3

Iterative Methods

3.1 Introduction

Given a quadratic equation:

$$ax^2 + bx + c = 0 \tag{3.1}$$

We know from school days the *closed form* solution, namely, a formula giving the solution of the equation as a function of the coefficients a, b, and c. The closed form solutions in this case are:

$$x_1 = \frac{(-b + \sqrt{b^2 - 4ac})}{2a}, x_2 = \frac{(-b - \sqrt{b^2 - 4ac})}{2a} \tag{3.2}$$

Similar formulae are available for cubic and quartic polynomial equations but we rarely remember them. For higher order polynomial equations and non-polynomial equations it is difficult, and in many cases impossible, to get closed form solutions. Besides this, when numbers are substituted in (available) closed form solutions, rounding errors often reduce their accuracy.

Another approach to solving equations is by *trial* and *error*. As a technique of solving equations trial and error is very general. It is, however, disliked by a human problem solver. For a computer, on the other hand, it is the most natural technique to use due to the following reasons:

(i) In a trial and error method of solution a sequence of steps is repeated over and over again (that is why it is disliked by the human mind). Such a procedure is concisely expressible as a computational algorithm. It is thus a natural method to use with computers.

(ii) It is possible to formulate, using trial and error, algorithms which tackle a class of similar problems. For instance a general computational algorithm to solve polynomial equations of order n (where n is any integer) may be written.

(iii) Rounding errors are negligible in trial and error procedures compared to procedures based on closed form solutions. The reason for this statement will become evident later in this chapter.

CamScanner

3.2 Beginning an Iterative Method

In this chapter we will discuss a number of trial and error or *iterative methods* of finding solutions of the equation $f(x) = 0$. These solutions are also known as the *roots of the equation* $f(x) = 0$.

There are five basic questions one has to ask which are relevant to all iterative methods. These are:

(i) Where does one start the iterative cycle? In other words, how does one choose the initial guess value of the root?
(ii) How does one proceed from the initial guess to the subsequent approximations?
(iii) How fast do these successive approximations converge to the root?
(iv) How much computational effort is required in each iteration?
(v) When does one stop the iterative cycle?

We will first consider a method called the bisection method which is a very simple and reliable iterative method.

The best way to make an initial guess of a root of $f(x) = 0$ is to sketch x vs $f(x)$. The roots of the equation are the points where the graph of $y = f(x)$ cuts the x-axis. For example, consider the equation:

$$f(x) = x^3 - 2.5x^2 - 2.46x + 3.96 = 0.$$

It is plotted in Figure 3.1 based on Table 3.1.

From the sketch it is seen that this function has three roots. They are

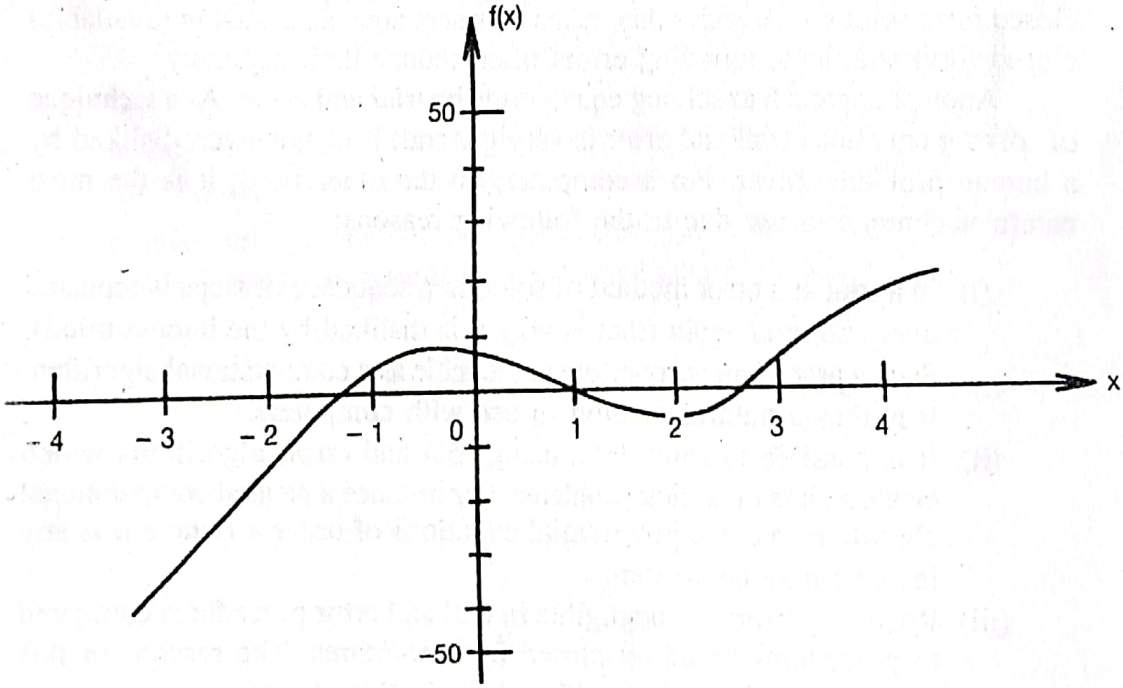


Fig. 3.1 Sketch of the function $f(x) = x^3 - 2.5x^2 - 2.46x + 3.96$.

TABLE 3.1 Table Giving x vs $f(x)$ for Example Function

x	−4	−3	−2	−1	0	1	2	3	4
$f(x)$	−90.2	−38.2	−9.1	2.9	4.0	0	−3	1.1	18.1

approximately −1.5, 1 and 2.5. Thus it is easy to guess an approximate value for a desired root and refine it using an iterative method.

If the function is complicated then we can use the computer itself to print out a table such as Table 3.1 and quickly sketch the curve or plot it on a video terminal. An algorithm to tabulate an arbitrary function is given below.

Let x_{min} and x_{max} be the end points of the interval over which $f(x)$ is to be tabulated. Let Δx be the tabulation interval.

Algorithm 3.1 Procedure to tabulate $f(x)$

1 Read x_{min}, x_{max}, Δx
2 $x \leftarrow x_{min}$
3 *While* $x \leq x_{max}$ *do*
  begin
4 $y \leftarrow f(x)$
5 Write x, y
6 $x \leftarrow x + \Delta x$
  end
7 Stop

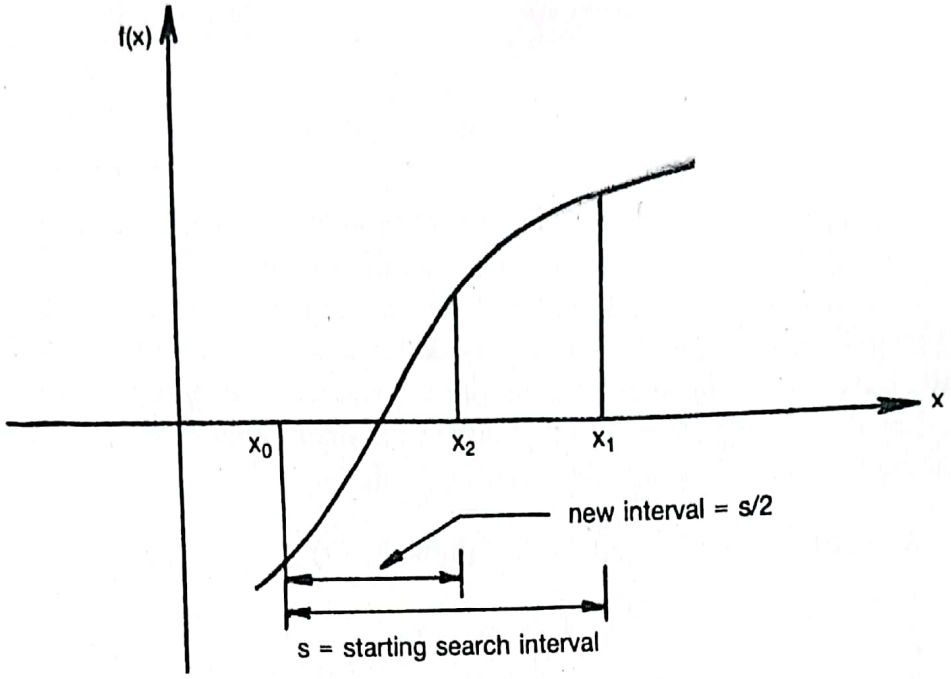
3.3 The Method of Successive Bisection

In this method we begin the iterative cycle by picking two trial points which enclose the root. Two points x_0 and x_1 enclose a root if $f(x_0)$ and $f(x_1)$ are of opposite signs. We now bisect the interval (x_0, x_1) and denote the mid-point by x_2 so that

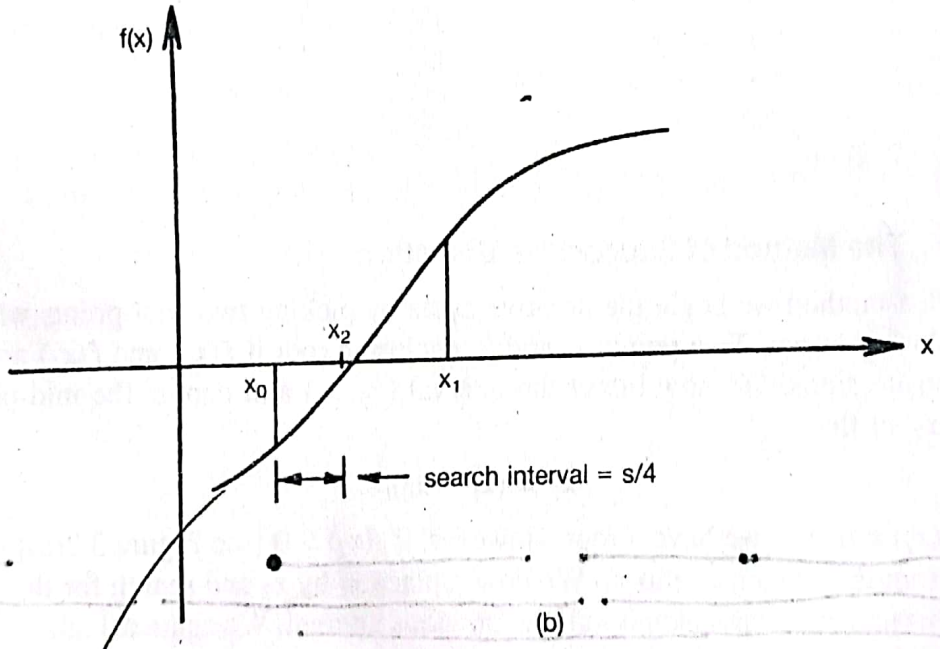
$$x_2 = (x_1 + x_0)/2.$$

If $f(x_2) = 0$ then we have a root. However, if $f(x_2) > 0$ [see Figure 3.2(a)] then the root is between x_0 and x_2. We now replace x_1 by x_2 and search for the root in this new interval which is half the previous interval. We again calculate $f(x_2)$ at the mid point of this new interval [see Figure 3.2(b)]. This time $f(x_2) < 0$. Thus, the root lies between x_2 and x_1. We now replace x_0 by x_2 and again bisect the interval.

By repeating this bisection procedure we always enclose the root in the search interval and the search interval is halved in each iteration. Thus ten steps will reduce the search interval by $2^{10} \simeq 1000$ and twenty steps will reduce the interval size by $2^{20} \simeq 10^6$. (Each iteration leads us closer to the root by one binary digit.) We stop the iterative cycle as soon as the search interval becomes smaller than the precision with which we want the answer.



(a)



(b)

Fig. 3.2(a), (b) Illustrating bisection method.

Example 3.1

We will find the square root of a number by this method. The square root of a number d is evaluated by finding the root of the equation $x^2 - d = f(x) = 0$. Let

$$d = 25.0, \quad x_0 = 2.0, \quad x_1 = 7.0.$$

The iterations are shown in Table 3.2.

TABLE 3.2 Illustrating Bisection Method

Iteration		x_0	x_1	Mid Point x_2	$y_0 = f(x_0)$	$f(x_1)$	$f(x_2)$
	0	2.0000	7.0000	4.5000	−21	24	−4.7500
$x_0 \leftarrow x_2$	1	4.5000	7.0000	5.7500	−4.75	24	8.0625
$x_1 \leftarrow x_2$	2	4.5000	5.7500	5.1250	−4.75	8.0625	1.2656
$x_1 \leftarrow x_2$	3	4.5000	5.1250	4.8125	−4.75	1.2656	−1.8398
$x_0 \leftarrow x_2$	4	4.8125	5.1250	4.9687	−1.8398	1.2656	−.3120
$x_0 \leftarrow x_2$	5	4.9687	5.1250	5.0468	−.3120	1.2656	.4706
$x_1 \leftarrow x_2$	6	4.9687	5.0468	5.0077	−.3120	.4706	.0775
$x_1 \leftarrow x_2$	7	4.9687	5.0077	4.9882	−.3120	.0775	−.0117
$x_0 \leftarrow x_2$	8	4.9882	5.0077	4.9979	−.0117	.0775	−.0204
$x_0 \leftarrow x_2$	9	4.9979	5.0077	5.0028	−.0204	.0775	.0280
$x_1 \leftarrow x_2$	10	4.9979	5.0028	5.0003	−.0204	.0280	.0035

An algorithm has been evolved for the bisection method as Algorithm 3.2.

Algorithm 3.2 Bisection Method to Find Root

1 Read x_0, x_1, e

Remarks: (x_0 and x_1 are two starting values which enclose the desired root, e is the error allowed in the answer) Steps 2, 3 and 4 are called initialisation steps which prepare the ground for the succeeding steps in the algorithm.

2 $y_0 \leftarrow f(x_0)$

3 $y_1 \leftarrow f(x_1)$

4 $i \leftarrow 0$

5 *if* (sign (y_0) = sign (y_1)) *then*
begin Write 'starting values unsuitable'
Write x_0, x_1, y_0, y_1
Stop *end*

6 *While* $|(x_1 - x_0)/x_1| > e$ *do*
begin

7 $x_2 \leftarrow (x_0 + x_1)/2$

8 $y_2 \leftarrow f(x_2)$

9 $i \leftarrow i + 1$

10 *if* (sign (y_0) = sign (y_2)) *then* $x_0 \leftarrow x_2$ *else* $x_1 \leftarrow x_2$
end

11 Write 'solution converges to a root'

12 Write 'Number of iterations = ', i

13 Write x_2, y_2

14 Stop

In Algorithm 3.2, Step 1 reads two starting values which lie on either side of the root along with a value for the allowed error in the solution.

In Steps 2 and 3 the two values of the function at the two starting values are computed. If $f(x)$ is a complicated function a procedure to calculate it (given the argument) may be separately coded. Step 5 checks if the two guess values x_0, x_1, do indeed lie on either side of the root. This step is necessary to prevent the procedure doing unnecessary further work if a careless mistake is committed in picking x_0 and x_1. If a mistake has been made then sign (y_0) = sign (y_1) and an appropriate message is written and the algorithm terminates. Such safety exists are essential in algorithms.

The main procedure begins at Step 6. The earlier Steps 1 to 4 are called *initialisation steps*. Step 6 commands that trial and error consisting of Steps 7 to 10 should continue till the interval in which the root lies is within the 'prescribed precision'. What is a prescribed precision? There are two measures of precisions which may be used. One is absolute precision that is to check if $|x_1 - x_0| < e$. This would be valid if x_1 and x_0 are neither very small nor very large. For example, if the true root is between 1/100 and 100 then absolute error measure is acceptable. However, if the root is very small, say, 10^{-6}, a difference of 10^{-6} between x_0 and x_1 would be unacceptable. In such cases it is appropriate to use a relative error measure or equivalently determine the number of *significant digits* in the approximate root before stopping the iterative process. A measure of relative precision is given by the absolute value of $(x_1 - x_0)/x_1$. If the answer is required with n significant digits then the iterative process is continued till $|(x_1 - x_0)/x_1|$ becomes smaller than or equal to 10^{-n}. In Algorithm 3.2 a relative error measure is used.

Step 7 of the algorithm finds the mid point of the interval and Step 8 calculates the value of the function at the mid point. Step 10 determines which of the two end points x_0 or x_1 is to be moved to the mid point. Observe that y_0 which was set in the initialization step (Step 2) is used without any modification. This may be justified by referring to Figure 3.2(a) and (b). If the mid point x_2 falls to the right of the root then sign $(y_0) \neq$ sign (y_2) and x_1 is 'moved' to x_2; y_0 retains the sign of $f(x_0)$ to the left of the root. If sign (y_0) = sign (y_2) then x_0 and x_2 are to the left of the root. Now x_0 is moved to x_2. The sign of y_0 and the sign of $f(x_2)$ are same. Thus there is no need to recompute y_0 in the iterative cycle.

