive relationship), we have

$$\sum_{j=0}^{n} c_j x^j = (x-r)\sum_{j=0}^{n-1} d_{j+1} x^j + d_0 = d_n x^n + \sum_{j=0}^{n-1} \left(d_j - r d_{j+1}\right) x^j \tag{3.28}$$

Equating the coefficients of different powers of x, we obtain the recursive relations

$$\begin{aligned} d_n &= c_n \\ d_j &= c_j + r d_{j+1} \qquad \text{for } j = n-1 \text{ to } 0 \end{aligned} \tag{3.29}$$

which provides us with the residual, d_0, as a function of the assumed value of the root, r. If we use the Newton-Raphson method to find Δr from

$$d_0(r) + \Delta r \frac{d}{dr}\left[d_0(r)\right] = d_0(r + \Delta r) = 0 \tag{3.30}$$

it can be easily verified that we get the same equation (Eq. 3.15) for the iterative scheme, since $d_0(r)$ is same as f(r).

Box 3.3: **Roots of a polynomial**

Most polynomials occurring in a practical problem would have real coefficients. Hence we describe some properties related to the roots of such polynomials.

In standard form a polynomial is expressed as

$$p_n(x) = \sum_{j=0}^{n} c_{n-j} x^{n-j}$$

If an n^{th} degree polynomial has *k* roots, $r_1, r_2, \ldots, r_k$, which are of multiplicity $m_1, m_2, \ldots, m_k$, respectively, then

$$p_n(x) = (x-r_1)^{m_1}(x-r_2)^{m_2}\ldots(x-r_k)^{m_k} \text{ and } \sum_{i=1}^{k} m_k = n$$

[1] Note that x^j represents *x* raised to the power *j* and should not be confused with $x^{(j)}$ which is the value of *x* at iteration *j*.

The *n* roots of the polynomial follow Viéte's formulae, according to which the k[th] symmetric sum of the roots is given by

$$\sum_{1\le i_1<i_2<\ldots<i_k\le n} r_{i_1} r_{i_2} \ldots r_{i_k} = (-1)^k \frac{c_{n-k}}{c_n} \text{ for each } k=1,2,\ldots,n$$

Thus the sum of all the roots (k=1) is equal to $-c_{n-1}/c_n$, sum of the product of two roots at a time (k=2), i.e., $r_1r_2 + r_1r_3 + \ldots + r_1r_n + r_2r_3 + r_2r_4 + \ldots r_{n-2}r_{n-1} + r_{n-1}r_n$, is equal to c_{n-2}/c_n, and product of all the roots (k=n) is equal to $(-1)^n c_0/c_n$.

If a polynomial has all real coefficients, then either all roots are real or there are even number of non-real complex roots which occur in conjugate pairs. Descartes' rule of signs states that the number of positive roots of $p_n(x)=0$ is either equal to the number of sign changes in the coefficients of $p_n(x)$ (zero coefficients are ignored), or less than that by an *even number* (which represents the number of complex roots occurring in conjugate pairs). Similarly, the number of negative roots of $p_n(x)=0$ is either equal to the number of sign changes in the coefficients of $p_n(-x)$ or less than that by an even number.

If we define ρ as $1+\frac{\max_{0\le j<n}|c_j|}{|c_n|}$, Cauchy provided the upper bound of the roots of the polynomial as ρ and the lower bound as $1/\rho_q$, where ρ_q is the radius of the polynomial $q_n(x)=x_n p_n(1/x)$, i.e., $1+\frac{\max_{0<j\le n}|c_j|}{|c_0|}$. In other words, the magnitude of all roots would be between $\frac{|c_0|}{|c_0|+\max_{0<j\le n}|c_j|}$ and $1+\frac{\max_{0\le j<n}|c_j|}{|c_n|}$.

In the Bairstow method, instead of the synthetic division by (x–r), we carry out a division by the quadratic term $(x-r_1)(x-r_2)$, and aim to make the remainder zero. Irrespective of whether these two roots are real or complex conjugates, the quadratic term may be written in terms of real coefficients[1] as $\left(x^2-\alpha_1 x-\alpha_0\right)$, the quotient as $\sum_{j=0}^{n-2} d_{j+2}x^j$, and the remainder as $d_0 + d_1\left(x-\alpha_1\right)$. We then have

$$\begin{aligned}\sum_{j=0}^{n} c_j x^j &= (x^2-\alpha_1 x-\alpha_0)\sum_{j=0}^{n-2} d_{j+2}x^j + d_1\left(x-\alpha_1\right)+d_0 \\ &= d_n x^n + \left(d_{n-1}-\alpha_1 d_n\right)x^{n-1} + \sum_{j=0}^{n-2}\left(d_j-\alpha_1 d_{j+1}-\alpha_0 d_{j+2}\right)x^j\end{aligned} \tag{3.31}$$

[1] If the two roots are complex conjugates, α_0 will be negative.

giving us the recursive relations

$$\begin{aligned} d_n &= c_n \\ d_{n-1} &= c_{n-1} + \alpha_1 d_n \\ d_j &= c_j + \alpha_1 d_{j+1} + \alpha_0 d_{j+2} \qquad \text{for } j = n-2 \text{ to } 0 \end{aligned} \tag{3.32}$$

We now aim at making the remainder zero by making d_0 and d_1 equal to zero (note that both d_0 and d_1 are functions of the guess values of α_0 and α_1). A Newton scheme could be written as (more details on solving multivariate nonlinear equations are provided in section 3.7)

$$\begin{aligned} d_0(\alpha_0,\alpha_1) + \frac{\partial d_0(\alpha_0,\alpha_1)}{\partial \alpha_0}\Delta\alpha_0 + \frac{\partial d_0(\alpha_0,\alpha_1)}{\partial \alpha_1}\Delta\alpha_1 = d_0(\alpha_0+\Delta\alpha_0,\alpha_1+\Delta\alpha_1) = 0 \\ d_1(\alpha_0,\alpha_1) + \frac{\partial d_1(\alpha_0,\alpha_1)}{\partial \alpha_0}\Delta\alpha_0 + \frac{\partial d_1(\alpha_0,\alpha_1)}{\partial \alpha_1}\Delta\alpha_1 = d_1(\alpha_0+\Delta\alpha_0,\alpha_1+\Delta\alpha_1) = 0 \end{aligned} \tag{3.33}$$

to iteratively improve the guessed values till the changes, $\Delta\alpha_0$ and $\Delta\alpha_1$, are very small. In order to compute the partial derivatives in Eq. (3.33), it is easy to see from Eq. (3.32) that

$$\frac{\partial d_n}{\partial \alpha_0} = 0 \qquad \frac{\partial d_{n-1}}{\partial \alpha_0} = 0 \qquad \frac{\partial d_j}{\partial \alpha_0} = d_{j+2} + \alpha_0 \frac{\partial d_{j+2}}{\partial \alpha_0} + \alpha_1 \frac{\partial d_{j+1}}{\partial \alpha_0} \quad \text{for } j = n-2 \text{ to } 0 \tag{3.34}$$

and, similarly,

$$\frac{\partial d_n}{\partial \alpha_1} = 0 \qquad \frac{\partial d_{n-1}}{\partial \alpha_1} = d_n \qquad \frac{\partial d_j}{\partial \alpha_1} = d_{j+1} + \alpha_0 \frac{\partial d_{j+2}}{\partial \alpha_1} + \alpha_1 \frac{\partial d_{j+1}}{\partial \alpha_1} \quad \text{for } j = n-2 \text{ to } 0 \tag{3.35}$$