The algorithm leaves the while loop if $|(x_1 - x_0)/x_1| \leq e$ and in Step 13 the solution is written out. The computation is stopped in Step 14.

We will now summarise the method by answering the five questions raised in Section 3.2 in relation to the bisection method.

(i) We *start* the iterative process by first plotting $f(x)$ and finding two values of x which fall on the opposite sides of a root.

(ii) The iterative cycle *proceeds* by bisecting the interval and by locating the root in one of the halves. The half interval in which the root is situated is further bisected.

CamScanner

(iii) Convergence is *assured* in the bisection method as we always keep the root enclosed between approximations to the root. Each iteration takes us closer to the root by one binary digit.

(iv) Computational effort is normally measured in terms of the number of evaluations of $f(x)$ as this is the most time consuming part in the procedure. In the bisection method one function evaluation is performed in each iteration.

(v) The iterative cycle is *stopped* when the interval size is smaller than the specified precision required in the value of the root.

3.4 The Method of False Position

The bisection method guarantees that the iterative process will converge. It is, however, slow. Thus attempts have been made to speed up bisection method retaining its guaranteed convergence. A method of doing this is called the method of false position or *regula falsi*. We start this procedure also by locating two points x_0 and x_1 where the function has opposite signs. We now connect the two points $f(x_0)$ and $f(x_1)$ by a straight line and find where it cuts the x-axis. Let it cut the x-axis at x_2. We find $f(x_2)$. If $f(x_2)$ and $f(x_0)$ are of opposite signs then we replace x_1 by x_2 and draw a straight line connecting $f(x_2)$ to $f(x_0)$ to find the new intersection point. If $f(x_2)$ and $f(x_0)$ are of the same sign then x_0 is replaced by x_2 and proceed as before. In both cases the new interval of search is smaller than the initial interval and ultimately it is guaranteed to converge to the root.

We will now get an equation to find successive approximations to the root from Figure 3.3.

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \tan\theta = \frac{f(x_1)}{x_1 - x_2} \tag{3.3}$$

$$x_1 - x_2 = \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

or

$$x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)} \tag{3.4}$$

An algorithm to implement the method of false position is given as Algorithm 3.3. In this algorithm we have two new features. Firstly we limit the total number of iterations to n (a number which can be read in as a parameter). Secondly, convergence is checked by checking the value of $f(x)$. If it is nearly zero we declare that the corresponding x is the root. Checking the value of $f(x)$ is acceptable provided the slope of $f(x)$ near the specified root is not too small. If the slope is too small (Figure 3.4) there can be a large error in the root even though $f(x)$ may be small.

CamScanner

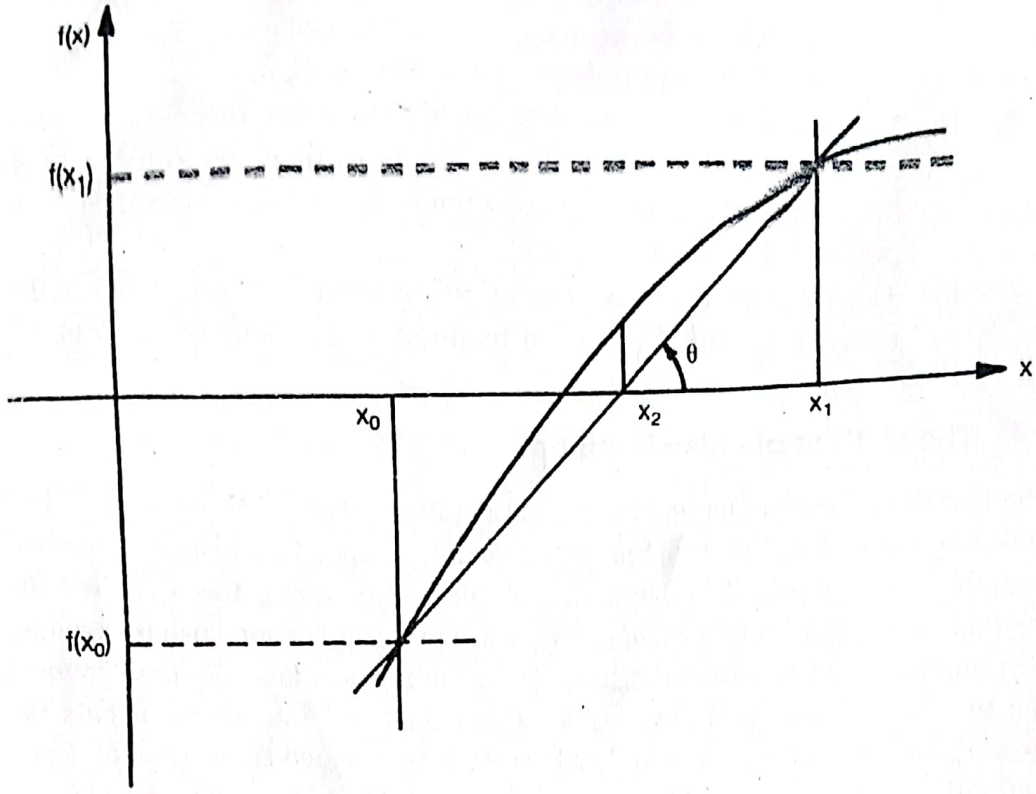


Fig. 3.3 Illustrating false position method.

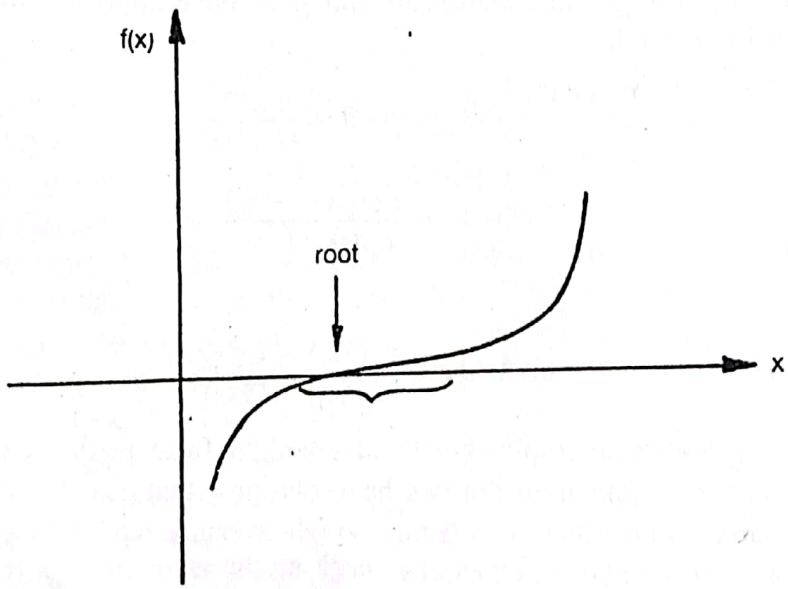


Fig. 3.4 Illustrating that the criterion $|f(x)| \leq e$ may lead to a large error in the root if slope of $f(x)$ is small near the root.

Algorithm 3.3 Method of False Position

1 Read x_0, x_1, e, n

Remarks: x_0 and x_1 are two initial guesses to the root such that sign $(f(x_0)) \neq$ sign $(f(x_1))$. The prescribed precision is e and n is the maximum number of iterations. Steps 2 and 3 are initialization steps.

2 $f_0 \leftarrow f(x_0)$
3 $f_1 \leftarrow f(x_1)$
4 *for* $i = 1$ to n in steps of 1 *do*
5 $\quad x_2 \leftarrow (x_0 f_1 - x_1 f_0)/(f_1 - f_0)$
6 $\quad f_2 \leftarrow f(x_2)$
7 $\quad$ *if* $|f_2| \leq e$ *then*
8 $\quad\quad$ *begin* Write 'convergent solution', x_2, f_2
9 $\quad\quad\quad$ stop *end*
10 $\quad$ *if* sign $(f_2) \neq$ sign (f_0)
11 $\quad\quad$ *then begin* $x_1 \leftarrow x_2$
12 $\quad\quad\quad f_1 \leftarrow f_2$ *end*
13 $\quad\quad$ *else begin* $x_0 \leftarrow x_2$
14 $\quad\quad\quad f_0 \leftarrow f_2$ *end*
endfor
15 Write 'Does not converge in n iterations'
16 Write x_2, f_2
17 Stop

The root of $x^2 - 25 = 0$ is numerically found using Algorithm 3.3 below. Compare this with bisection method to see the speed of convergence to the root.

Example 3.2

$$f(x) = x^2 - 25 = 0.$$

Table 3.3 gives the values of variables obtained by using Algorithm 3.3

TABLE 3.3 False Position Method

Iteration No		x_0	x_1	x_2	f_0	f_1	f_2
Initialisation		2.0	7.0		−21	24	
$x_0 \leftarrow x_2$	1	4.8		4.8	−1.96		−1.96
$x_1 \leftarrow x_2$	2		6.587	6.587		18.38	18.38
$x_0 \leftarrow x_2$	3	4.9721		4.9721	−.2782		−.2782
$x_0 \leftarrow x_2$	4	4.9961		4.9961	−.039		−.039
$x_0 \leftarrow x_2$	5	4.9994		4.9994	−.0053		−.0053
	6			4.9998			−.0014

3.5 Newton-Raphson Iterative Method

Consider the curve $f(x)$ given in Figure 3.5. Starting from an arbitrary point x_0 one may reach the root by using the following procedure.

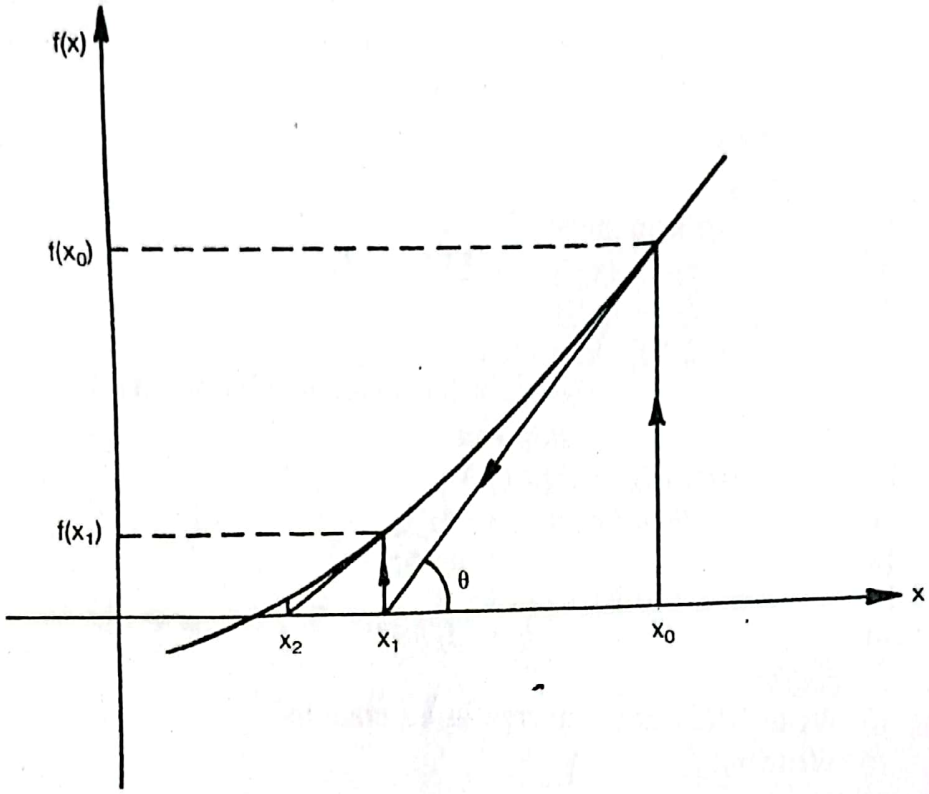


Fig. 3.5 Illustrating Newton-Raphson method.

Step 1: Determine the slope of $f(x)$ at $x = x_0$. Call it $f'(x_0)$.

Step 2: Pick the next approximation x_1 to the root by using the equation

$$\frac{f(x_0)}{(x_0 - x_1)} = \tan\theta = f'(x_0)$$

or

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{3.5}$$

In general proceed from the ith to the $(i + 1)$th approximation using the relation:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{3.6}$$

Note that in writing Equation 3.6 it is assumed that $f'(x_i)$ is never zero for any x_i.

Step 3: Stop the iterative cycle when two successive values of x_i are nearly equal within a prescribed tolerance.

The procedure may be geometrically explained from Figure 3.5 by observing

that a tangent to the curve $f(x)$ is drawn at $x = x_0$ and the point where it cuts the x-axis is taken as the next approximation. The same procedure is repeated till $f(x_i)$ becomes nearly zero which in turn implies that x_{i+1} and x_i are almost equal.

The procedure developed above is called the *Newton-Raphson* iterative method. This method is applicable in general and may be used to find the roots of polynomial and transcendental equations.

The Newton-Raphson formula (Eq. 3.6) may also be derived analytically by using the following Taylor series approximation of the function $f(x)$. If x_i is an approximation to the root then

$$f(x_{i+1}) = f(x_i + h) = f(x_i) + hf'(x_i) + \frac{h^2}{2!}f''(x_i) + \ldots = 0.$$

Neglecting all terms of second and higher degree in h we obtain

$$f(x_i) + hf'(x_i) = 0$$

$$h = -f(x_i)/f'(x_i)$$

$$\text{if} \quad f'(x_i) \neq 0 \text{ for all } x_i$$

$$h = (x_{i+1} - x_i) = -f(x_i)/f'(x_i)$$

$$x_{i+1} = x_i - [f(x_i)/f'(x_i)]$$

which is the same as Eq. (3.6).

Proper choice of the initial guess is very important in the Newton-Raphson method. If the initial choice happens to be near the root the technique converges very fast. If it is not near the root or if one is unfortunate to pick a wrong starting point the procedure may lead to an endless cycle. This is illustrated by the curve given in Figure 3.6. The initial guess x_{01} leads to fast convergence to the correct root. The initial guess x_{02} leads to an endless cycle. The guess x_{03} leads to the trivial root at $x = 0$. The guess x_{04} leads to a divergent solution as $f'(x_{04})$ is very nearly equal to zero. It is thus clear that a bad initial choice will lead one astray and there are many traps if one is careless.

If an algorithm is developed to implement this procedure on a computer it should check to see if $f'(x)$ is too small. This should be done not only at the initially guessed root but right through all the iterative cycles as $f'(x)$ might become small at any of the intermediate points. A 'safety exit' out of the iterative loop should be provided after completing a pre-assigned number of iterations to prevent the program from getting into an endless loop, which happened when x_{02} was chosen as the initial guess, or when the solution diverges, which occurred when x_{04} was chosen as the initial guess.

As the success of the Newton-Raphson procedure depends on the right choice of the initial guess it is essential to quickly sketch $f(x)$ and then decide on the starting point as was pointed out in Section 3.2. Once a preliminary

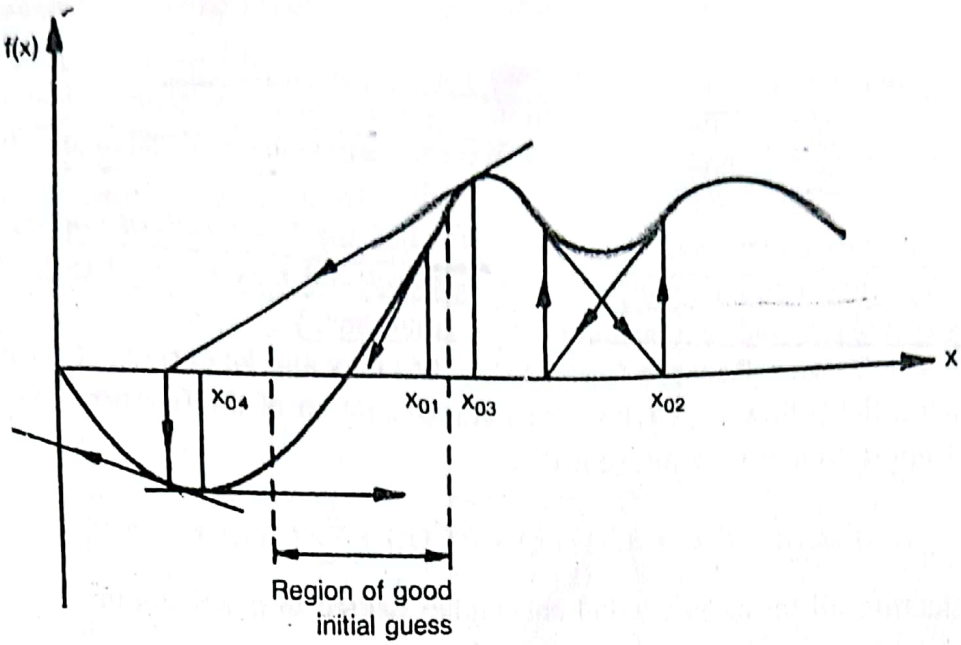


Fig. 3.6 Illustrating traps in the choice of initial guess of root in Newton-Raphson method.

estimate is available then the Newton-Raphson procedure may be safely applied to quickly converge to the root.