If we now define $\Delta_{j,0} = \frac{\partial d_j}{\partial \alpha_0}$ and $\Delta_{j,1} = \frac{\partial d_j}{\partial \alpha_1}$, Eqs. (3.34) and (3.35) could be written as

$$\Delta_{n-1,0} = 0 \qquad \Delta_{n-2,0} = d_n \qquad \Delta_{j-1,0} = d_{j+1} + \alpha_0 \Delta_{j+1,0} + \alpha_1 \Delta_{j,0} \quad \text{for } j = n-2 \text{ to } 1 \tag{3.34a}$$

and

$$\Delta_{n,1} = 0 \qquad \Delta_{n-1,1} = d_n \qquad \Delta_{j,1} = d_{j+1} + \alpha_0 \Delta_{j+2,1} + \alpha_1 \Delta_{j+1,1} \quad \text{for } j = n-2 \text{ to } 0 \tag{3.35a}$$

These equations could be used to obtain the derivatives required in Eq. (3.33) by a recursive algorithm applied separately to Eqs. (3.34a) and (3.35a). However, a close look at these two equations clearly shows the equivalence of $\Delta_{j-1,0}$ and $\Delta_{j,1}$ and the SAME recursive algorithm could be applied to both the equations. Defining $\delta_j = \frac{\partial d_{j-1}}{\partial \alpha_0} \left(\text{i.e., } \Delta_{j-1,0}\right) = \frac{\partial d_j}{\partial \alpha_1} \left(\text{i.e., } \Delta_{j,1}\right)$, we get the following algorithm for computing the derivatives required in Eq. (3.33):

$$\delta_{n-1} = d_n \qquad \delta_{n-2} = d_{n-1} + \alpha_1 \delta_{n-1} \qquad \delta_j = d_{j+1} + \alpha_1 \delta_{j+1} + \alpha_0 \delta_{j+2} \quad \text{for } j = n-3 \text{ to } 0$$

Eq. (3.33) can then be written as

$$\begin{aligned} \delta_1 \Delta\alpha_0 + \delta_0 \Delta\alpha_1 &= -d_0 \\ \delta_2 \Delta\alpha_0 + \delta_1 \Delta\alpha_1 &= -d_1 \end{aligned} \tag{3.36}$$

and solved to obtain the desired increments[1]. After the iterations converge, the two roots, r_1 and r_2, are obtained from

$$r_{1,2} = 0.5\left(\alpha_1 \pm \sqrt{\alpha_1^2 + 4\alpha_0}\right) \tag{3.37}$$

[1] It can be shown that the coefficient matrix for the set of linear equations in Eq. (3.36) is not singular and, therefore, a solution exists.

Since the coefficients of the quotient (d_j) are easily obtainable using Eq. (3.32), the quadratic factorization may be repeated (till the quotient becomes quadratic or linear) to find all the roots of the polynomial, f(x).

Example 3.11: Find all the roots of the equation

$$x^5 - 5.05000x^4 + 12.20000x^3 - 16.48000x^2 + 12.56440x - 4.28442 = 0$$

using the Bairstow method.

Solution:

j	Iteration 1 α_0= -1 α_1=1		Iteration 2 α_0= -1.174 α_1=1.467		Iteration 3 α_0= -1.582 α_1=1.936		Iteration 4 α_0= -1.986 α_1=2.196		Iteration 5 α_0= -2.018 α_1=2.198		Iteration 6 α_0= -2.02 α_1=2.2	
	d_j	δ_j	d_j	δ_j	d_j	δ_j	d_j	δ_j	d_j	δ_j	d_j	δ_j
0	1.130	-2.096	0.484	-0.283	0.204	0.154	0.010	-0.363	0.001	-0.475	0.000	
1	0.134	0.870	0.202	0.864	0.138	0.603	0.016	0.294	0.001	0.206	0.000	
2	-5.280	3.100	-3.809	1.491	-2.669	0.728	-2.145	0.516	-2.123	0.460	-2.121	
3	7.150	-3.050	5.770	-2.116	4.589	-1.177	3.947	-0.658	3.913	-0.653	3.910	
4	-4.050	1.000	-3.583	1.000	-3.114	1.000	-2.854	1.000	-2.852	1.000	-2.850	
5	1.000		1.000		1.000		1.000		1.000		1.000	
	$\Delta\alpha_0$=-0.174 $\Delta\alpha_1$=0.467		$\Delta\alpha_0$=-0.407 $\Delta\alpha_1$=0.470		$\Delta\alpha_0$=-0.404 $\Delta\alpha_1$=0.260		$\Delta\alpha_0$=-0.032 $\Delta\alpha_1$=0.002		$\Delta\alpha_0$=-0.002 $\Delta\alpha_1$=0.002			

After 5 iterations, the iterations have converged to α_0=–2.02 and α_1=2.2. Using Eq. (3.37), the two roots are obtained as $1.1 \pm 0.9i$. The sixth iteration is performed to get the coefficients in the quotient obtained after dividing the original function, f(x), by (x^2–2.2x+2.02). The *deflated* polynomial is thus obtained as x^3–2.850 x^2+3.910 x – 2.121. Bairstow's method can again be applied to this polynomial, starting with an initial guess (–2,2) as shown below:

j	Iteration 1 α_0= -2 α_1=2		Iteration 2 α_0= -1.867 α_1=1.702		Iteration 3 α_0= -2.016 α_1=1.811		Iteration 4 α_0= -2.020 α_1=1.800		Iteration 5 α_0= -2.02 α_1=1.800	
	d_j	δ_j	d_j	δ_j	d_j	δ_j	d_j	δ_j	d_j	δ_j
0	-0.001	0.510	0.174	-0.837	-0.004	-0.606	0.000	-0.671	0.000	
1	0.210	1.150	0.089	0.553	0.012	0.772	0.000	0.749	0.000	
2	-0.850	1.000	-1.148	1.000	-1.039	1.000	-1.050	1.000	-1.050	
3	1.000		1.000		1.000		1.000		1.000	
	$\Delta\alpha_0$=0.133 $\Delta\alpha_1$=-0.298		$\Delta\alpha_0$=-0.150 $\Delta\alpha_1$=0.110		$\Delta\alpha_0$=-0.003 $\Delta\alpha_1$=-0.011		$\Delta\alpha_0$=-0.0003 $\Delta\alpha_1$=0.0003			

After 4 iterations, the iterations have converged to α_0=–2.02 and α_1=1.8. Using Eq. (3.37), the two roots are obtained as $0.9 \pm 1.1i$. The fifth iteration is performed to get the coefficients in the quotient obtained after dividing the deflated function by (x^2–1.8x+2.02). The deflated polynomial is, obviously, a linear function given by x–1.05. The fifth root is therefore 1.05.