3.5.1 *Rate of Convergence of the Iterative Procedure*

Equation 3.6 which is reproduced again below gives the rule to proceed from the ith to the $(i + 1)$th step:

$$x_{i+1} = x_i - [f(x_i)/f'(x_i)]. \tag{3.6}$$

The convergence of the iterative procedure is judged by the rate at which the error between the true root and the calculated root decreases. The *order of convergence* of an iterative process is defined in terms of the errors e_i and e_{i+1} in successive approximations. An iterative algorithm is kth order convergent if k is the largest integer such that:

$$\lim_{i \to \infty} (e_{i+1}/e_i^k) \leq M \tag{3.7}$$

where M is a finite number. In other words the error in any step is proportional to the kth power of the error in the previous step.

The Newton-Raphson procedure is second order convergent as may be seen from the following derivation:

Let y be the root of the equation $f(x) = 0$. Let the ith iterate x_i differ from the true root by an amount e_i. Thus e_i is the error in the ith iteration. Equation 3.6 may be rewritten as:

$$y + e_{i+1} = y + e_i - [f(y + e_i)/f'(y + e_i)].$$

By expanding $f(y + e_i)$ and $f'(y + e_i)$ in a Taylor series around y and simplifying we get

$$e_{i+1} = e_i - \frac{f(y) + e_i f'(y) + (e_i^2/2) f''(y) + \ldots}{f'(y) + e_i f''(y) + (e_i^2/2) f'''(y) + \ldots}. \tag{3.8}$$

By definition $f(y) = 0$. Using this and dividing the numerator and denominator by $f'(y)$ with the assumption that $f'(y) \neq 0$ we obtain

$$e_{i+1} = e_i - \frac{[e_i + \frac{1}{2} e_i^2 f''(y)/f'(y) + (e_i^3/3!) f'''(y)/f'(y) + \ldots]}{[1 + e_i f''(y)/f'(y) + (e_i^2/2) f'''(y)/f'(y) + \ldots]}$$

If $f''(y)/f'(y)$ is finite and e_i small so that e_i^2, e_i^3, etc., may be neglected in comparison with e_i, we obtain:

$$e_{i+1} \simeq e_i - e_i\left[1 + \frac{e_i}{2} f''(y)/f'(y)\right][1 - e_i f''(y)/f'(y)]$$

$$e_{i+1} = \frac{e_i^2}{2} \frac{f''(y)}{f'(y)} \tag{3.9}$$

Thus $$\lim_{i \to \infty} \frac{e_{i+1}}{e_i^2} \to \frac{f''(y)}{2f'(y)}. \tag{3.10}$$

Thus if $[f''(y)/f'(y)]$ is finite and not zero then the Newton-Raphson method is second order convergent.

Physically second order convergence means that in each iteration (as we approach the root) the number of significant digits in the approximation doubles. If the error $|e_i| \leq 10^{-n}$, i.e., the value of x is good to n significant digits (refer Section 2.6) then

$$|e_{i+1}| < \left|\frac{f''}{2f'}\right| e_i^2 < \left|\frac{f''}{2f'}\right| 10^{-2n}.$$

This formula assumes that $|f''/2f'|$ is finite.

3.5.2 *Computer Implementation of the Newton-Raphson Procedure*

In writing a procedure for the computer to implement the Newton-Raphson algorithm, care must be taken to provide for all possible contingencies. To prevent the program getting stuck in a loop an alternative exit after completing a pre-assigned number of cycles must be provided. Further, if $f'(x)$ becomes small anytime during the iterative process it may not converge. Some check should thus be made on this. A computer oriented procedure is described below for implementing the Newton-Raphson technique.

CamScanner

Algorithm 3.4 Newton-Raphson Method

1 Read x_0, epsilon, delta, n

Remarks: x_0 is the initial guess, epsilon is the prescribed relative error, delta is the prescribed lower bound for f' and n the maximum number of iterations to be allowed. Statements 3 to 8 are repeated until the procedure converges to a root or iterations equal n.

2 *for* $i = 1$ to n in steps of 1 *do*
3 $\quad f_0 \leftarrow f(x_0)$
4 $\quad f_0' \leftarrow f'(x_0)$
5 $\quad$ *if* $|f_0'| \le$ delta *then* GOTO 11
6 $\quad x_1 \leftarrow x_0 - (f_0/f_0')$
7 $\quad$ *if* $|(x_1 - x_0)/x_1| <$ epsilon *then* GOTO 13
8 $\quad x_0 \leftarrow x_1$
endfor
9 Write 'Does not converge in n iterations', f_0, f_0', x_0, x_1
10 Stop
11 Write 'Slope too small' x_0, f_0, f_0', i
12 Stop
13 Write 'convergent solution', $x_1, f(x_1), i$
14 Stop

Example 3.3

We will again solve the equation $f(x) = x^2 - d = 0$ to illustrate Newton-Raphson iteration method. We will follow Algorithm 3.4.

$$f(x) = x^2 - 25 = 0; \; f'(x) = 2x$$

Starting value $x = 7$.

The iterations are shown in Table 3.4.

TABLE 3.4 Illustrating Newton-Raphson Method

Iteration Number	x_0	x_1	$f(x_0)$	$f'(x_0)$	$\left\lvert \frac{x_1 - x_0}{x_1} \right\rvert$
1	7	5.2857	24	14	.3242
2	5.2857	5.0070	2.9386	10.5714	.0527
3	5.0070	5.0000	0.0700	10.014	.0013
4	5.0000	5.0000	0	10	0

Observe that compared to the bisection and false position methods Newton-Raphson method converges faster. Further we need only one starting value as opposed to two starting values needed in the other two methods.

CamScanner

3.6 The Secant Method

A serious disadvantage of the Newton-Raphson method is the need to calculate $f'(x)$ in each iteration. Many situations arise in which the expression $f'(x)$ is long and much computer time is needed to evaluate it. There are other situations where a closed form expression for $f'(x)$ is not available. Thus numerical methods which do not need the evaluation of $f'(x)$ have been sought. One of these is the secant method. In this method $f'(x)$ is approximated by the expression:

$$f'(x) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{3.11}$$

where x_i and x_{i-1} are two approximations to the root. The iterative formula to go from the ith to the $(i + 1)$th approximation is given by (Refer to Eq. 3.6)

$$x_{i+1} = x_i - \frac{f(x_i)}{(f(x_i) - f(x_{i-1}))/(x_i - x_{i-1})}. \tag{3.12}$$

A geometrical interpretation of the iterative method is shown in Figure 3.7. Observe that in this method a secant is drawn connecting $f(x_{i-1})$ and $f(x_i)$. The point where it cuts the x-axis is x_{i+1}. Another secant is drawn through $f(x_i)$ and $f(x_{i+1})$ to get x_{i+2} (Figure 3.7). This is repeated till $f(x) \simeq 0$. For computation the formula of Eq. 3.12 is not suitable as it involves division by a small

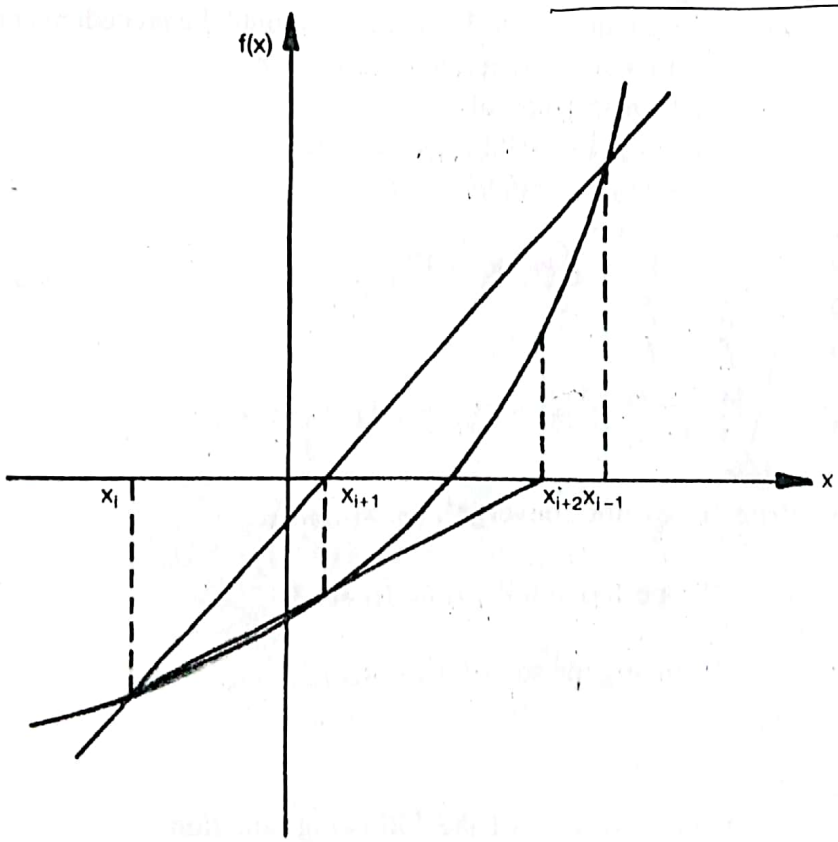


Fig. 3.7 Illustrating the secant method.

quantity which would lead to loss of significant digits. An iterative formula suitable for computation is:

$$x_{i+1} = \frac{x_{i-1}f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})}. \tag{3.13}$$

An algorithm to implement the secant method is given as Algorithm 3.5. The student should note the exit provided when $(f(x_i) - f(x_{i-1}))$ is too small. The criterion used to check convergence is to check if $f(x)$ is small. This criterion is more appropriate as in this method successive values of x_i's tend to fall on either side of the root when a root is approached and checking the value of $f(x_0)$ might lead to an earlier end of the iterative cycle. As in the Newton-Raphson method a limit is put on the number of iterations to be allowed.

Algorithm 3.5 Secant Method

1 Read x_0, x_1, e, delta, n
Remarks: $x_0 x_1$ are two initial guesses to the root. e is the prescribed precision, delta the minimum allowed value of slope and n the maximum number of iterations to be permitted.
2 $f_0 \leftarrow f(x_0)$
3 $f_1 \leftarrow f(x_1)$
Remarks: Statements 5 to 12 are repeated until the procedure converges to a root or iterations exceed n
4 *for* $i = 1$ to n in steps of 1 *do*
5 *if* $|f_1 - f_0| <$ delta *then* GOTO 15
6 $x_2 \leftarrow (x_0 f_1 - x_1 f_0)/(f_1 - f_0)$
7 $f_2 \leftarrow f(x_2)$
8 *if* $|f_2| < e$ *then* GOTO 17
9 $f_0 \leftarrow f_1$
10 $f_1 \leftarrow f_2$
11 $x_0 \leftarrow x_1$
12 $x_1 \leftarrow x_2$
endfor
13 Write 'Does not converge', x_0, x_1, f_0, f_1
14 Stop
15 Write 'Slope too small', i, f_0, f_1, x_0, x_1
16 Stop
17 Write 'Convergent solution', i, x_2, f_2
18 Stop

Example 3.4
Find the smallest positive root of the following equation

$$f(x) = x^3 - 3x^2 + x + 1 = 0$$

A quick plot gives

x	-1	0	2	3	4
$f(x)$	-2	1	-1	4	21

It is evident that one root is between $(-1, 0)$, another between 0 and 2 and the third root between 2 and 3.

For finding the smallest positive root we start with two trial points $x_0 = 0$, $x_1 = 1.5$. We now find the root using Algorithm 3.5.

Iteration	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
x_0	0	1.5	0.7059	0.9361	0.9757
f_0	1	−1.125	0.4592	0.0677	0.0484
x_1	1.5	0.7059	0.9361	0.9757	1.0777
f_1	−1.125	0.4592	0.0677	0.0484	−0.1550
x_2	0.7059	0.9361	0.9757	1.0777	1.000
f_2	0.4592	0.0677	0.0484	−0.1550	0

Observe that the solution is obtained in 5 iterations.

The order of convergence of the secant method will now be derived. The iterative formula is Eq. (3.13)

$$x_{i+1} = \frac{x_{i-1} f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})} \tag{3.13}$$

Let y be a root of $f(x) = 0$. Let $e_{i+1} = x_{i+1} - y$, $e_i = x_i - y$ and $e_{i-1} = x_{i-1} - y$. Expanding $f(x_i) = f(y + e_i)$ and $f(x_{i-1}) = f(y + e_{i-1})$ around y we obtain:

$$y + e_{i+1} = \left[(y + e_{i-1}) \left\{ f(y) + e_i f'(y) + \frac{e_i^2}{2} f''(y) \right\} - (y + e_i) \left\{ f(y) + e_{i-1} f'(y) + \frac{e_{i-1}^2}{2} f''(y) \right\} \right] \Big/ \left[f(y) + e_i f'(y) + \frac{e_i^2}{2} f''(y) - f(y) - e_{i-1} f'(y) - \frac{e_{i-1}^2}{2} f''(y) \right].$$

or

$$e_{i+1} = \frac{(e_{i-1} e_i^2 - e_i e_{i-1}^2) f''(y)}{2(e_i - e_{i-1}) f'(y)}$$

$$= e_i e_{i-1} \frac{f''(y)}{2 f'(y)}. \tag{3.14}$$

In order to find the order of convergence it is necessary to find a formula of the type

$$e_{i+1} = Ke_i^m \tag{3.15}$$

with an appropriate value of m.

If $$e_{i+1} = Ke_i^m$$

then $$e_i = Ke_{i-1}^m$$

or $$e_{i-1} = (e_i/K)^{1/m}. \tag{3.16}$$

Substituting Eq. (3.16) in Eq. (3.14)

$$e_{i+1} = e_i e_i^{1/m} \frac{f''(y)}{2f'(y)K^{1/m}}. \tag{3.17}$$

Comparing Eq. (3.17) with Eq. (3.15) we see that:

$$e_i^m = e_i^{(1+1/m)} \tag{3.18}$$

and

$$K = \frac{f''(y)}{2f'(y)K^{1/m}}. \tag{3.19}$$

Solving for m from Eq. (3.18) we get

$$m = 1 + \frac{1}{m} \text{ or } m^2 - m - 1 = 0$$

which gives $m = 1.62$. Thus

$$e_{i+1} = \left(\frac{f''(y)}{2f'(y)}\right)^{.62} e_i^{1.62}. \tag{3.20}$$

If $[f''(y)/2f'(y)]$ is small then the iterative method converges. The order of convergence as defined in Sec. 3.5.1 is 1.62 which is smaller than in the Newton-Raphson method. Thus this method converges at a slower rate. However, the function is evaluated only once in each iteration as compared to two evaluations (namely $f(x_0)$ and $f'(x_0)$) per iteration in Newton-Raphson method. On the average it is found that secant method is more efficient compared to Newton-Raphson method.

It is important to note that in all iterative methods rounding errors do not accumulate as the number of iterations increase. The total rounding error equals the error in the last iteration. The previous rounding errors are of no consequence. This is due to the fact that in each iteration x_i may be thought of as the starting value from which x_{i+1} is obtained. Thus the rounding error in the final result x_n is only due to the arithmetic operations which produced it from x_{n-1}.