Multiple roots

As we saw in Example 3.5, the otherwise quadratic convergence of Newton-Raphson method was reduced to a mere linear convergence if the desired root is a double root. Let the given function, f(x), have a root (ξ) of multiplicity[1] *m* (i.e., the function value as well as its first *m-1* derivatives are zero at x=ξ). We can then write

$$f(x) = A(x)(x-\xi)^m$$

where A(x) is such that it does not have a zero at ξ. We can show that the Newton-Raphson scheme will be linearly convergent as follows:

$$x^{(i+1)} = x^{(i)} - \frac{f_i}{f_i'} = x^{(i)} - \frac{A\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^m}{mA\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^{m-1} + A'\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^m}$$

from which

$$x^{(i+1)} - \xi = x^{(i)} - \xi - \frac{A\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^m}{mA\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^{m-1} + A'\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)^m}$$

$$= \left(x^{(i)}-\xi\right)\left[1 - \frac{1}{m + A'\left(x^{(i)}\right)\left(x^{(i)}-\xi\right)/A\left(x^{(i)}\right)}\right]$$

Therefore, in the limit, when the iterations converge to the root, we have a linear convergence

$$e^{(i+1)} = \left(1 - \frac{1}{m}\right)e^{(i)}$$

Note that for a double root, the asymptotic error constant is equal to ½, the same as the bisection method. In order to maintain the quadratic convergence behaviour, various modifications have been suggested. These methods aim at transforming the given function, f(x), in such a way that the transformed function has a simple root at ξ.

If we know the multiplicity, *m*, we define a transformed function $\left[f(x)\right]^{1/m}$ and find its root using the Newton-Raphson method. We will get a quadratic convergence since the modified function has a simple root at ξ. The iterative scheme is written as

$$x^{(i+1)} = x^{(i)} - \frac{\left[f_i\right]^{1/m}}{\left[d\left\{f(x)\right\}^{1/m}/dx\right]_{x=x^{(i)}}} = x^{(i)} - m\frac{f_i}{f_i'} \qquad (3.38)$$

Note that the asymptotic error constant which was earlier obtained as $\left|\frac{f''(\xi)}{2f'(\xi)}\right|$ will now be based on the $1/m^{th}$ power of *f*.

Eq. (3.38) will work well but assumes the fact that the multiplicity of the root is known. In some cases, we may know that there is a multiple root, but may not know its multiplicity. We can then use the fact that the derivative of the function will have a root of multiplicity one

---

[1] Recall that *f(x)* has a root, ξ, of multiplicity *m*, if there exists a nonzero real number, *c*, such that $\lim_{x\to\xi}\frac{|f(x)|}{|x-\xi|^m} = c$.

smaller than that of the function and may be written as $f'(x) = B(x)(x-\xi)^{m-1}$. Defining a new function, $f(x)/f'(x)$, it is obvious that it will have a simple root at ξ. The iterative scheme is then written as

$$x^{(i+1)} = x^{(i)} - \frac{f_i / f_i'}{d(f/f')/dx\big|_{x=x^{(i)}}} = x^{(i)} - \frac{f_i f_i'}{(f_i')^2 - f_i f_i''} \tag{3.39}$$

Example 3.12: In Example 3.5 we obtained a root (near x=1) of the following equation

$$x^3 - 1.25020000x^2 - 1.56249999x + 1.95343750 = 0$$

using the Newton-Raphson scheme, and saw that it was linearly convergent. Obtain the root using the modified schemes (Eqs. 3.38 and 3.39).

Solution:

The table below shows the iterations, with m=2, since we know that there is a *double* root.

Iteration no.	$x^{(i)}$	f_i	f_i'	$x^{(i+1)}$ (Eq. 3.38)	ε_r (%)
0	1	0.140738	-1.0629	1.264818	20.93724
1	1.264818	0.000545	0.074243	1.250143	1.173841
2	1.250143	-7.8E-09	0.000216	**1.250216**	0.005778

Note the much faster convergence compared to the unmodified Newton-Raphson method.

Had we not known that the multiplicity of the root is 2, we could use Eq 3.39 as shown below.

Iteration no.	$x^{(i)}$	f_i	f_i'	f_i''	$x^{(i+1)}$ (Eq. 3.39)	ε_r (%)
0	1	0.140738	-1.0629	3.4996	1.23475	19.01193
1	1.23475	0.000585	-0.07605	4.908098	1.250052	1.224103
2	1.250052	-6.7E-09	-0.00024	4.99991	**1.250034**	0.001403

In most cases, however, we will not know about the presence of a multiple root. We could, of course, use Eq. (3.39) in such cases. Though it would compute a simple root also, it would be inefficient because of the additional computation of the second derivative.

Exercise 3.4

1. Find all the roots of the equation $x^4 - 2x^3 - 53x^2 + 54x + 504 = 0$ by the Bairstow's method to an accuracy of 0.01%.
2. Find the multiple root of the equation $(x-0.5236)(0.8660\sin x - 0.5000\cos x) = 0$ near x=0.5. (The root, clearly, is 0.5236). First use the Newton method, then use the modified Newton method given the fact that the multiplicity is 2. Also, assuming that the multiplicity is unknown, use the iterative scheme on the function $f(x)/f'(x)$.

3.5 Design of methods of arbitrary order

The methods discussed in this chapter were broadly divided into bracketing methods and open methods. While the basic methodology is different for these methods, all of the methods generate a sequence of values which (hopefully) converges to the root. In the bracketing method the generated sequence always lies within an interval which contains the desired root and the aim is to reduce this interval to some pre-set small value. In the open methods, the aim is to generate a sequence in such a way that the distance between the root and the terms of the sequence gets progressively smaller. The rate at which the convergence is achieved is important from the point of view of computational efficiency and it would be beneficial to have methods of higher order of convergence. However, as we have seen, the Newton-Raphson method, though of higher order of convergence, may not be as efficient if the effort required in computing the derivative of the function is large. Still, a general technique to generate methods of arbitrary order of convergence would be interesting to study. There are a number of techniques which have been used to achieve a cubic or higher order of convergence. In this section we describe one such technique, proposed by Schroder.

Assuming that the function is smooth (infinitely differentiable) and writing the Taylor's series about $x^{(i)}$, we get

$$f\left(x^{(i)}+h\right)=f_i+\sum_{j=1}^{\infty}\frac{f^{(j)}\left(x^{(i)}\right)}{j!}h^j \tag{3.40}$$

Since the objective is to make the function value equal to 0, we write[1]

$$\frac{f_i}{f_i'}=-h-\frac{1}{f_i'}\sum_{j=2}^{\infty}\frac{f_i^{(j)}}{j!}h^j=-h-\sum_{j=2}^{\infty}d_j h^j \tag{3.41}$$

in which d_j represents the j^{th} derivative divided by the first derivative and factorial j. Using series reversion (see Box 3.4), we may now write an iterative scheme of order p as

$$x^{(i+1)}=x^{(i)}-\sum_{j=1}^{p-1}c_j\left(\frac{f_i}{f_i'}\right)^j \tag{3.42}$$

with $c_1=1;c_2=d_2;c_3=2d_2^2-d_3;\ldots$. Clearly, for quadratic convergence, p=2, c_1=1, and we get the Newton-Raphson iteration (which was shown to have a quadratic convergence). To increase the order of convergence, we simply add more terms. For example, a cubic convergence[2] is achieved by the following:

$$x^{(i+1)}=x^{(i)}-\frac{f_i}{f_i'}-\frac{f_i''}{2f_i'}\left(\frac{f_i}{f_i'}\right)^2 \tag{3.43}$$

---

[1] Assuming that the first derivative of the function does not vanish near the desired root. If it does, i.e., the root is a multiple root, we may transform the function as discussed in the previous section.

[2] It could be seen from Eq. 3.41 that if we truncate the series after one term, j=2, we get a quadratic equation in h, which could be solved to obtain another form of a cubic-convergent method. It is known as the Euler scheme and is given by $x_{i+1}=x_i-\frac{2f_i}{\sqrt{\left(f_i'\right)^2-2f_if_i''}+f_i'}$. Various other forms could be derived by using the fact that near the root $ff''\approx\left(f'\right)^2$ (one of these forms would be identical to Eq. 3.43).

Higher order methods could be written but are not very common unless the higher order derivatives are easy to compute (e.g., a polynomial). It should be noted that additional computational effort is involved in evaluating the derivatives and the overall efficiency of the method may not be as good as a lower order method.

Box 3.4: **Inversion of series**

Given a series

$$y = -d_1 x - d_2 x^2 - \ldots$$

we can compute the coefficients of the inverse series

$$x = c_1 y + c_2 y^2 + \ldots$$

by substituting the inverse series in the forward series and equating the coefficients of various powers of y.

It is easy to show that

$$c_1 = 1/d_1; c_2 = d_2/d_1^3; c_3 = \left(2d_2^2 - d_1 d_3\right)/d_1^5; \ldots$$

from which it is seen that it would simplify the computations if d_1 is taken as 1. That is the reason we have used a division by f_i' in Eq. (3.41).

While Eq. (3.42) does produce a higher order method, it suffers from the drawback that derivatives of order higher than one are required to be computed. Various alternatives are available to attain cubic convergence using the function and its first derivative only. For example, it can be shown that we get a cubic-convergent method if in the Newton-Raphson method we replace the derivative at the point $x^{(i)}$, by an *average* derivative between the points $x^{(i)}$ and $x^{(i+1)}$in one of the following alternative forms:

$$x^{(i+1)} = x^{(i)} - \frac{f_i}{0.5\left(f_i' + f'\left(x^{(i)} - \frac{f_i}{f_i'}\right)\right)} \tag{3.44a}$$

or

$$x^{(i+1)} = x^{(i)} - \frac{f_i}{f'\left(x^{(i)} - 0.5\frac{f_i}{f_i'}\right)} \tag{3.44b}$$

Similarly, cubic convergence is achieved in the Potra-Ptak scheme

$$x^{(i+1)} = x^{(i)} - \frac{f_i + f\left(x^{(i)} - \dfrac{f_i}{f_i'}\right)}{f_i'} \tag{3.45}$$

Any of the methods listed here could be used to obtain a faster convergence at the expense of more function or derivative evaluations. We believe that methods with more than cubic convergence are not useful for a general problem and leave it to the reader, if interested, to derive them using Eq. (3.42).

Exercise 3.5

1. Show that the iteration scheme given by Eq. 3.43 has a cubic order of convergence.
2. Use the iteration scheme of Eq. (3.43) to obtain the root of the function $f(x) = x^3 - 1.2500000x^2 - 1.5625250x + 1.9530938$ near x=–1.5.

3.6 Introduction to system of nonlinear equations

A number of times we encounter a system of equations involving several unknown quantities. If these equations are all linear, we could use the techniques described in the previous chapter. However, if even one out of these equations is nonlinear, those techniques will generally not be applicable. In this section, we introduce the concept of a system of nonlinear equations and discuss the complexities as compared to a single nonlinear equation. The next section describes some of the methods used for solving such systems.

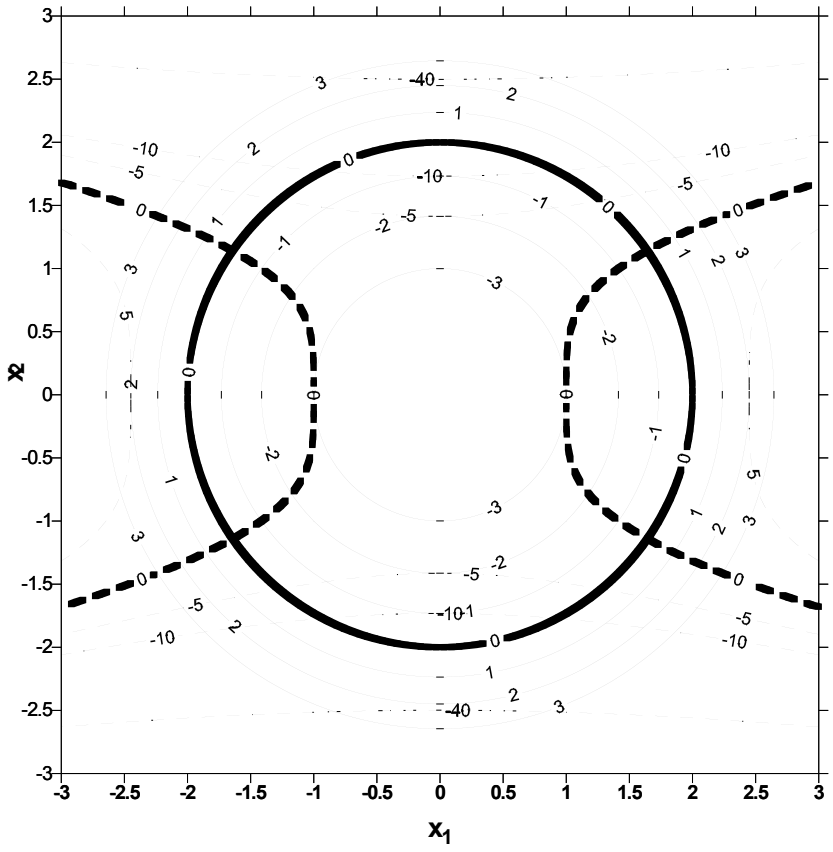
Going back to the battle between Batman and Joker, let us assume that Batman is circling the Vampire State Building keeping a distance of 2 km. We also assume that Joker is following a path such that at all times, he is at a distance of x_1 km east of the building and x_2 km north of the building with $x_1^2 - x_2^4 = 1$. Since Batman wants to catch Joker, he would like to know at which point he should wait so that he is able to complete his task. The problem could be formulated as a set of two nonlinear equations as shown below:

$$\begin{aligned} x_1^2 + x_2^2 = 2^2 \quad &\Rightarrow f_1(x_1, x_2) = x_1^2 + x_2^2 - 4 = 0 \\ x_1^2 - x_2^4 = 1 \quad &\Rightarrow f_2(x_1, x_2) = x_1^2 - x_2^4 - 1 = 0 \end{aligned} \tag{3.46}$$