CamScanner

3.7 The Method of Successive Approximations

If the equation $f(x) = 0$ whose roots are to be found is expressed as $x = g(x)$ then an iterative method which is very easy to program for a computer may be formulated. For example the equation

$$f(x) = e^{-x} - 2x + 1 = 0$$

may be rewritten as

$$x = (1 + e^{-x})/2. \tag{3.21}$$

The roots of the equation $f(x) = 0$ are the same as the points of intersection of the straight line representing x and the function $g(x)$. This is illustrated in Figure 3.8.

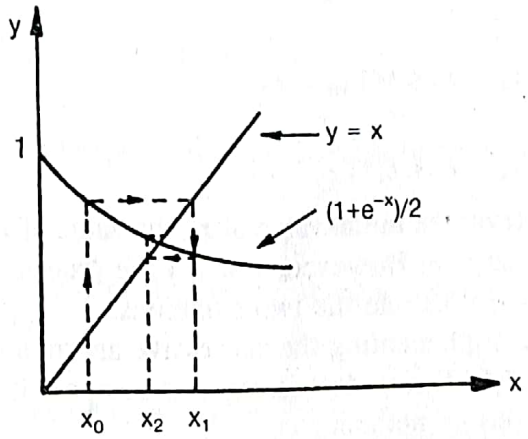


Fig. 3.8 Illustrating the successive approximation method.

The iterative technique is to guess a starting value x_0. Substitute it in $g(x)$. Take $g(x_0)$ as the next guess till two successive values of x are 'close enough'. In other words the iterative formula is:

$$x_1 = g(x_0),\ x_2 = g(x_1),\ \ldots,\ x_{i+1} = g(x_i),\ \ldots,\ x_n = g(x_{n-1}). \tag{3.22}$$

It may be shown that this procedure will converge if $|g'(x_i)| < 1$ by the following argument.

Let r be the root. Thus for $x = r$ we have $g(r) = r$.

If x_n is the nth approximation to the root then

$$x_n - r = g(x_{n-1}) - g(r). \tag{3.23}$$

Dividing both sides of equation (3.23) by $(x_{n-1} - r)$ we get:

$$\frac{x_n - r}{x_{n-1} - r} = \frac{g(x_{n-1}) - g(r)}{(x_{n-1} - r)}. \tag{3.24}$$

By the mean value theorem[1]

$$\frac{g(x_{n-1}) - g(r)}{(x_{n-1} - r)} = g'(\lambda) \tag{3.25}$$

where λ is a point between x_{n-1} and r.

Thus

$$(x_n - r) = g'(\lambda)(x_{n-1} - r). \tag{3.26}$$

Let M be the maximum value of $g'(x)$ occurring at $x = \lambda$ in the entire range $[x_0, r]$.

Then we have

$$|x_n - r| \le |g'(\lambda)|\,|x_{n-1} - r| \le M\,|x_{n-1} - r|$$
$$|x_{n-1} - r| \le M\,|x_{n-2} - r|$$
$$\vdots$$
$$|x_1 - r| \le M\,|x_0 - r|.$$

Thus

$$|x_n - r| \le M^n\,|x_0 - r|. \tag{3.27}$$

Thus if $M < 1$, whatever the initial value of x_0, the value of x_n will tend towards r for a sufficiently large n. However, if $M \ge 1$ the process will not converge. Figure 3.9(a) and (b) illustrate the two situations.

An algorithm implementing the successive approximation procedure is given as algorithm 3.6. The student is urged to compare it with the algorithm for the Newton-Raphson method.

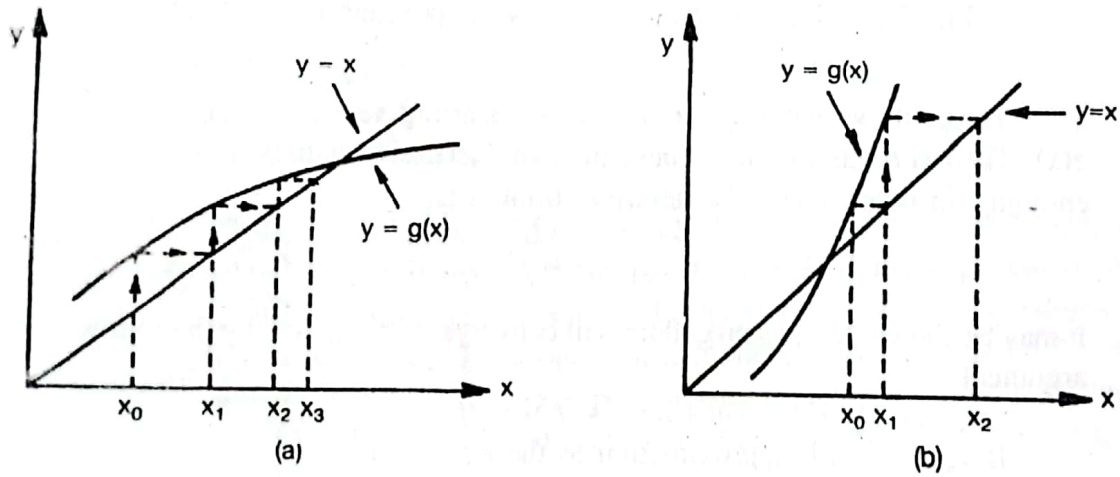


Fig. 3.9 Illustrating (a) convergent and (b) divergent solution in successive approximation method.

[1]Mean value theorem states that given a continuous function $g(x)$ with a continuous first derivative the slope of the chord to the curve drawn between two arbitrary points a, b on it, namely, $\frac{g(b) - g(a)}{(b - a)}$ is equal to the slope of a tangent to the curve at some point between a and b.

Algorithm 3.6 Successive Approximation Method

1 Read x_0, e, n
Remarks: x_0 is the initial guess, e is the allowed error in the root and n the total iterations to be allowed for convergence.
2 $x_1 \leftarrow g(x_0)$
Remarks: Steps 4 to 6 are repeated until the procedure converges to a root or iterations reach n.
3 *for* $i = 1$ to n in steps of 1 *do*
4 $\quad x_0 \leftarrow x_1$
5 $\quad x_1 \leftarrow g(x_0)$
6 $\quad$ *if* $|(x_1 - x_0)/x_1| \le e$ *then* GOTO 9
endfor
7 Write 'Does not converge to a root', x_0, x_1
8 Stop
9 Write 'Converges to a root', i, x_1,
10 Stop

The successive approximation method is first order convergent as shown below. From Eq. (3.22)

$$x_{i+1} = g(x_i).$$

Let y be the root, e_i the error in the ith iteration, and e_{i+1} the error in the $(i + 1)$th iteration. Then

$$y + e_{i+1} = g(y + e_i) = g(y) + \left.\frac{dg}{dx}\right|_y e_i + \left.\frac{d^2g}{dx^2}\right|_y \frac{e_i^2}{2} + \ldots$$

or

$$e_{i+1} = \left.\frac{dg}{dx}\right|_y e_i + (g(y) - y) + 0(e_i^2)$$

where $0\,(e_i^2)$ means terms of the order of e_i^2 or higher powers of e_i. As $g(y) = y$ we have

$$\lim_{i \to \infty} \left| \frac{e_{i+1}}{e_i} \right| = \text{constant} = \left.\frac{dg}{dx}\right|_y$$

3.8 Comparison of Iterative Methods

The main basis of comparison of the various iterative methods presented in this chapter are:

(i) The rate of convergence
(ii) The amount of computational effort per iteration
(iii) Sensitivity of the method to the starting value and the intermediate values.

CamScanner

Of the methods presented the Newton-Raphson method has the fastest convergence. It is an excellent method if one is already near the root. It is, however, quite sensitive as it may diverge if $f'(x)$ is near zero anytime during the iterative cycle. Besides this the amount of computational effort required in computing the values of $f(x)$ and $f'(x)$ is considerable and the overall computing time may be large in spite of the inherently high rate of convergence.

The bisection method converges slowly. It is, however, the simplest iterative method and is guaranteed to converge. Convergence is slow and steady.

The false position method is guaranteed to converge. It is, however, slow as it is first order convergent. The amount of computational effort per iteration is one function evaluation. It is not necessarily a monotonic convergence to the root but most often it will be superior to bisection method.

The secant method is not guaranteed to converge. It, however, converges faster than the false position method; the order of convergence is 1.62. It also computes the given function only once per iteration. It may be considered the most economical method giving reasonably rapid convergence at a low cost.

The successive approximation method is suitable in some special cases. It is easy to program and is first order convergent. It is not guaranteed to converge.

Table 3.5 gives a summary of the methods.

TABLE 3.5 Comparison of Iterative Methods

Method	*Iterative formulae*	*Order of convergence*	*Evaluation of functions per iteration*	*Reliability of convergence*
1 Bisection	$x_{i+1} = \frac{(x_i + x_{i-1})}{2}$ x_i and x_{i-1} enclose root	Gain of one bit per iteration	1	Guaranteed convergence.
2 False Position	$x_{i+1} = \frac{(x_{i-1}f_i - x_i f_{i-1})}{(f_i - f_{i-1})}$ x_i and x_{i-1} enclose root	1	1	Guaranteed convergence.
3 Newton-Raphson	$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$	2	2	Sensitive to starting value. Convergence fast if starting point near the root.
4 Secant	$x_{i+1} = \frac{(x_{i-1}f_i - x_i f_{i-1})}{(f_i - f_{i-1})}$ x_i, x_{i-1} need not enclose root	1.62	1	No guarantee of convergence if not near root. Economical method.
5 Successive approximation	$x_i = g(x_{i-1})$	1	1	Easy to program. No guarantee of convergence.

From this table it may be seen that a good practical way to solve nonlinear equation is to follow the steps below.

(i) Plot $f(x)$ and find the approximate location of root.
(ii) Use two initial guesses to enclose the required root.
(iii) Start iterations using bisection procedure.
(iv) When a root is neared, switch to Newton-Raphson method, if the formula for $f'(x)$ is not complicated. If $f'(x)$ is difficult to compute switch to secant method.

3.9 Solution of Polynomial Equation

A polynomial equation

$$a_0 + a_1x + a_2x^2 + \ldots + a_nx^n = 0 \tag{3.28}$$

is a particular case of the general equation

$$f(x) = 0$$

We treat it separately as it belongs to an important class of equations of great practical interest. For polynomial equations we have more information available regarding the roots than for a general equation. If $x_1, x_2, \ldots, x_n$ are the n roots of Eq. (3.28) we know that

$$\Sigma_{i=1}^{n} x_i = -(a_{n-1}/a_n) \tag{3.29}$$

$$\sum_{i,j=1, i\neq j}^{n} x_ix_j = a_{n-2}/a_n \tag{3.30}$$

and

$$\prod_{i=1}^{n} x_i = (-1)^n a_o/a_n \tag{3.31}$$

From these we also know that for any root x_k

$$\begin{aligned} x_k^2 &\leq x_1^2 + x_2^2 + \ldots + x_n^2 \\ &= (x_1 + x_2 + \ldots + x_n)^2 - 2(x_1x_2 + x_1x_3 + \ldots) \\ &= \left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right) \end{aligned} \tag{3.32}$$

Hence every root x_k is such that

$$|x_k| \leq \sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)} \tag{3.33}$$

CamScanner

This gives an upper bound for all the roots of the polynomial equation and we do not need to search for a root beyond this interval

$$-\sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)}, +\sqrt{\left(\frac{a_{n-1}}{a_n}\right)^2 - 2\left(\frac{a_{n-2}}{a_n}\right)} \tag{3.34}$$

The general strategy to find all the roots of the equation are:

1. Examine if all the coefficients are real. In this case all the roots of the equation are real or pairs of roots are complex conjugates.
2. Find the range of roots from Eq. (3.34). The function $f(x)$ is now plotted in this range. We should take care not to use too coarse an interval lest we miss some roots. In Figure 3.10, for example, root r_1 will be detected as $f(x)$ changes sign between successive plot points. Roots r_2 and r_3 will be missed as they lie within a plot interval. We can simultaneously plot $f'(x)$ which is anyway required. The change of sign of $f'(x)$ will warn of possible roots in an interval.
3. Having guessed the roots from the plots we can count them and see if all roots are accounted for (An nth order polynomial must have n roots).
4. If the number of roots counted is less than the order of the polynomial there could be multiple roots or complex pairs of roots. They have to be found after all the simple real roots are found.
5. The largest root r is found by using Newton-Raphson procedure which we explain in detail later in this section. The polynomial is divided by $(x - r)$. (The division algorithm was given as Algorithm 1.5 in Chapter 1). The reduced polynomial is used to find the next root and remove it. The procedure of reducing the order of a polynomial is known as *deflation*. When all simple roots are removed complex conjugate pairs and repeated roots are tackled.

We will first give the Newton-Raphson method for a polynomial equation to find simple real roots. We assume that the polynomial has real coefficients and all roots are real. One may wonder why yet another method is given for polynomials when Newton-Raphson method is applicable for any function. The main reason is efficiency. The computational labour is reduced with the special algorithm we will present now.

A polynomial equation may be written as:

$$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n = 0 \tag{3.35}$$

We may rewrite

$$f(x) = g(x)(x - x_0) + R \tag{3.36}$$

where $g(x)$ is a $(n - 1)$th order polynomial obtained by dividing $f(x)$ by

$(x - x_0)$ and R is a constant which is the remainder after division. Differentiating Eq. (3.36):

$$f'(x) = g'(x)(x - x_0) + g(x) \tag{3.37}$$

$$f'(x_0) = g(x_0) \tag{3.38}$$

From Example 1.5 of Chapter 1 we know that

$$g(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1} \tag{3.39}$$

which is the polynomial quotient obtained by dividing $f(x)$ by $(x - x_0)$. The coefficients of the polynomial, $b_{n-1}, b_{n-2} \ldots b_0$ as derived in Example 1.5 are given by:

$$\begin{aligned} b_{n-1} &= a_n \\ b_{n-2} &= a_{n-1} + x_0 b_{n-1} \\ b_{n-3} &= a_{n-2} + x_0 b_{n-2} \\ b_{n-4} &= a_{n-3} + x_0 b_{n-3} \end{aligned} \tag{3.40}$$

and finally

$$b_0 = a_1 + x_0 b_1 \tag{3.41}$$

Also observe that Eq. (3.36) may be written as:

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + \ldots + a_nx^n \\ &= (b_0 + b_1x + b_2x^2 + \ldots + b_{n-1}x^{n-1})(x - x_0) + R = 0 \end{aligned} \tag{3.42}$$

Equating constant terms on both sides of Eq. (3.42) we get

$$a_0 = -b_0x_0 + R \tag{3.43}$$

$$R = a_0 + b_0x_0 \tag{3.44}$$

From Eq. (3.36):

$$f(x_0) = R$$

Substituting the value of R from Eq. (3.44) we get:

$$f(x_0) = a_0 + b_0x_0 \tag{3.45}$$

Also observe that

$$f'(x_0) = g(x_0) = b_0 + b_1x_0 + b_2x_0^2 + \ldots + b_{n-1}x_0^{n-1} \tag{3.46}$$

which is expressed using Horner's rule as

$$g(x_0) = b_0 + x_0(b_1 + x_0(b_2 + \ldots x_0(b_{n-2} + x_0b_{n-1}))) \tag{3.47}$$

Using Newton-Raphson technique for finding the root

$$x_{i+1} \leftarrow x_i - \frac{f(x_i)}{f'(x_i)} = x_i - \frac{f(x_i)}{g(x_i)} \tag{3.48}$$

CamScanner

$i = 0, 1, \ldots N$ until $|(x_{i+1} - x_i)/x_{i+1}| \le e$ where e is the allowed error in the root.

The following algorithm finds a root of a polynomial. Observe that in order to compute b_0 and $g(x_0)$ we use Algorithms 1.5 and 1.4 of Chapter 2. S is used to accumulate the polynomial $g(x_0)$ and P for $f(x_0)$. This algorithm is known as the *Birge-Vieta* method.

Algorithm 3.7 Newton-Raphson Method for a Polynomial Equation

```
Read x0, e, n, N
Remarks: x0 is the initial guess of the root, e the allowed error, n the
         order of the polynomial and N the total number of iterations.
         x0 is guessed using the points mentioned earlier
for i = 0 to n in steps of 1 do Read a_i endfor
for i = 0 to n - 1 in steps of 1 do Read b_i endfor
      P ← a_n
      b_{n-1} ← a_n
      S ← b_{n-1}
for k = 1 to N in steps of 1 do
      for i = 1 to n - 1 in steps of 1 do
            b_{n-(i+1)} ← a_{n-i} + x0 b_{n-i}
            S ← b_{n-(i+1)} + x0 S
      endfor
            P ← a_0 + b_0 x_0
            x_1 ← x_0 - (P/S)
      if |x_1 - x_0/x_1| ≤ e GOTO 18
        else
            x_0 ← x_1
      endif
endfor
Write "Root not found in N iterations"
Write S, P, x_1, x_0
Stop
Write "Root found in k iterations"
x_0 ← x_1
Write x_0, S, P
Stop
```

We now give an example of using the Birge-Vieta method.