As we saw for a linear set of equations, these two equations may or may not have a solution. However, the nonlinearity makes it difficult to perform an analysis similar to that done for the linear system. Even if a solution exists, it may not be physically correct (e.g., a complex number in this case or a negative value for, say, mass of an object). We will, therefore, not consider analysing the system to determine the existence or otherwise of a solution and would proceed directly to the solution.

Fig. 3.9 shows a contour plot of the two functions listed in Eq. (3.46). Clearly, the *four* solutions are the points of intersection of the two thick lines representing the contours of zero values. However, in most practical problems, it would not be possible to determine the total number of solutions. On the other hand, for most such problems, we would have a fairly good idea about the approximate location of *a solution* and would want to find only that particular

solution. For some cases, it may be possible to reduce the system of equations to a single nonlinear equation which can then be solved using any of the techniques discussed earlier in this chapter. For example, Eq. (3.46) may be written as $3 - x_2^2 - x_2^4 = 0$, which may be analytically solved to obtain $x_2^2 = \dfrac{-1 \pm \sqrt{13}}{2}$ =1.303 (ignoring the non-physical negative value). Thus $x_2 = \pm 1.141$ and $x_1 = \pm 1.642$, as may be seen in the figure. Again, for a general case, it is not possible to manipulate the given equations into a single equation involving one variable only. We, therefore, need to look at methods which are more generally applicable.



In the previous chapter, we looked at various methods of solving a system of *linear* equations and earlier in this chapter, we discussed the solution of a single nonlinear equation. It would

appear that a combination of these techniques should work for the solution of a system of nonlinear equations in which *n* equations (at least one of them being nonlinear) have to be solved for *n* unknown parameters. However, not all methods which apply to a single nonlinear equation could be extended to a system of equations. For example, taking the simplest case with two unknown quantities (x_1 and x_2) and two equations $\left[f_1\left(x_1,x_2\right)=0, f_2\left(x_1,x_2\right)=0\right]$, it is readily seen that the bracketing methods will NOT work. For a well behaved function of a single variable, there would be a root between two points at which the function has opposite values. However, if we consider the extension to two-dimensions and choose a rectangle in such a way that both the functions f_1 and f_2 show a change of sign within this rectangle, the only thing we are assured of is that zeroes of both f_1 and f_2 will lie within this rectangle. However, there is no guarantee that these *contours* of zero values of f_1 and f_2 will intersect within this rectangle. For example, in Fig. 3.9, if we choose a square spanning 1 to 1.5 in both north and east directions, the function values at the four corners are computed as:

$$x_1 = 1, x_2 = 1 \quad \Rightarrow f_1 = -2, f_2 = -1$$
$$x_1 = 1.5, x_2 = 1 \quad \Rightarrow f_1 = -0.75, f_2 = 0.25$$
$$x_1 = 1.5, x_2 = 1.5 \quad \Rightarrow f_1 = 0.5, f_2 = -3.8125$$
$$x_1 = 1, x_2 = 1.5 \quad \Rightarrow f_1 = -0.75, f_2 = -5.0625$$

Since both f_1 and f_2 show a change of sign, they would have zero contours within this square. However, as we have already computed, the solution of the system of equations does not lie within this square. Thus the bracketing methods would not work on a set of equations since the functions and their zero-value contours are all independent. The open methods, on the other hand, should work and are described in the next section.

3.7 Solution Methods

Out of the various methods described for a single nonlinear equation, the three most commonly used for a system of nonlinear equation are the fixed-point iteration method, the Newton method, and the Secant method. Since the basic philosophy of these methods has already been described in detail, we provide a relatively brief account here. We assume that there are *n* equations in *n* unknown variables and denote these as

$$\begin{aligned} f_1\left(x_1,x_2,...,x_n\right)&=0\\ f_2\left(x_1,x_2,...,x_n\right)&=0\\ &.\\ &.\\ f_n\left(x_1,x_2,...,x_n\right)&=0 \end{aligned} \qquad (3.47)$$

Fixed-point iteration

The set of equations (3.47) may be written as

$$
\begin{aligned}
x_1 &= \phi_1\left(x_1, x_2, ..., x_n\right) \\
x_2 &= \phi_2\left(x_1, x_2, ..., x_n\right) \\
&. \\
&. \\
x_n &= \phi_n\left(x_1, x_2, ..., x_n\right)
\end{aligned}
\tag{3.48}
$$

Using a concise notation with vector $\boldsymbol{x}$ representing $\left(x_1, x_2, ..., x_n\right)^T$, and the vector $\boldsymbol{\phi}$ representing $\left(\phi_1, \phi_2, ..., \phi_n\right)^T$, the sequence of iteration is written as $\boldsymbol{x}^{(i+1)} = \boldsymbol{\phi}(\boldsymbol{x}^{(i)})$ and the iterations are stopped when some norm of the approximate error vector $\boldsymbol{\varepsilon}^{(i+1)} = |\boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)}|$ or the relative approximate error falls below the desired error tolerance (generally the maximum norm is used). We would now look at the conditions under which this method converges.

The true error at iteration *i+1* could be written as (note that the true solution $\boldsymbol{\xi}$ is also a vector)

$$e^{(i+1)} = \xi - x^{(i+1)} = \phi(\xi) - \phi(x^{(i)}) \tag{3.49}$$

As done earlier, we assume that partial derivatives of $\boldsymbol{\phi(x)}$, with respect to all the variables $\boldsymbol{x}$, exist. It is then possible to write the components of the error vector as

$$e_1^{(i+1)} = \left.\frac{\partial \phi_1}{\partial x_1}\right|_{\bar{x}_1} e_1^{(i)} + \left.\frac{\partial \phi_1}{\partial x_2}\right|_{\bar{x}_2} e_2^{(i)} + ... + \left.\frac{\partial \phi_1}{\partial x_n}\right|_{\bar{x}_n} e_n^{(i)} \tag{3.50}$$

where $\bar{x}$ are located in the appropriate intervals. Similar expressions for the other components may be easily written. Clearly, if

$$\left|\frac{\partial \phi_j}{\partial x_1}\right| + \left|\frac{\partial \phi_j}{\partial x_2}\right| + ... \left|\frac{\partial \phi_j}{\partial x_n}\right| < 1 \quad \forall j \text{ from 1 to } n \tag{3.51}$$

the maximum component of the error (i.e., the maximum norm) at any iteration would be smaller than the maximum norm of the error at the previous iteration and convergence is guaranteed. It is also seen that even without this condition being fulfilled, the iterations may converge if the positive and negative errors cancel out in Eq. (3.50). The following example solves the set of equations (3.46) to enable Batman to position himself at an appropriate location.

Example 3.13: Using the fixed-point iteration method, solve the set of equations (3.46) to a maximum relative error of 0.1% starting with an initial guess of (1.5,1.5).

Solution:

The tables below show the iterations using the following scheme:

$$
\begin{aligned}
x_1 &= \sqrt{4 - x_2^2} \\
x_2 &= \left(x_1^2 - 1\right)^{1/4}
\end{aligned}
$$

As described in the iterative solution of a system of linear equations (section 2.?), two different methodologies could be adopted for the iterations: (1) computing all the variables at an iteration using the values of all variables at the previous iteration (similar to Gauss-Jacobi),

or (2) computing all the variables at an iteration using the most recent estimates of all variables (similar to Gauss-Seidel). The first table uses scheme (1) while the second uses scheme (2).

Iteration no.	x_1	x_2	$\phi_1(x_1,x_2)$	$\phi_2(x_1,x_2)$	ε_{1r} (%)	ε_{2r} (%)	max
0	1.5000	1.5000	1.3229	1.0574	-13.3893	-41.8612	41.8612
1	1.3229	1.0574	1.6976	0.9306	22.0754	-13.6219	22.0754
2	1.6976	0.9306	1.7703	1.1713	4.1048	20.5466	20.5466
3	1.7703	1.1713	1.6212	1.2086	-9.2000	3.0929	9.2000
4	1.6212	1.2086	1.5935	1.1296	-1.7368	-6.9975	6.9975
5	1.5935	1.1296	1.6505	1.1138	3.4521	-1.4147	3.4521
6	1.6505	1.1138	1.6611	1.1459	0.6426	2.7951	2.7951
7	1.6611	1.1459	1.6392	1.1517	-1.3378	0.5062	1.3378
8	1.6392	1.1517	1.6351	1.1397	-0.2502	-1.0557	1.0557
9	1.6351	1.1397	1.6435	1.1374	0.5116	-0.1995	0.5116
10	1.6435	1.1374	1.6451	1.1421	0.0955	0.4076	0.4076
11	1.6451	1.1421	1.6419	1.1429	-0.1966	0.0758	0.1966
12	1.6419	1.1429	1.6413	1.1411	-0.0367	-0.1561	0.1561
13	1.6413	1.1411	1.6425	1.1408	0.0754	-0.0292	0.0754

Iteration no.	x_1	x_2	$\phi_1(x_1,x_2)$	$\phi_2(x_1,x_2)$ *	ε_{1r} (%)	ε_{2r} (%)	max
0	1.5000	1.5000	1.3229	0.9306	-13.3893	-61.1855	61.1855
1	1.3229	0.9306	1.7703	1.2086	25.2741	23.0040	25.2741
2	1.7703	1.2086	1.5935	1.1138	-11.0965	-8.5112	11.0965
3	1.5935	1.1138	1.6611	1.1517	4.0725	3.2872	4.0725
4	1.6611	1.1517	1.6351	1.1374	-1.5913	-1.2573	1.5913
5	1.6351	1.1374	1.6451	1.1429	0.6066	0.4831	0.6066
6	1.6451	1.1429	1.6413	1.1408	-0.2334	-0.1853	0.2334
7	1.6413	1.1408	1.6427	1.1416	0.0895	0.0711	0.0895

* the updated value of x_1 (i.e., $\phi_1(x_1,x_2)$ is used to compute $\phi_2(x_1,x_2)$)

The solution obtained is quite close to the exact solution but the second scheme is seen to have a faster convergence. In general, it has been observed to work better and is preferred over the first scheme. If we use a different starting point, the iterations may converge slowly or may not even converge. For example, starting with (1,1), we require 14 iterations in the first scheme and only 6 in the second; and if we start with an initial guess of (2,2), the iterations lead to complex numbers in both schemes.

Due to the similarity of this scheme with the Gauss-Jacobi and Gauss-Seidel methods, and considering that the linear equations may be thought of as a special case of nonlinear equations, we could apply the convergence criteria given by Eq. (3.51) to a system of linear equations and show that a diagonally dominant coefficient matrix would result in convergence of iterations (as already established in section 2.?).

Newton Method

As before, starting from an initial guess, we move along the *tangent* to estimate the location of the zeroes. Expanding the functions about the estimate at the i^{th} iteration and ignoring

higher order terms, the first equation in the set of equations (3.47) is written as (note that the iteration counter is changed to k since we would be using i and j to represent the rows and column of matrices)

$$f_1\left(x^{(k+1)}\right) \approx f_1\left(x^{(k)}\right) + \sum_{j=1}^{n} \left.\frac{\partial f_1}{\partial x_j}\right|_{x^{(k)}} \left(x_j^{(k+1)} - x_j^{(k)}\right) = 0$$

Similar expressions for other functions are written and the following set of *linear* equations is obtained

$$J\left(x^{(k)}\right)\Delta x^{(k+1)} = -f\left(x^{(k)}\right) \tag{3.52}$$

in which J is the Jacobian matrix defined by $J_{i,j} = \dfrac{\partial f_i(x)}{\partial x_j}$, and $\boldsymbol{\Delta x}$ is the vector of desired change in the vector $\boldsymbol{x}^{(k)}$ which is expected to bring the functions $\boldsymbol{f}$ closer to zero. Eqs. (3.52) are solved using any of the techniques described in the previous chapter. The iterations are stopped when the norm of the vector $\boldsymbol{\Delta x}$ becomes less than a pre-specified tolerance[1]. For a small system, inversion of the Jacobian may not be computationally expensive and the iterative scheme may be written as

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k+1)} = x^{(k)} - J^{-1}\left(x^{(k)}\right) f\left(x^{(k)}\right) \tag{3.53}$$

For example, in the 2x2 system of equations related to the Batman problem, we have

$$\begin{Bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{Bmatrix} = \begin{Bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{Bmatrix} - \frac{1}{\left(\dfrac{\partial f_1}{\partial x_1}\dfrac{\partial f_2}{\partial x_2} - \dfrac{\partial f_1}{\partial x_2}\dfrac{\partial f_2}{\partial x_1}\right)_{\left(x_1^{(k)}, x_2^{(k)}\right)}} \left[\begin{matrix} \dfrac{\partial f_2}{\partial x_2} & -\dfrac{\partial f_1}{\partial x_2} \\ -\dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_1}{\partial x_1} \end{matrix}\right]_{\left(x_1^{(k)}, x_2^{(k)}\right)} \begin{Bmatrix} f_1\left(x_1^{(k)}, x_2^{(k)}\right) \\ f_2\left(x_1^{(k)}, x_2^{(k)}\right) \end{Bmatrix} \tag{3.54}$$

However, for most practical cases, the number of variables would be larger and the inversion would not be efficient. If the evaluation of Jacobian is expensive, one may opt for *updating* the Jacobian at every, say, fifth iteration and use an LU decomposition technique to achieve economy (since the same decomposition is used for 5 iteration steps). We will not discuss these issues in this book.