Example 3.5

All the roots of the polynomial equation

$$f(x) = x^4 - 4x^3 + 3x^2 + 8x - 10 = 0 \tag{3.49}$$

are to be found.

The bounds of the roots of Eq. (3.49) are given by (3.34) and are:

$$\pm\sqrt{\left(\frac{-4}{1}\right)^2 - 2\left(\frac{3}{1}\right)} = \pm\sqrt{10} < \pm 3.2$$

Thus we plot $f(x)$ in the range $\pm$ 3.2

$x =$	−3.2	−3	−2	−1	0	1	2	3	3.2
$f(x) =$	233.6	182	34	−10	−10	−2	2	14	22.8

From the above table we see that there are two real roots. One root is between − 2 and − 1 and the other between 1 and 2. We first find the root between 1 and 2 starting with a trial value of 1.5 using Algorithm 3.7. The following table shows the calculations.

Let $x_0 = 1.5$

i	a_i	b_i	S	$P = a_0 + b_0 x_0$
0	−10	6.875	3.6875	$= -10 + (6.875)(1.5)$
1	8	−7.5	−2.125	$x_1 \leftarrow x_0 - P/S$
2	3	−2.5	−1	$= 1.5 - .3125/3.6875$
3	−4	1	1	
4	1			$= 1.4153$

$x_0 = 1.4153$

i	a_i	b_i	S	
0	−10	7.0686	3.8121	$x_0 \leftarrow 1.4153 - .0041/3.8121$
1	8	−0.6581	−2.3132	
2	3	−2.5847	−1.1694	$= 1.4142$
3	−4	1	1	$e = 0.00077$
4	1			

As calculations used are up to 4 significant digits only, we stop here as $e \leq 0.001$. Thus $x_0 = 1.414$ is one root

To find the other root which is between − 2 and − 1, we again start with a trial value of − 1.5 and obtain − 1.414 as a root using a similar calculation.

Now we remove root 1.414 from Eq. (3.49) using Algorithm 1.5. We get the values of b_i from the last table above as: $b_3 = 1$, $b_2 = -2.5847$, $b_1 = -.6581$ and $b_0 = 7.0686$

Thus the reduced polynomial is

$$x^3 - 2.5847\,x^2 - 0.6581\,x + 7.0686 = 0 \qquad (3.50)$$

We now remove $x = -1.414$ from Eq. (3.50) as shown in the table below:

i	b_i	c_i	$c_2 \leftarrow b_3 = 1$
0	7.0686	4.9961	$c_1 \leftarrow b_2 + (-1.414)c_2$
1	−0.6581	−3.9987	$= -2.5847 - 1.414$
2	−2.5847	1	$= -3.9987$
3	1		$c_0 \leftarrow b_1 + (-1.414)c_1$
4			$= 4.9961$

The reduced polynomial is

$$x^2 - 3.9987x + 4.9961$$

The solution of this is

$$3.9987 \pm \sqrt{(3.9987)^2 - 19.9844}/2$$

$$x_{1,2} = (3.9987 \pm i2)/2 = 1.9994 \pm i$$

Thus the four roots are:

$$+1.414, -1.414, 1.9994 \pm i$$

In fact the correct roots are:

$$\pm\sqrt{2}, 2 \pm i$$

which are close to what we have found numerically.

In the example given above we found the two real roots and removed them from the polynomial. After removing these roots we obtained a quadratic equation which was solved using a known closed form formula. If a polynomial equation has complex coefficients then one or more roots are complex. In this case we have no choice but to use complex arithmetic. Many programming languages such as Fortran allow the use of complex arithmetic by declaring the required variables as complex. Complex arithmetic is, however, expensive in terms of computing time. Thus it should be used with some care. When a complex root is found it may be used to deflate the polynomial. If the resulting polynomial has all real coefficients then one may revert to real arithmetic and method of finding real and complex conjugate roots.

In a polynomial equation with real coefficients when all the real roots are removed an even degree polynomial will be left which will factorize to

$$(x^2 + p_1x + q_1)(x^2 + p_2x + q_2) \ldots (x^2 + p_kx + q_k)$$

With each quadratic factor having complex conjugate roots. Complex arithmetic may be now used to find the complex roots. However, it is more economical to find the quadratic factors, one after another. We can then find the complex conjugate roots of each quadratic factor individually. There are many methods

4

Solution of Simultaneous Algebraic Equations

4.1 Introduction

Simultaneous linear algebraic equations occur often in diverse fields of science and engineering and is an important area of study. In this chapter we will be mainly concerned with the solution of a set of n linear algebraic equations in n unknowns.

We are normally taught (in schools) a method of solving simultaneous equations, based on the evaluation of determinants, called Cramer's rule. For a small number of simultaneous equations (namely, 3 or 4) this rule is satisfactory. The number of multiplication operations needed when this method is used increases very rapidly as the number of equations increase. For example for 10 equations it is nearly 70 million arithmetic operations and with 50 equations it goes up to 10^{64} operations which is very large even for a fast computer. A different approach is thus needed for solving such equations on a computer.

In this chapter we will consider two techniques of solving simultaneous equations. These methods are particularly suited for computers. Typically these techniques would need about 1000 arithmetic operations for solving 10 simultaneous equations as compared to seventy million operations of Cramer's rule. The two methods are commonly known as the *direct method* and the *iterative method*. The direct method is based on the elimination of variables to transform the set of equations to a triangular form. The iterative method is a successive approximation procedure. Each one of these methods has its advantages and an understanding of the methods is needed to make a judicious choice when a set of equations is given.

4.2 The Gauss Elimination Method

To illustrate the method we shall consider three simultaneous equations in three unknowns:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \tag{4.1}$$

CamScanner

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = a_{24} \tag{4.2}$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = a_{34} \tag{4.3}$$

The first step is to eliminate the first term from Equations (4.2) and (4.3). In order to do it Equation (4.1) is divided by a_{11} and multiplied by a_{21} and subtracted from (4.2). This eliminates x_1 from (4.2) as shown below:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \tag{4.1}$$

$$\left(a_{21} - \frac{a_{11}}{a_{11}}a_{21}\right)x_1 + \left(a_{22} - \frac{a_{12}}{a_{11}}a_{21}\right)x_2 + \left(a_{23} - \frac{a_{13}}{a_{11}}a_{21}\right)x_3 = \left(a_{24} - \frac{a_{14}}{a_{11}}a_{21}\right) \tag{4.4}$$

If we call $\frac{a_{21}}{a_{11}} = k_2$ Equation (4.4) may be rewritten as

$$(a_{21} - k_2a_{11})x_1 + (a_{22} - k_2a_{12})x_2 + (a_{23} - k_2a_{13})x_3 = (a_{24} - k_2a_{14}) \tag{4.5}$$

Multiplying Equation (4.1) by $\left(\frac{a_{31}}{a_{11}}\right) = k_3$ and subtracting from (4.3) we get

$$(a_{31} - k_3a_{11})x_1 + (a_{32} - k_3a_{12})x_2 + (a_{33} - k_3a_{13})x_3 = (a_{34} - k_3a_{14}) \tag{4.6}$$

Observe that $(a_{21} - k_2a_{11})$ and $(a_{31} - k_3a_{11})$ are both zero.

In the steps above we have assumed that a_{11} is not zero. We will consider the case when a_{11} is zero later so that too many issues are not confused at this time. The above elimination procedure is algorithmically expressed below.

Algorithm 4.1 Elimination of x_1 from 3 Equations

1 *for* i = 1 to 3 and j = 1 to 4 in steps of 1 *do* Read a_{ij} *endfor*
2 *for* i = 2 to 3 in steps of 1 *do*
3 $\quad u \leftarrow a_{ij}/a_{11}$
4 $\quad$ *for* j = 1 to 4 in steps of 1 *do*
5 $\quad\quad a_{ij} \leftarrow a_{ij} - ua_{1j}$ *endfor*
endfor

In the above algorithm instructions 2 and 4 both command that instructions up to 5 be repeated a certain number of times. The way it is executed is as follows:

Instruction 2 commands that all instructions up to and including 5 be repeated first with $i = 2$ and next with $i = 3$. Thus i is set equal to 2 and the next instruction taken up for execution. This instruction sets $u = a_{21}/a_{11}$. Instruction 4 commands that instruction 5 be done first with $j = 1$, then with $j = 2$, next with $j = 3$ and lastly with $j = 4$. Thus when instruction 5 is reached $i = 2$ and

$j = 1$. After executing 5 with these values of i and j the inner loop (consisting of instruction 5) is executed again with $i = 2$ and $j = 2, 3, 4$. After the inner loop is executed for all values of j, the outer loop command (namely, instruction 2) is taken up for execution. Thus i is set equal to 3 and with this value the inner loop is again executed for all values of j, namely, 1, 2, 3 and 4.

This type of a structure of loops in an algorithm is called *nested* loops. In nested loops if the outer loop is to be repeated for $i = 1, n$ and the inner loop for $j = 1$ to m then the instructions inside the inner loop will be executed mn times and those in the outer loop n times. The sequence of values for (i, j) is shown below:

$$\begin{array}{lll} i & 1\,1\,1 \ldots 1 & 2\,2\,2 \ldots 2 \ldots n\,n\,n \ldots n \\ j & 1\,2\,3 \ldots m & 1\,2\,3 \ldots m \ldots 1\,2\,3 \ldots m \end{array}$$

The execution of Algorithm 4.1 is traced in the table below:

i	j	
2		$u \leftarrow a_{21}/a_{11}$
2	1	$a_{21} \leftarrow a_{21} - ua_{11} \left(a_{21} \leftarrow a_{21} - \frac{a_{21}}{a_{11}} a_{11} = 0 \right)$
2	2	$a_{22} \leftarrow a_{22} - ua_{12}$
2	3	$a_{23} \leftarrow a_{23} - ua_{13}$
2	4	$a_{24} \leftarrow a_{24} - ua_{14}$
3		$u \leftarrow a_{31}/a_{11}$
3	1	$a_{31} \leftarrow a_{31} - ua_{11} = 0$
3	2	$a_{32} \leftarrow a_{32} - ua_{12}$
3	3	$a_{33} \leftarrow a_{33} - ua_{13}$
3	4	$a_{34} \leftarrow a_{34} - ua_{14}$

The reduced equations are:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \tag{4.6}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24} \tag{4.7}$$

$$a_{32}x_2 + a_{33}x_3 = a_{34} \tag{4.8}$$

The next step is to eliminate $a_{32}x_2$ from Equation (4.8). This is done by multiplying Equation (4.7) by $u = a_{32}/a_{22}$ and subtracting the resulting equation from (4.8). Algorithmically this is represented below.

Algorithm 4.2 For Eliminating x_2 from Third Equation

1 $i = 3$
2 $u \leftarrow a_{i_2}/a_{22}$

3 *for* $j = 2$ to 4 in steps of 1 *do*
4 $\quad a_{ij} \leftarrow a_{ij} - ua_{2j}$ *endfor*

Tracing the above algorithm we get the table;

i	j	
3		$u \leftarrow a_{32}/a_{22}$
3	2	$a_{32} \leftarrow a_{32} - u\,a_{22} = a_{32} - \dfrac{a_{32}}{a_{22}} a_{22} = 0$
3	3	$a_{33} \leftarrow a_{33} - u\,a_{23}$
3	4	$a_{34} \leftarrow a_{34} - u\,a_{24}$

Thus at the end of this step the equations are

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = a_{14} \tag{4.1}$$

$$a_{22}x_2 + a_{23}x_3 = a_{24} \tag{4.7}$$

$$a_{33}x_3 = a_{34} \tag{4.9}$$

The above set of equations are said to be in a *triangular* form.

Algorithm 4.3 Elimination of x_1 Extended to n Unknowns

1 *for* $i = 1$ to n in steps of 1 and $j = 1$ to $(n + 1)$ in steps of 1 *do*
Read a_{ij} *endfor*
2 *for* $i = 2$ to n in steps of 1 *do*
3 $\quad u \leftarrow a_{i1}/a_{11}$
4 $\quad$ *for* $j = 1$ to $(n + 1)$ in steps of 1 *do*
5 $\quad\quad a_{ij} \leftarrow a_{ij} - u\,a_{1j}$ *endfor*
endfor

Algorithm 4.4 Elimination of x_{n-1} Extended to n Unknowns

1 $i = n$
2 $u \leftarrow a_{i(n-1)}/a_{(n-1)(n-1)}$
3 *for* $j = (n - 1)$ to $(n + 1)$ in steps of 1 *do*
4 $\quad a_{ij} \leftarrow a_{ij} - u\,a_{(n-1)j}$ *endfor*

In general while eliminating x_k the procedure would be:

Algorithm 4.5 Eliminating x_k from n Equations

1 *for* $i = (k + 1)$ to n in steps of 1 *do*
2 $\quad u \leftarrow a_{ik}/a_{kk}$
3 $\quad$ *for* $j = k$ to $(n + 1)$ in steps of 1 *do*
4 $\quad\quad a_{ij} \leftarrow a_{ij} - u\,a_{kj}$ *endfor*
endfor

CamScanner

Combining these ideas we may evolve the general algorithm given below to triangularize a set of n equations in n unknowns.

Algorithm 4.6 Triangularizing n Equations in n Unknowns

```
for i = 1 to n in steps of 1 and j = 1 to (n + 1) in steps of 1 do
      Read a_ij    endfor
for k = 1 to n − 1 in steps of 1 do
      for i = (k + 1) to n in steps of 1 do
            u ← a_ik/a_kk
         for j = k to (n + 1) in steps of 1 do
            a_ij ← a_ij − u a_kj    endfor
      endfor
endfor
```

The student is urged to check the above algorithm for $n = 3$ by making a table.

From the triangular form obtained in Equations (4.1), (4.7) and (4.9) the values of x_3, x_2 and x_1 are obtained by *back substitution* as follows:

$$x_3 = a_{34}/a_{33} \tag{4.10}$$

$$x_2 = (a_{24} - a_{23}x_3)/a_{22} \tag{4.11}$$

$$x_1 = (a_{14} - a_{12}x_2 - a_{13}x_3)/a_{11} \tag{4.12}$$

Extending the above to n equations the back substitution may be expressed as the following equations:

$$x_n = a_{n(n+1)}/a_{nn}. \tag{4.13}$$

for $i = (n - 1), (n - 2), (n - 3), \ldots 1$ we may write

$$x_i = (a_{in+1} - \sum_{j=(i+1)}^{n} a_{ij}x_j)/a_{ii} \tag{4.14}$$

The above equations may be expressed as algorithm 4.7 as given below.

Algorithm 4.7 Back Substitution

```
x_n ← a_n(n+1)/a_nn
for i = (n − 1) to 1 in steps of − 1 do
sum ← 0
      for j = (i + 1) to n in steps of 1 do
      sum ← sum + a_ij x_j    endfor
   x_i ← (a_i(n+1) − sum)/a_ii
endfor
```

Example 4.1

Given the simultaneous equations shown below (i) triangularize them (ii) use back substitution to solve for x_1, x_2, x_3

$$2x_1 + 3x_2 + 5x_3 = 23$$
$$3x_1 + 4x_2 + x_3 = 14$$
$$6x_1 + 7x_2 + 2x_3 = 26$$

Solution

The matrix of coefficients is given as Row 1, Row 2, and Row 3. We give below each row how they get altered by the algorithm

	a_{11}	a_{12}	a_{13}	a_{14}	
	2	3	5	23	Row 1
	3	4	1	14	Row 2
Step 1	0	– 0.5	– 6.5	– 20.5	Row 2.1
	6	7	2	26	Row 3
Step 2	0	– 2	– 13	– 43	Row 3.1
Step 3	0	0	13	39	Row 3.2

Triangularization

Step 1: Divide Row 1 by 2 and multiply it by 3. Subtract from Row 2.

Step 2: Divide Row 1 by 2 and multiply it by 6. Subtract from Row 3.