Example 3.14: Using the Newton method, solve the set of equations (3.46) to a maximum relative error of 0.1% starting with an initial guess of (2,2).

Solution:

The table below shows the iterations and the much faster convergence compared to the fixed-point method (Eq. 3.54 has been used to compute the values of Δx_1 and Δx_2):

Iter.	x_1	x_2	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	Δx_1	Δx_2	ε_{1r} (%)	ε_{2r} (%)	max
0	2.0000	2.0000	4.0000	-13.0000	-0.5278	-0.4722	-35.8491	-30.9091	35.8491
1	1.4722	1.5278	0.5015	-4.2806	0.1162	-0.2761	7.3153	-22.0597	22.0597
2	1.5884	1.2517	0.0897	-0.9314	0.0495	-0.0987	3.0230	-8.5591	8.5591
3	1.6379	1.1530	0.0122	-0.0844	0.0043	-0.0114	0.2640	-1.0027	1.0027

[1] Sometimes, the stopping criterion is based on the magnitude of the functions rather than the change in variables since our intent is to bring the function values close to zero.

4	1.6423	1.1415	0.0001	-0.0010	0.0001	-0.0001	0.0032	-0.0124	0.0124
5	1.6423	1.1414	2.3E-08	-1.5E-07					

The Jacobian for the first few iterations is shown below:

$$J^{(0)} = \begin{bmatrix} 4 & 4 \\ 4 & -32 \end{bmatrix} \quad J^{(1)} = \begin{bmatrix} 2.944 & 3.056 \\ 2.944 & -14.26 \end{bmatrix} \quad J^{(2)} = \begin{bmatrix} 3.177 & 2.503 \\ 3.177 & -7.844 \end{bmatrix} \quad J^{(3)} = \begin{bmatrix} 3.276 & 2.306 \\ 3.276 & -6.131 \end{bmatrix}$$

The solution is obtained as (1.6423,1.1414) which is again close to the exact solution and the values of the functions at this point are very close to zero (it should be noted that the table shows truncated values only. The computations were performed in a much higher precision!). For this problem, we see that the starting guess is not very critical to the convergence. Even when starting far away from the solution at the point (10,10), the convergence is achieved in 10 iterations!

Secant Method

Although the Newton method works well for most problems, we do need to evaluate the partial derivatives of the given nonlinear functions. In many cases, the analytical derivatives are not available. Numerical evaluation of the derivatives using finite differences would be computationally expensive. In such cases, the Secant method discussed earlier for a single equation may be extended to a system of equation as follows (since this method was first proposed by Broyden, it is commonly called the Broyden's method).

Given two *points* which are at a *distance* of $\boldsymbol{\Delta x}$, the Jacobian may be approximated by a "Broyden" matrix, $\boldsymbol{B}$, which satisfies $B \cdot \Delta x = \Delta f$ and an iterative scheme is written as (see Eq. 3.52)

$$B^{(k)} \Delta x^{(k+1)} = -f\left(x^{(k)}\right) \tag{3.55}$$

While for a single equation, $\boldsymbol{B}$ was uniquely determined from the approximation of the tangent by the Secant, for a multidimensional case it is not possible[1]. To uniquely determine $\boldsymbol{B}$, Broyden suggested using a constraint that the change in $\boldsymbol{B}$ from its value at the previous iteration should have a minimum norm (see Box 3.5)[2]. For the first iteration, the $\boldsymbol{B}$ matrix may be taken as the analytically or numerically obtained Jacobian. However, if the Jacobian is not easy to evaluate, any approximation (e.g., an identity matrix) may be used. Convergence, however, is likely to be poor if the initial guess is not a good approximation of the Jacobian. The steps of the iterative scheme can then be written as:

---

[1] There are n^2 unknowns and only n equations. For example, if $\Delta x = \begin{Bmatrix} 0.1 \\ 0.1 \end{Bmatrix}$ and $\Delta f = \begin{Bmatrix} 0.1 \\ 0.1 \end{Bmatrix}$, it is rather easy to see that both the matrices $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$ would satisfy the Jacobian approximation.

[2] Another alternative could be to store n previous iterates and use $B \cdot \Delta x = \Delta f$ for each. However, it increases the storage requirement and is not preferred. Moreover, it is likely that the solution vectors at various iterations would be linearly dependent and an accurate solution for $\boldsymbol{B}$ may not be obtainable.

1. Assume a starting point $x^{(0)}$. The physics of the problem would help in choosing an appropriate starting point.
2. Take the ***B*** matrix as the Jacobian at this point $\left[B^{(0)} = J\left(x^{(0)}\right)\right]$ or an approximation to it.
3. Solve Eq. (3.55) to obtain the change in the variables, $\Delta x^{(1)}$, and obtain $x^{(1)}\left(= x^{(0)} + \Delta x^{(1)}\right)$.
4. Repeat for k=1,2…till convergence
    a. Compute an updated B matrix as

$$B^{(k)} = B^{(k-1)} + \frac{\left(\Delta f^{(k)} - B^{(k-1)}\Delta x^{(k)}\right)\left(\Delta x^{(k)}\right)^T}{\left(\Delta x^{(k)}\right)^T \Delta x^{(k)}} \tag{3.56}$$

    in which $\Delta x^{(k)} = x^{(k)} - x^{(k-1)}$ and $\Delta f^{(k)} = f\left(x^{(k)}\right) - f\left(x^{(k-1)}\right)$.

    b. Solve Eq. (3.55) to obtain $\Delta x^{(k+1)}$ and update the variables $\left[x^{(k+1)} = x^{(k)} + \Delta x^{(k+1)}\right]$.

The following example illustrates the use of the Broyden's method.