Step 3: Divide Row 2.1 by – 0.5 and multiply it by – 2. Subtract from Row 3.1.

At the end of these steps the equations are in triangular form.

$a_{11} = 2,\quad a_{12} = 3,\quad a_{13} = 5,\quad a_{14} = 23$

$a_{21} = 0,\quad a_{22} = -0.5,\quad a_{23} = -6.5,\quad a_{24} = -20.5$

$a_{31} = 0,\quad a_{32} = 0,\quad a_{33} = 13,\quad a_{34} = 39$

Using Algorithm 4.7 for backsubstitution

$$x_3 \leftarrow a_{34}/a_{33} = 39/13 = 3$$
$$i = 2, j = 3 \text{ sum} \leftarrow 0 + a_{23}x_3 = -19.5$$
$$x_2 \leftarrow a_{24} - (-19.5)/a_{22} = (-20.5 + 19.5)/-0.5 = 2$$
$$i = 1, j = 2 \text{ sum} \leftarrow 0 + a_{12}x_2 = 3 * 2 = 6$$
$$i = 1, j = 3 \text{ sum} \leftarrow 6 + a_{13}x_3 = 6 + 15 = 21$$
$$x_1 \leftarrow (a_{14} - \text{sum})/a_{11} = (23 - 21)/2 = 1$$

Thus $x_1 = 1$, $x_2 = 2$ and $x_3 = 3$.

CamScanner

values of e_i's obtained. A new approximation to the solutions is then

$$x_i^{(1)} = x_i^{(0)} + e_i^{(0)}$$

This procedure may be repeated by substituting $x_i^{(1)}$'s in (4.1), (4.2), (4.3) and further refinements to solution obtained. It has been found that a few cycles of this refinement procedure lead to more accurate solutions even in illconditioned systems.

4.6 The Gauss-Seidel Iterative Method

The Gauss-Seidel method will be first illustrated with two simultaneous equations. We will later generalise it to n equations in n unknowns.

Consider the simultaneous equations given below:

$$x_1 + x_2 = 2 \tag{4.27}$$

$$3x_1 - 10x_2 = 3 \tag{4.28}$$

Start with an initial value of $x_2^0 = 0$. Substitute this in Equation (4.27) and obtain the value of x_1^0 as 2. Use this guessed value in Equation (4.28) to obtain a refined guess for x_2. This is given by $x_2^1 = (3x_1^0 - 3)/10 = (6 - 3)/10 = .3$. Use this in Equation (4.27) to get another value of $x_1^{(2)}$. The progress of the iteration is shown in Table 4.1.

TABLE 4.1 Illustrating Iterative Method

Iteration Number	x_1	x_2
1	2	.3
2	1.7	.21
3	1.79	.237
4	1.763	.229
5	1.771	.231
6	1.769	.231
7	1.769	.231

Initial g $x_2^0 = 0$

Observe that the values of x_1 and x_2 converge to 1.769 and .231 respectively which is the solution. As this is an iterative method iterations are stopped when successive values of x_1 as well as of x_2 are 'close enough'.

This is a successive approximation method and the technique is illustrated graphically in Figure 4.2. The student is urged to compare this with the iterative technique presented in Chapter 3, Section 3.7. The convergence of the iterative method for two simultaneous equations will be investigated now.

Consider the equations

$$a_{11}x_1 + a_{12}x_2 = a_{13} \tag{4.29}$$

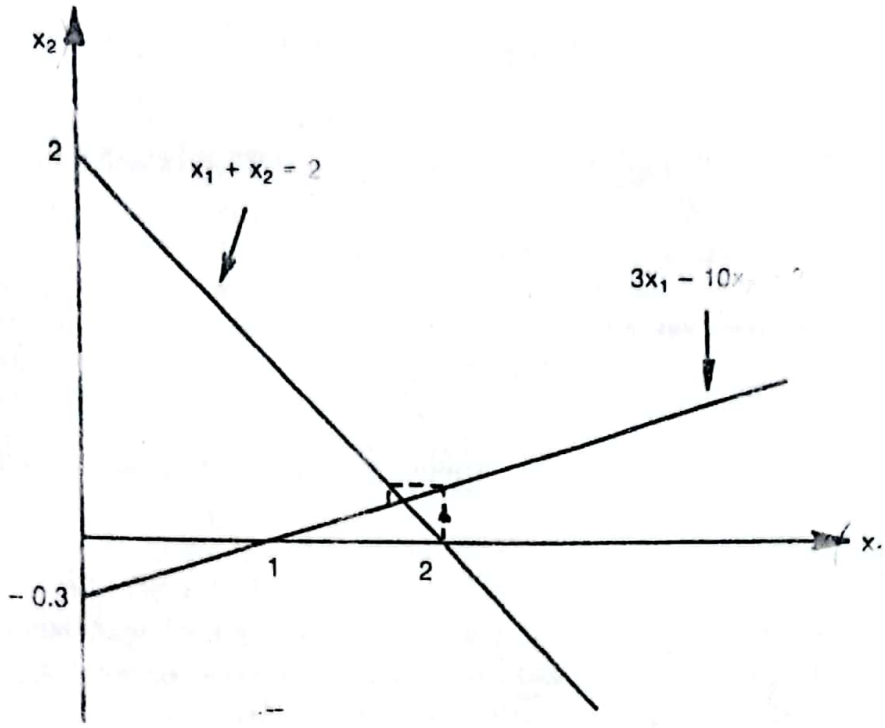


Fig. 4.2 Illustrating the Gauss-Seidel method.

$$a_{21}x_1 + a_{22}x_2 = a_{23} \tag{4.30}$$

The iterative method to proceed from the kth to the $(k+1)$th iteration is shown below:

$$a_{11}x_1^{k+1} = a_{13} - a_{12}x_2^k \tag{4.31}$$

$$a_{22}x_2^{k+1} = a_{23} - a_{21}x_1^{k+1} \tag{4.32}$$

Eliminating $x_1^{(k+1)}$ between Equations (4.31) and (4.32) we get

$$\frac{a_{11}}{a_{21}}[-a_{22}x_2^{k+1} + a_{23}] = a_{13} - a_{12}x_2^k \tag{4.33}$$

Similarly for the $(k+1)$th to $(k+2)$th iteration we get

$$\frac{a_{11}}{a_{21}}[-a_{22}x_2^{k+2} + a_{23}] = a_{13} - a_{12}x_2^{k+1} \tag{4.34}$$

Subtracting (4.33) from (4.34) and calling $(x_2^{k+1} - x_2^k) = e_2^k$ and $(x_2^{k+2} - x_2^{k+1}) = e_2^{k+1}$ we get

$$e_2^{k+1} = \frac{a_{12}a_{21}}{a_{11}a_{22}} e_2^k \tag{4.35}$$

Thus if the difference between successive iterations is to decrease we must satisfy the condition

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| < 1 \tag{4.36}$$

If we eliminate x_2 and work with x_1 and call the difference between successive iterations in x_1, namely

$$(x_1^{k+1} - x_1^k) = e_1^k$$

then we obtain the equation

$$e_1^{k+1} = \frac{a_{12}a_{21}}{a_{11}a_{22}} e_1^k \tag{4.37}$$

Again for the convergence of the iterative procedure we should ensure that

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| < 1 \tag{4.36}$$

Convergence will thus be ensured if the diagonal terms are larger than the off diagonal terms. If $|a_{11}| > |a_{12}|$ and $|a_{22}| \geq a_{21}$ or if $|a_{11}| \geq a_{12}$ and $|a_{22}| > a_{21}$ the above condition (Equation 4.36) would be satisfied.

For Equations (4.27) and (4.28) we have

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| = \left|\frac{3}{-10}\right| = 0.3$$

and the procedure should converge. If the same equations are interchanged we get

$$3x_1 - 10x_2 = 3 \tag{4.28}$$

$$x_1 + x_2 = 2 \tag{4.27}$$

For these equations

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| = \left|\frac{-10 \times 1}{3}\right| = 3.33 > 1$$

Thus the same iterative procedure would diverge!

The student is urged to draw the straight lines corresponding to these equations and satisfy himself that the procedure diverges and answer why it diverges.

4.7 An Algorithm to Implement the Gauss-Seidel Method

Consider the set of n equations in n unknowns

$$a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n = a_{1n+1} \tag{4.38}$$

$$a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n = a_{2n+1} \tag{4.39}$$

CamScanner

$$a_{31}x_1 + a_{32}x_2 + \dots a_{3n}x_n = a_{3n+1}$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \dots a_{nn}x_n = a_{nn+1} \tag{4.41}$$

In the Gauss-Seidel method $x_2^0 \dots x_n^0$ are set equal to zero and x_1^1 is calculated from (4.38). This value of x_1 and zeros are substituted for $x_3, x_4, \dots, x_n$ in (4.39) and x_2 computed from Eq. (4.39). The values of x_1^1 and x_2^1 along with zeros for $x_4, x_5, \dots x_n$ are used in (4.40) to compute x_3^1. Finally $x_1^1, x_2^1, \dots, x_{n-1}^1$ are used in (4.41) to compute x_n^1. With this the first iteration ends. In the second iteration $x_2^1, x_3^1 \dots x_n^1$ are used to compute x_1^2. $x_1^2, x_3^1 \dots x_n^1$ are used to get x_2^2 and so on. The main point to observe is that always the latest approximations for the values of variables are used in an iteration.

The complete algorithm to implement the Gauss-Seidel iteration is given as Algorithm 4.9.

Algorithm 4.9 Gauss-Seidel Iterative Method

```
1  for i = 1 to n in steps of 1 and j = 1 to n + 1
   in steps of 1 do Read a_ij endfor
2  Read e, maxit
   Remarks: e is the allowed relative error in the result vector. maxit
            is the maximum number of iterations allowed for the solution
            to converge.
3  for i = 1 to n in steps of 1 do x_i ← 0    endfor
   Remarks: All x_i s are initialized to zero. The following for loop sets
            the limit maxit for the maximum number of iterations to
            be allowed.
4  for iter = 1 to maxit do
5  big ← 0
6          for i = 1 to n in steps of 1 do
7                sum ← 0
8                for j = 1 to n in steps of 1 do
9                  if (j ≠ i) then sum ← sum + a_ij x_j endfor
10                 temp ← (a_in+1 − sum)/a_ii
11                 relerror ← |(x_i − temp)/temp|
12                 if relerror > big then big ← relerror
13                 x_i ← temp
           endfor
14 if (big ≤ e), then
           begin Write 'Converges to a solution'
                 for i = 1 to n in steps of 1 do Write x_i endfor
                 Stop end
   endfor
15 Write 'Does not converge in maxit iterations'
16 for i = 1 to n in steps of 1 do Write x_i endfor
17 Stop
```

The main part of Algorithm 4.9 starts with statement 5. The variable big stores the largest magnitude of the relative error in the variables. It is initialised to 0 in statement 5. The loop consisting of statements 8 and 9 calculates for a given $i = k$

$$\sum_{\substack{j=1 \\ j \neq k}}^{n} a_{kj} x_j$$

In statement 10 the latest value of x_k is calculated and temporarily stored giving

$$\text{temp} = \text{latest value of } x_k = (a_{k(n+1)} - \sum_{\substack{j=1 \\ j \neq k}}^{n} a_{kj} x_j)/a_{kk}$$

Statement 11 calculates the relative error in x_i (for $i = k$) between two latest iterates. Statement 12 picks the biggest relative error in all x_is. In statement 13 x_i is set to the latest calculated value of the variable. Loop between 6 and 13 calculates the latest trial value of x_i for all i's. On coming out of the loop 'big' has the highest relative error

$$\max_i |x_i^{iter} - x_i^{iter-1}|$$

When big is smaller than the assigned error then the solution converges. Observe that if the relative error does not fall below the assigned error limit for a total number of iterations equal to maxit then the procedure stops after giving an appropriate message. Such a 'safety exit' is essential in all iterative methods. Typically the maximum number of iterations set would be 'reasonable', say, 25 iterations for a (20 × 20) set of simultaneous equations.

Observe that in the inner loop when a value of x_i for (say) $i = k$ is calculated the latest calculated values of all x_is for $i = 1$ to $(k - 1)$ is used.

The analysis given to test the convergence of two simultaneous equations may be extended to n simultaneous equations. The condition is

$$|a_{ii}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \text{for all } i$$

and

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \text{for at least one } i$$

These conditions are normally met when the diagonal elements dominate. Such cases occur particularly when simultaneous equations arise as a result of discretizing partial differential equations. In such applications a number of off diagonal elements are zero and sometimes the zeros form a regular pattern. When such a pattern is perceived the Gauss-Seidel algorithm may be modified to take advantage of the pattern and reduce the number of arithmetic operations.

The main part of Algorithm 4.9 starts with statement 5. The variable big stores the largest magnitude of the relative error in the variables. It is initialised to 0 in statement 5. The loop consisting of statements 8 and 9 calculates for a given $i = k$

$$\sum_{\substack{j=1 \\ j \neq k}}^{n} a_{kj} x_j$$

In statement 10 the latest value of x_k is calculated and temporarily stored giving

$$\text{temp} = \text{latest value of } x_k = (a_{k(n+1)} - \sum_{\substack{j=1 \\ j \neq k}}^{n} a_{kj} x_j)/a_{kk}$$

Statement 11 calculates the relative error in x_i (for $i = k$) between two latest iterates. Statement 12 picks the biggest relative error in all x_is. In statement 13 x_i is set to the latest calculated value of the variable. Loop between 6 and 13 calculates the latest trial value of x_i for all i's. On coming out of the loop 'big' has the highest relative error

$$\max_i | x_i^{iter} - x_i^{iter-1} |$$

When big is smaller than the assigned error then the solution converges. Observe that if the relative error does not fall below the assigned error limit for a total number of iterations equal to maxit then the procedure stops after giving an appropriate message. Such a 'safety exit' is essential in all iterative methods. Typically the maximum number of iterations set would be 'reasonable', say, 25 iterations for a (20 × 20) set of simultaneous equations.

Observe that in the inner loop when a value of x_i for (say) $i = k$ is calculated the latest calculated values of all x_is for $i = 1$ to $(k - 1)$ is used.

The analysis given to test the convergence of two simultaneous equations may be extended to n simultaneous equations. The condition is

$$| a_{ii} | \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} | a_{ij} | \quad \text{for all } i$$

and

$$| a_{ii} | > \sum_{\substack{j=1 \\ j \neq i}}^{n} | a_{ij} | \quad \text{for at least one } i$$

These conditions are normally met when the diagonal elements dominate. Such cases occur particularly when simultaneous equations arise as a result of discretizing partial differential equations. In such applications a number of off diagonal elements are zero and sometimes the zeros form a regular pattern. When such a pattern is perceived the Gauss-Seidel algorithm may be modified to take advantage of the pattern and reduce the number of arithmetic operations.

CamScanner

Lagrange Interpolation

The universal technique for interpolating is to fit a polynomial through the points surrounding the point 'y' where the value of the function is to be found. This polynomial is an approximation of the function and is used to find $f(y)$.

Consider for example the following table of hyperbolic cosines. Suppose it is required to find the value of Cosh x where $x = .16$.

x	0.1	0.2	0.3	0.4
Cosh x	1.005	1.020	1.045	1.081

The simplest polynomial is a straight line. One may thus assume a polynomial of the form

$$f(x) = a_1 + a_2 x \tag{5.1}$$

At $x = x_1$ and $x = x_2$ we have

$$f(x_1) = \text{Cosh } x_1 = \text{Cosh } .1 = a_1 + a_2 x_1 = 1.005 \tag{5.2}$$

$$f(x_2) = \text{Cosh } x_2 = \text{Cosh } .2 = a_1 + a_2 x_2 = 1.020 \tag{5.3}$$

These simultaneous equations may be solved for a_1 and a_2 and we obtain:

$$a_2 = [f(x_2) - f(x_1)]/(x_2 - x_1) \tag{5.4}$$

$$a_1 = f(x_1) - x_1 a_2 \tag{5.5}$$

Using (5.2) to (5.5) we get

$$a_2 = .15$$

$$a_1 = .99$$

Thus the polynomial is

$$p(x) = .99 + .15x \tag{5.6}$$

Substituting .16 for x we get

$$\text{Cosh } (.16) = .99 + .15\ (.16) = 1.014.$$

The true value of Cosh .16 is known to be 1.0128. The linear approximating polynomial is thus not sufficient. The next step is to use a second degree polynomial. One may assume the polynomial to be of the form

$$f(x) = a_1 + a_2 x + a_3 x^2 \tag{5.7}$$

The coefficients a_1, a_2 and a_3 may be found from the simultaneous equations given below:

$$f(x_1) = \text{Cosh } x_1 = a_1 + a_2 x_1 + a_3 x_1^2 \tag{5.8}$$

$$f(x_2) = \text{Cosh } x_2 = a_1 + a_2 x_2 + a_3 x_2^2 \tag{5.9}$$

$$f(x_3) = \text{Cosh}\, x_3 = a_1 + a_2 x_3 + a_3 x_3^2. \tag{5.10}$$

The form of the interpolating polynomial chosen in Equation (5.7) is not convenient for calculating a_1, a_2 and a_3. The following assumed form for the polynomial leads to a triangular form for the simultaneous equations thereby simplifying the determination of the coefficients of the polynomial.