Box 3.5: **Broyden's matrix**

The iterative sequence for the Broyden matrix is obtained by stipulating that the difference between the matrices at two successive iterations have a minimum Frobenius norm. Let us assume that we have performed iterations up to the level (k−1) and need to obtain $B^{(k)}$. Denoting the difference between the two matrices as

$$\Delta B^{(k)} = B^{(k)} - B^{(k-1)}$$

and noting, from the definition of the Secant, that

$$B^{(k)}\Delta x^{(k)} = \Delta f^{(k)}$$

we have,

$$\Delta B^{(k)}\Delta x^{(k)} = \Delta f^{(k)} - B^{(k-1)}\Delta x^{(k)}$$

This condition is clearly satisfied if we choose (other choices are also possible and lead to different types of methods)

$$\Delta B^{(k)} = \frac{\left(\Delta f^{(k)} - B^{(k-1)}\Delta x^{(k)}\right)\left(\Delta x^{(k)}\right)^T}{\left(\Delta x^{(k)}\right)^T \Delta x^{(k)}}$$

To show that this choice of ΔB results in minimum Frobenius norm, let us assume that there is another matrix, say, Δ', which satisfies the Secant requirement $\left(B^{(k-1)} + \Delta'\right)\Delta x^{(k)} = \Delta f^{(k)}$. We may then write

$$\left\|\Delta B^{(k)}\right\| = \left\| \frac{\left(\Delta f^{(k)} - B^{(k-1)}\Delta x^{(k)}\right)\left(\Delta x^{(k)}\right)^T}{\left(\Delta x^{(k)}\right)^T \Delta x^{(k)}} \right\|_F$$

$$= \left\| \frac{\Delta' \Delta x^{(k)} \left(\Delta x^{(k)}\right)^T}{\left(\Delta x^{(k)}\right)^T \Delta x^{(k)}} \right\|_F$$

$$\leq \left\|\Delta'\right\|_F \left\| \frac{\Delta x^{(k)} \left(\Delta x^{(k)}\right)^T}{\left(\Delta x^{(k)}\right)^T \Delta x^{(k)}} \right\|_F = \left\|\Delta'\right\|_F$$

The last step results from the fact that Frobenius norm (see Chapter 2) of the matrix formed by multiplying a vector, v, by its transpose as vv^T, is equal to v^Tv. Thus out of all possible matrices of change in the Broyden matrix at any iteration, the one given above as ΔB would have the minimum Frobenius norm.

Example 3.15: Using the Secant method, solve the set of equations (3.46) to a maximum relative error of 0.1% starting with an initial guess of (2,2).

Solution:

The table below shows the iterations:

Iter.	x_1	x_2	$f_1(x_1,x_2)$	$f_2(x_1,x_2)$	Δx_1	Δx_2	e_1 (%)	e_2 (%)	max
0	2.0000	2.0000	4.0000	-13.0000	-0.5278	-0.4722	-35.8491	-30.9091	35.8491
1	1.4722	1.5278	0.5015	-4.2806	0.0084	-0.1505	0.5701	-10.9256	10.9256
2	1.4807	1.3773	0.0893	-2.4061	0.0928	-0.1413	5.9004	-11.4292	11.4292
3	1.5735	1.2360	0.0037	-0.8582	0.0597	-0.0732	3.6550	-6.2977	6.2977
4	1.6332	1.1628	0.0195	-0.1609	0.0094	-0.0194	0.5703	-1.7004	1.7004
5	1.6426	1.1434	0.0053	-0.0109	-0.0001	-0.0019	-0.0081	-0.1661	0.1661
6	1.6424	1.1415	0.0005	-0.0001	-0.0001	-0.0001	-0.0062	-0.0061	0.0062
7	1.6423	1.1414	4.39E-05	2.27E-05					

The Broyden matrix for the first few iterations is shown below (note that for the 0^{th} iteration it is the same as the Jacobian):

$$B^{(0)} = \begin{bmatrix} 4 & 4 \\ 4 & -32 \end{bmatrix} \quad B^{(1)} = \begin{bmatrix} 3.472 & 3.528 \\ 8.505 & -27.97 \end{bmatrix} \quad B^{(2)} = \begin{bmatrix} 3.505 & 2.936 \\ 7.610 & -12.03 \end{bmatrix} \quad B^{(3)} = \begin{bmatrix} 3.517 & 2.918 \\ 4.822 & -7.788 \end{bmatrix}$$

The solution is obtained as (1.6423,1.1414) which is again close to the exact solution. Though it takes more iterations than the Newton method, the computational effort per iteration is smaller and, depending on the type of function and number of equations, overall efficiency may be better for the Secant method.

Although we did provide some discussion on convergence for the fixed point iteration scheme, analysis of convergence of the solution methods for a set of nonlinear equations is quite complicated. Without going into the proof, we just state that the Newton's method is quadratically convergent for initial guess close to the exact solution provided the inverse of the Jacobian exists. On the other hand, the Broyden's method has superlinear convergence[1]. Some other methods of solving a system of nonlinear equations are also available and most of these are based on optimization. For example, in the Newton scheme, when we move along the direction of the gradient, we may not take the full Newton step but decide the step size so as to minimize the function value. These methods are, however, a little advanced for this book and are not described here.

Exercise 3.7

1. Two numbers, x and y, are such that $x^x + y^y = 11.72$; $x^y + y^x = 6.71$. Find the numbers by using the fixed point iteration, Newton, and Broyden methods. Use starting guess as x=2, y=2 for the first two methods. For the Broyden method, use two starting points as (1,1) and (2,2).
2. In the previous problem, compare the performance of Broyden's method by using the initial estimate of the Broyden matrix as (a) Jacobian at point (1,1) (b) Jacobian at point (2,2), and (c) Identity matrix.
3. Solve the following equations using the Newton method using an appropriate starting point:

$$xe^x + y\sin y + z\ln z = 6$$
$$x^2y + y^2z + z^2x = 21$$
$$xyz = 7$$

3.8 Summary

Various bracketing and open methods were discussed in this chapter to solve a nonlinear equation in a single variable. Analysis of these methods was performed to obtain the rate of convergence and specific cases like complex roots and multiple roots were described. "Which method to use for a particular problem" is a question which does not have a unique answer. The bracketing methods converge, but very slowly, and the open methods converge faster, if at all. The most common strategy is to start with the bracketing methods and then use one of the open methods to accelerate the convergence.

For the multivariate case, some of the methods discussed for the single variable case were extended. The fixed-point method, Newton method, and the Secant method were discussed. The methods described in Chapter 2 for solving a set of linear simultaneous equations were shown to be useful here since the Newton and Secant iterative schemes for the nonlinear equations involve the solution of a linear system.

---

[1] If $\left\|x^{(k+1)} - \xi\right\| \le c_k \left\|x^{(k)} - \xi\right\|$ such that $\lim_{k\to\infty} c_k = 0$, the iteration scheme is called superlinearly convergent.

The Secant method has already introduced us to approximation of the derivative of a function by using the function values at two different points and assuming the function to be linear. However, this is not the only way to approximate the function or its derivative. For example, in Muller's method, we used three function values to fit a 2^{nd} degree polynomial and obtained its zero to approximate the root of a given equation. There are numerous such applications where we would need to approximate a function and/or its derivatives or integrals. We describe the approximation of functions in the next chapter (Chapter 4) and the approximation of derivatives and integrals in Chapter 5.