$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2). \tag{5.11}$$

The coefficients b_1, b_2 and b_3 are given by the simultaneous equations:

$$\text{Cosh}\, x_1 = f(x_1) = b_1 \tag{5.12}$$

$$\text{Cosh}\, x_2 = f(x_2) = b_1 + b_2(x_2 - x_1) \tag{5.13}$$

$$\text{Cosh}\, x_3 = f(x_3) = b_1 + b_2(x_3 - x_1) + b_3(x_3 - x_1)(x_3 - x_2) \tag{5.14}$$

From the above equations we have by inspection

$$b_1 = f(x_1) \tag{5.15}$$

$$b_2 = [f(x_2) - f(x_1)]/(x_2 - x_1) \tag{5.16}$$

$$b_3 = \frac{[f(x_3) - f(x_1)] - [\{f(x_2) - f(x_1)\}(x_3 - x_1)/(x_2 - x_1)]}{(x_3 - x_1)(x_3 - x_2)}$$

$$= \frac{f(x_3) - f(x_2)}{(x_3 - x_2)(x_3 - x_1)} - \frac{f(x_2) - f(x_1)}{(x_2 - x_1)(x_3 - x_1)} \tag{5.17}$$

Substituting the values of $x_1, x_2, x_3, f(x_1), f(x_2)$ and $f(x_3)$ in the above equations we obtain:

$$b_1 = 1.005$$

$$b_2 = .015/.1 = .15$$

$$b_3 = (.025 - .015)/.02 = .5$$

$$\text{Thus Cosh}\,(.16) = 1.005 + .15(.06) + .5(.06)(-.04) = 1.0128 \tag{5.18}$$

This value is equal to the actual value of Cosh (.16) to 4 decimal digits. Observe that in Equation (5.18) the first two terms correspond to linear interpolation and the third term is in the nature of a correction-term.

The calculation of the coefficients of the polynomial may be further simplified by assuming the following form for the polynomial:

$$f(x) = c_1(x - x_2)(x - x_3) + c_2(x - x_1)(x - x_3) + c_3(x - x_1)(x - x_2). \tag{5.19}$$

By inspection of the above polynomial we have:

$$f(x_1) = c_1(x_1 - x_2)(x_1 - x_3) \text{ or } c_1 = \frac{f(x_1)}{(x_1 - x_2)(x_1 - x_3)} \tag{5.20}$$

$$f(x_2) = c_2(x_2 - x_1)(x_2 - x_3) \text{ or } c_2 = \frac{f(x_2)}{(x_2 - x_1)(x_2 - x_3)} \tag{5.21}$$

$$f(x_3) = c_3(x_3 - x_1)(x_3 - x_2) \text{ or } c_3 = \frac{f(x_3)}{(x_3 - x_1)(x_3 - x_2)} \tag{5.22}$$

It is seen above that the coefficients of the polynomial $f(x)$ are directly obtained without solving simultaneous equations. The polynomial may thus be expressed as

$$\frac{f(x_1)(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + \frac{f(x_2)(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + \frac{f(x_3)(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

and in a more concise notation as:

$$\boxed{f(x) = \sum_{i=1}^{3} f(x_i) \prod_{\substack{j \neq i \\ j=1}}^{3} \frac{(x - x_j)}{(x_i - x_j)}} \tag{5.23}$$

This polynomial is called the *Lagrange polynomial* and is simple to program on a computer. It is, however, extremely cumbersome to use for hand calculation. Further, there is no indication of the error committed in using this approximation and thus it is difficult to pick the order of the polynomial to be fitted. Increasing the order of the polynomial does not necessarily yield a better accuracy of interpolation (see Problem 5.2).

Example 5.1

A Table of x vs $f(x)$ is given below. Find the value of $f(x)$ at $x = 4$

x	1.5	3	6
$f(x)$	− 0.25	2	20

Solution

Using Lagrange Polynomial

$$f(x) = \frac{-0.25(x-3)(x-6)}{(1.5-3)(1.5-6)} + \frac{2(x-1.5)(x-6)}{(3-1.5)(3-6)} + \frac{20(x-1.5)(x-3)}{(6-1.5)(6-3)}$$

$$f(4) = \frac{-0.25(1)(-2)}{(-1.5)(-4.5)} + \frac{2(2.5)(-2)}{(1.5)(-3)} + \frac{20(2.5)(1)}{(4.5)3}$$

$$= .074 + 2.222 + 3.703$$

$$= 5.999 = 6 \text{ (rounded)}$$

Observe the complicated hand calculation with a lot of divisions.

The Lagrange polynomial may be generalised to the nth order as given below:

$$f(x) = \sum_{i=1}^{n+1} f(x_i) \prod_{\substack{j \neq i \\ j=1}}^{n+1} \frac{(x - x_j)}{(x_i - x_j)}. \tag{5.24}$$

This polynomial is indispensable in theoretical studies in numerical analysis but otherwise not very popular. An algorithm to interpolate based on this polynomial is given as Algorithm 5.1.

Algorithm 5.1 Lagrange Interpolation

```
1  Read x, n
2  for i = 1 to (n + 1) in steps of 1 do Read x_i, f_i endfor
   Remarks: The above statement reads x_i s and the corresponding values
            of f_i s.
3  sum ← 0
4  for i = 1 to (n + 1) in steps of 1 do
5  prodfunc ← 1
6        for j = 1 to (n + 1) in steps of 1 do
7              If (j ≠ i) then
               prodfunc ← prodfunc × (x − x_j)/(x_i − x_j)
         endfor
8  sum ← sum + f_i × prodfunc
   Remarks: sum is the value of f at x
   endfor
9  Write x, sum
10 Stop
```

5.3 Difference Tables

In Equation (5.11) an assumed form for the polynomial was given as:

$$f(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) \tag{5.11}$$

The coefficients b_1, b_2 and b_3 were given in Equations (5.15), (5.16) and (5.17). The method of obtaining these coefficients may also be expressed in the tabular form of Table 5.2.

The polynomial of Equation (5.11) may thus be written as

$$f(x) = f(x_1) + \Delta_d f_1(x - x_1) + \Delta_d^2 f_1(x - x_1)(x - x_2) \tag{5.25}$$

where $\Delta_d f_1$ and $\Delta_d^2 f_1$ are called *divided differences* and the table called a *divided difference table*.

This may be generalised and for n tabulated points a $(n - 1)$ order polynomial may be fitted. The general polynomial formula is:

9

Numerical Solution of Differential Equations

9.1 Introduction

In this chapter we will consider the numerical solution of differential equations of the type

$$\frac{dy}{dx} = f(x, y) \tag{9.1}$$

with an initial condition

$$y = y_1 \text{ at } x = x_1 \tag{9.2}$$

The function $f(x, y)$ may be a general non-linear function of (x, y) or may be a table of values. When the value of y is given at $x = x_1$ and the solution is required for $x_1 \le x \le x_f$ then the problem is called an *initial value problem.* On the other hand if y is given at $x = x_f$ and the solution is required for $x_f \ge x \ge x_1$ then the problem is called a boundary value problem. In this book we will consider only the initial value problem.

We would now ask the question: 'what do we mean by the term solution of a differential equation?' A solution is a curve $g(x, y)$ in the (x, y) plane whose slope at every point (x, y) in the specified region is given by the equation $dy/dx = f(x, y)$. The initial point (x_1, y_1) of the solution curve and the slope of the curve at this point are given. With this information we are asked to *extrapolate* the values of y for the set of values of x in the range $[x_1, x_f]$. The fundamental method used in obtaining the solution of a differential equation is thus judicious extrapolation.

9.2 Euler's Method

This method uses the simplest extrapolation techniques for developing a solution. Equations (9.1) and (9.2) are written below for ready reference.

$$\frac{dy}{dx} = f(x, y) \tag{9.1}$$

$$y(x_1) = y_1 \tag{9.2}$$

CamScanner

Given (x_1, y_1) the slope at this point is obtained from (9.1) as

$$\frac{dy}{dx}(x_1, y_1) = f(x_1, y_1) \tag{9.3}$$

The next point y_2 on the solution curve may be extrapolated by taking a small step in a direction given by the above slope. Thus

$$y(x_1 + h) = y_2 = y_1 + hf(x_1, y_1) \tag{9.4}$$

One may similarly proceed from y_2 to y_3. In general the $(i + 1)$th point may be obtained from the ith point using the equation

$$y_{i+1} = y_i + hf(x_i, y_i) \tag{9.5}$$

A geometrical interpretation of the procedure is illustrated in Figure 9.1. It is seen that Euler's method is basically a technique of developing a *piecewise linear* approximation of the solution.

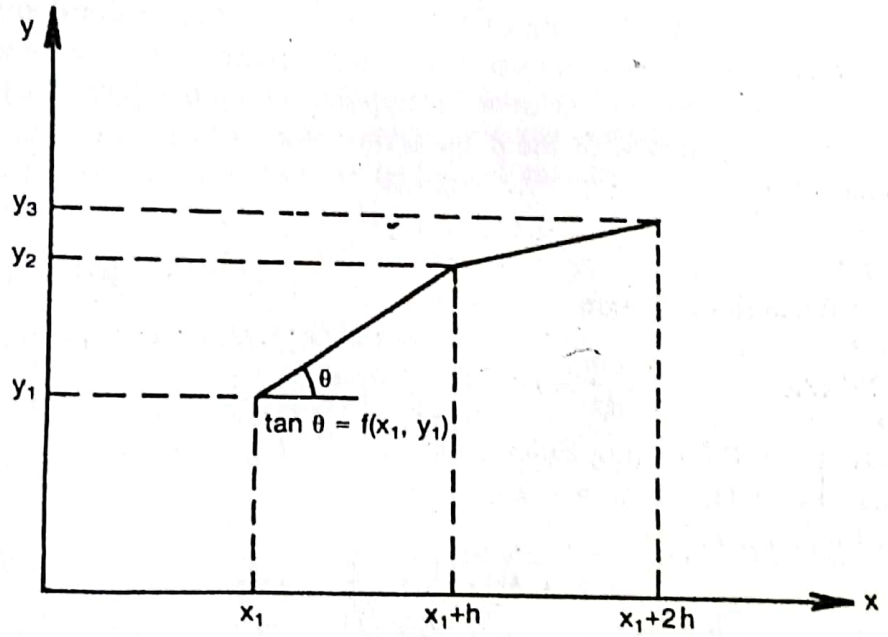


Fig. 9.1 Illustrating Euler's method.

At any step the error in the above formula is of the order of $\frac{h^2}{2} y''(z)$ where z is a point in the region of computation.

The error propagation in successive steps of the Euler technique may be obtained by the following simple argument. Suppose at the ith step the error in the solution y_i is e_i. Then the error e_{i+1} in y_{i+1} (which is obtained from y_i) is given by:

$$y_{i+1} = y^*_{i+1} + e_{i+1} = (y^*_i + e_i) + hf(x_i, y^*_i + e_i)$$

In the above equation y^*_i and y^*_{i+1} are the 'correct' solutions to the differential equation. Expanding $f(x_i, y^*_i + e_i)$ in a Taylor series:

$$y_{i+1}^* + e_{i+1} = y_i^* + e_i + h\left\{f(x_i, y_i^*) + e_i \left.\frac{\partial f}{\partial y}\right|_{x_i, y_i^*}\right\} \tag{9.6}$$

$$= y_i^* + hf(x_i, y_i^*) + e_i\left(1 + h\frac{\partial f}{\partial y}\right)$$

As $y_{i+1}^* \simeq y_i^* + hf(x_i, y_i^*)$ Equation (9.6) reduces to:

$$e_{i+1} = e_i\left(1 + h\frac{\partial f}{\partial y}\right) \tag{9.7}$$

If $\left|1 + h\left(\frac{\partial f}{\partial y}\right)\right| < 1$ then the errors will damp down with successive iterations. In this case the Euler method is said to be *stable*. Otherwise the errors increase in successive iterations and the procedure will be *unstable*. In practice the error build up in using Euler's method is substantial and thus the method is rarely used. It would thus be necessary to search for better methods. One such technique is to use Taylor series approximation of $y(x)$ and truncate the series after the quadratic or the cubic terms. We will consider this in the next section.

Example 9.1

Solve the differential equation

$$\frac{dy}{dx} + xy = 0, \quad y(0) = 1$$

from $x = 0$ to $x = 0.25$ using Euler's method.

Let us pick $h = 0.05$, $f(x, y) = -xy$

The Euler formula is:

$$y_{i+1} = y_i + hf(x_i, y_i) = y_i - hx_iy_i$$

$$= y_i - (0.05)x_iy_i$$

At $x_0 = 0$,

$y_0 = 1$

$y_1 = 1 - 0.05 * 0 = 1$

$y_2 = 1 - (0.05)(0.05)(1) = 0.9975$

$y_3 = 0.9975 - (0.05)(0.1)(0.9975) = 0.9925$

$y_4 = 0.9925 - (0.05)(0.15)(0.9925) = 0.985$

$y_5 = 0.985 - (0.05)(0.2)(0.985) = 0.9752$

The solution is tabulated below:

CamScanner

x_i	0	0.05	0.1	0.15	0.2	0.25
y_i	1	1	0.997	0.992	0.985	0.975

The analytical solution to the differential equation is exp $(-x^2/2)$ and is tabulated below:

x_i	0	0.05	0.1	0.15	0.2	0.25
y_i	1	0.999	0.995	0.989	0.980	0.969

In this example

$$\frac{\partial f}{\partial y} = -x$$

$$\left|1 + h\frac{\partial f}{\partial y}\right| = |1 - 0.05x| \leq 0.9875 \text{ for } x_{max} = 0.25$$

Thus Euler's method will give a stable solution. The error will however build up as x_i increases. This is clear by inspecting the above tables of solutions obtained by Euler's method and the analytical solution.

9.3 Taylor Series Method

Consider again the differential equation

$$\frac{dy}{dx} = f(x, y) \tag{9.1}$$

$$y(x_1) = y_1 \tag{9.2}$$

If the solution curve $y(x)$ is expanded in a Taylor series around $x = x_1$ we obtain:

$$y(x) = y_1 + (x - x_1)\frac{dy}{dx} + \frac{(x - x_1)^2}{2}\frac{d^2y}{dx^2} + \frac{(x - x_1)^3}{6}\frac{d^3y}{dx^3} + \dots \tag{9.8}$$

Thus y at $x = x_1 + h$ may be obtained from the series

$$y(x_1 + h) = y_1 + h\left.\frac{dy}{dx}\right|_{x_1} + \frac{h^2}{2}\left.\frac{d^2y}{dx^2}\right|_{x_1} + \frac{h^3}{3}\left.\frac{d^3y}{dx^3}\right|_{x_1} + \dots \tag{9.9}$$

Observe that $\frac{dy}{dx} = f(x, y)$ and that y is a function of x. Thus

$$\frac{d^2y}{dx^2} = \frac{d}{dx}\left(\frac{dy}{dx}\right) = \frac{d}{dx}f(x, y) = \frac{\partial}{\partial x}f(x, y) + \frac{\partial}{\partial y}f(x, y)\frac{dy}{dx}$$

CamScanner

Call

$$\frac{\partial}{\partial x} f(x, y) = f_x$$

$$\frac{\partial}{\partial y} f(x, y) = f_y$$

and

$$f(x, y) = f \tag{9.10}$$

Equation (9.9) may be rewritten as

$$y(x_1 + h) = y_1 + hf(x_1, y_1) + \frac{h^2}{2}(f_x + f(x_1, y_1) f_y) + \dots \tag{9.11}$$

Observe from Equation (9.11) that Euler's method of extrapolation corresponds to taking only the first two terms of the above series. Thus the errors due to the truncation of the series would be of the order of h^2. An improvement of the Euler method would thus be to include the h^2 term in the above expansion and use the formula of Equation (9.12) to extrapolate to the point $(x_1 + h)$ from x_1

$$y_2 = y_1 + hf(x_1, y_1) + \frac{h^2}{2}[f_x(x_1, y_1) + f(x_1, y_1) f_y(x_1, y_1)] \tag{9.12}$$

The truncation error in the above formula is of the order of h^3. This method, however, is not applicable in general because the partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ of $f(x, y)$ are not always easy to obtain and considerable computational labour is involved in evaluating them. In some cases, as for instance, when $f(x, y)$ is given in a tabular form then this method is not applicable. Thus attempts have been made to find methods which will be in essence equivalent to the Taylor series method but will use only the values of $f(x, y)$ at specified values of x and y and will not require the derivatives to be evaluated. These mehods are the Runge-Kutta family of methods which will be considered next.

9.4 Runge-Kutta Methods

Consider the following geometric method of extrapolating the $y(x)$ curve to obtain the solution to the differential equation

$$\frac{dy}{dx} = f(x, y)$$

$$y(x_1) = y_1$$

Draw a straight line from (x_1, y_1) with a slope $s_1 = f(x_1, y_1)$. Let it cut the vertical line through $x_1 + h$ at $(x_1 + h, \bar{y}_2)$. [See Figure 9.2.] Determine the slope dy/dx of the solution curve $y(x)$ at $(x_1 + h, \bar{y}_2)$. This is given by $s_2 = f(x_2, \bar{y}_2)$ $(x_2 = x_1 + h)$. Now draw a straight line from (x_1, y_1) with a slope

8

Differentiation and Integration

8.1 Introduction

If a function is defined as an expression its derivative or integral may often be determined using the techniques of calculus. In a number of cases getting a closed form formula for integrating a function using calculus is difficult. For example the indefinite integral of $(\sin x/x)$ cannot be expressed in a closed form. Only an infinite series approximation is available for this function. There are other cases where the closed form expression is so complicated as to be almost useless. For example the integral

$$\int \left(\frac{p+x}{q-x}\right)^{1/2} = -\sqrt{p+x}\,\sqrt{q-x} - (p+q)\sin^{-1}\left(\frac{q-x}{p+q}\right)^{1/2}$$

is obtained after considerable difficulty. As in most problems in engineering and science we are concerned with getting numerical answers. After obtaining such expressions, values of variables are to be substituted. The formula may be such as to lead to difficult numerical computation giving results with severe rounding errors. There are many cases where experimental data is obtained whose integral may be required. In all the above cases we have to resort to numerical integration.

Differentiation operation in calculus is much easier compared to integration. It may be necessary to numerically differentiate if we are given a tabulated function or if the closed form expression is complicated.

Conceptually numerical differentiation and integration are performed by fitting an interpolating polynomial through a discrete set of values of the function and differentiating or integrating this polynomial. Any one of the interpolation polynomials discussed in Chapter 5 may be used.

The accuracy attainable by these methods would depend on the given function and the order of the polynomial used. If the polynomial fitted is exact then the error would be theoretically zero. In practice, however, rounding errors will introduce errors in the calculated values. The error introduced in obtaining derivatives is in general much worse than that introduced in determining integrals. The reason for this is evident from Figure 8.1. It may be observed

CamScanner

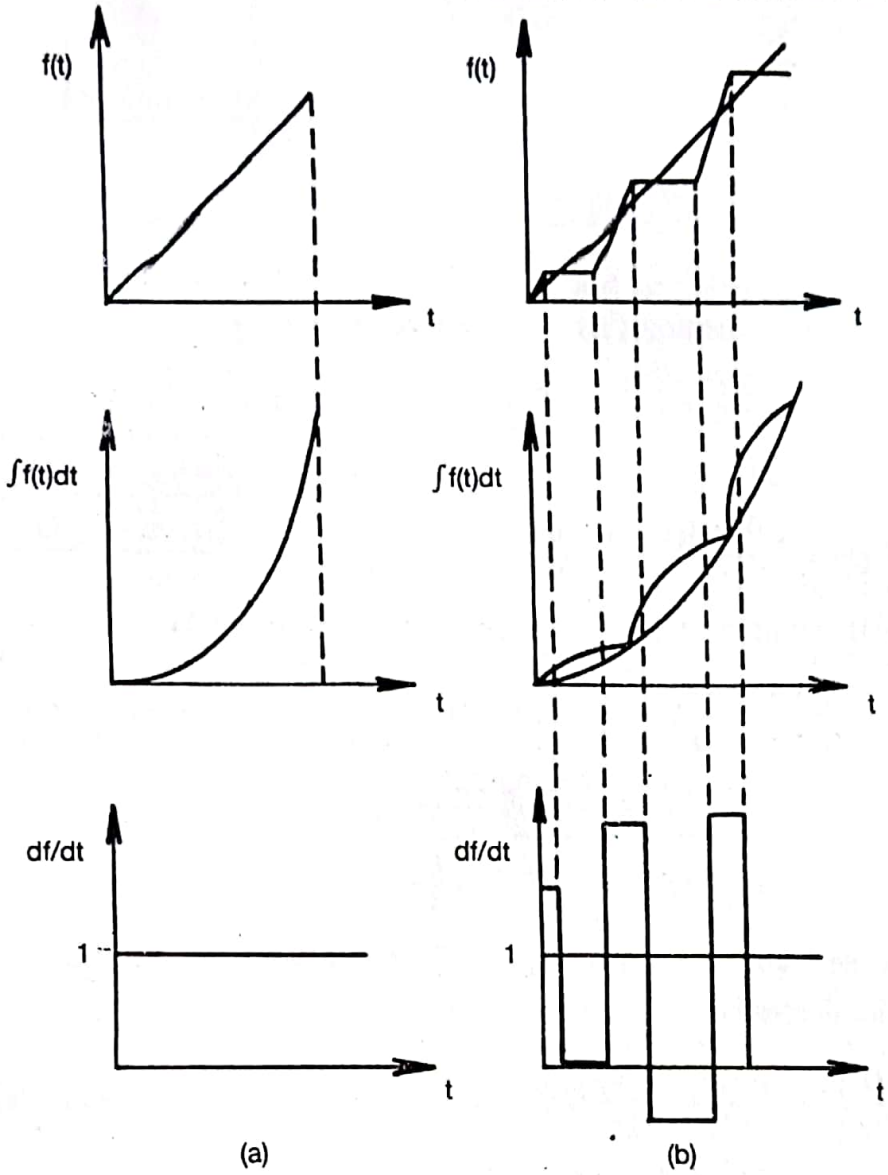


Fig. 8.1 Illustrating integration and differentiation of a function: (a) Function (b) Function $f(t)$ with superposed error.

that any errors in approximating a function are amplified while taking the derivative whereas they are smoothed out in integration.

Thus numerical differentiation should be avoided if an alternative exists.

8.2 Formulae for Numerical Differentiation

Formulae for numerical differentiation are obtained by using an interpolating polynomial through the appropriate set of points. For instance if the values of a function are available at $(x_1, x_2, \ldots, x_n)$ and its derivative at $x_1 < x < x_3$ is required we may use the polynomial

$$f(x) = f(x_1) + \frac{\Delta f_1}{h}(x - x_1) + \frac{\Delta^2 f_1}{2!h^2}(x - x_1)(x - x_2) + R(x) \qquad (8.1)$$

where Δf_1, $\Delta^2 f_1$ are the forward differences and $R(x)$ is the error term given by

$$R(x) = \frac{f^{(3)}(a)(x - x_1)(x - x_2)(x - x_3)}{3!} \tag{8.2}$$

In the above equation $x_3 \geq a \geq x_1$.
Differentiating Equation (8.1) with respect to x we get:

$$\frac{df}{dx} = \frac{\Delta f_1}{h} + \frac{\Delta^2 f_1}{2!h^2}(x - x_1 + x - x_2) + R'(x) \tag{8.3}$$

where

$$R'(x) = \frac{f^{(3)}(a)[(x - x_1)(x - x_2) + (x - x_1)(x - x_3) + (x - x_2)(x - x_3)]}{3!}$$

If df/dx is required at $x = x_2$ we get from Equation (8.3)

$$\left.\frac{df}{dx}\right|_{x_2} = \frac{\Delta f_1}{h} + \frac{\Delta^2 f_1}{2h^2}(x_2 - x_1) + \frac{f^{(3)}(a)}{6}(x_2 - x_1)(x_2 - x_3)$$

$$= \frac{f_2 - f_1}{h} + \frac{f_3 - 2f_2 + f_1}{2h} - f^{(3)}(a)\frac{h^2}{6}$$

$$= \frac{1}{2h}(f_3 - f_1) - f^{(3)}(a)\frac{h^2}{6} \tag{8.4}$$

Thus the derivative at $x = x_2$ is $(f_3 - f_1)/2h$ with a truncation error of $f^{(3)}(a)\frac{h^2}{6}$.

The derivative at $x = x_3$ is given by

$$\left.\frac{df}{dx}\right|_{x_3} = \frac{\Delta f_1}{h} + \frac{\Delta^2 f_1}{2h^2}(2x_3 - x_1 - x_2) + \frac{f^{(3)}(a)}{6}(x_3 - x_1)(x_3 - x_2)$$

$$= \frac{f_1 - 4f_2 + 3f_3}{2h} + \frac{f^{(3)}(a)}{3}h^2 \tag{8.5}$$

We will now consider an example.

Example 8.1

x	*Angle radians*	*Sine*	Δf	$\Delta^2 f$
x_1	.25	.2474		
			.0097	
				− .0001
x_2	.26	.2571		
			.0096	
				+ .0001
x_3	.27	.2667		
			.0097	
				− .0001
x_4	.28	.2764		
			.0096	
x_5	.29	.2860		

CamScanner

A portion of a table of sines is given above. It is required to find the derivative of this function at $x = .27$. From Equation (8.5) we get:

$$\left.\frac{df}{dx}\right|_{x_3} = \frac{\Delta f_1}{h} + \frac{3\Delta^2 f_1}{2h} = \frac{f_1 - 4f_2 + 3f_3}{2h}$$
$$= .9550$$

The correct value of the derivative to 4 digits is .9638. We will now use a coarser interval of tabulation and calculate the derivative again.

x	*Angle*	*Sine*	Δf	$\Delta^2 f$
x_1	.2	.1987		
			.0487	
				– .0006
x_2	.25	.2474		
			.0481	
				– .0007
x_3	.3	.2955		
			.0474	
x_4	.35	.3429		

From Equation (8.3) we obtain:

$$\left.\frac{df}{dx}\right|_{.27} = \frac{\Delta f_2}{h} + \frac{\Delta^2 f_2}{2h^2}(x - x_2 + x - x_3)$$
$$= \frac{.0481}{(.05)} + \frac{(-.0007)(.02 - .03)}{2(.05)^2}$$
$$= .9620 + .0014 = .9634$$

The second value with a coarser interval is nearer the true value!

The result obtained in the above example is not surprising as the differences of tabulated values are taken in differentiation and if these values are close there is a severe loss of significance. For instance we saw that

$$\left.\frac{df}{dx}\right|_{x_3} = \frac{f_1 - 4f_2 + 3f_3}{2h}$$

If there is a rounding error of e in f_3 and $-e$ in f_2 and e in f_1 then the error in the derivative due to rounding error would be

$$E_r = \frac{8e}{2h} = \frac{4e}{h}$$

With 4 significant digits in tabulated values and $h = .01$ this error could equal $\frac{4 \times .0001}{.01} = 0.04$. The truncation error on the other hand would be given by:

$$R'(x) = f^3(a)\frac{h^2}{6} \le h^2/6 \qquad (\because f(a) \text{ for } \sin x \le 1)$$

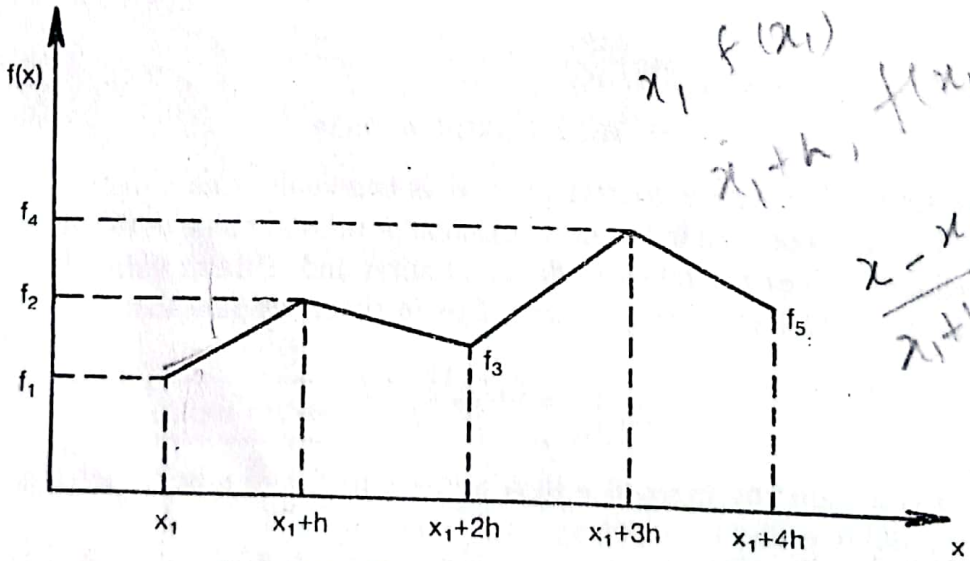
$$\therefore \qquad R'(x) \simeq \frac{h^2}{6} = \frac{(.01)^2}{6} = .000016$$

Thus for small values of h rounding errors would exceed truncation error. On the other hand for large h the truncation error would be high. It would thus be necessary to pick h to balance these two sources of error. Such balancing is quite difficult in practice. It is thus advisable to avoid differentiation whenever possible.

8.3 Numerical Integration

Formulae for numerical integration called *quadrature* are also based on fitting a polynomial through a specified set of points and integrating this approximating function. Assume that the values of a function $f(x)$ are given at x_1, $x_1 + h$, $x_1 + 2h \ldots x_1 + nh$ and that it is required to find the integral of $f(x)$ between x_1 and $x_1 + nh$. The simplest technique to use would be to fit straight lines through $f(x_1)$, $f(x_1 + h)$ ... and determine the area under this approximating function as shown in Figure 8.2. The equation of the straight line between x_i and $x_i + h$ is given by

$$f(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{h}(x - x_i)$$



x_1 $f(x_1)$

x_1+h, $f(x_1+h)$

$\frac{x - x_1}{x_1+h_1-x_1}$

Fig. 8.2 Illustrating trapezoidal rule.

$$\therefore \qquad S = \int_{x_1}^{x_n} f(x)\,dx = \sum_{i=1}^{n} \int_{x_i}^{x_i+h} \left[f_i + \frac{\Delta f_i}{h}(x - x_i)\right] dx$$

$$= \sum_{i=1}^{n} f_i h + \frac{(f_{i+1} - f_i)}{h}\frac{h^2}{2}$$

$$= \sum_{i=1}^{n} \frac{(f_i + f_{i+1})}{2} h \tag{8.6}$$

The integration formula of Equation (8.6) is known as *Trapezoidal rule*. The integral may be rewritten as

$$S = \left(\frac{f_1}{2} + f_2 + f_3 + f_4 \ldots + \frac{f_{n+1}}{2}\right) h \tag{8.7}$$

Thus to evaluate the integral the value of the function is evaluated at $(n + 1)$ points. Further, this rule gives the exact value for the integral only if the function $f(x)$ is linear. The question of error in using this formula will be discussed later.

Example 8.2

Integrate the function given in Example 8.1

$$S = \left(\frac{0.2474}{2} + 0.2571 + 0.2667 + 0.2764 + \frac{0.2860}{2}\right) \times 0.01$$

$$= 0.010669 \simeq 0.0107 \text{ (to 4 significant digits)}$$

From a table of integrals: $\int_{0.25}^{0.29} \sin x = -\cos x \Big|_{0.25}^{0.29}$

$$= 0.0107 \text{ (to 4 digits)}$$

Thus the matching is exact.

8.4 Simpson's Rule

Simpson's rule is a popular numerical integration technique. It is based on approximating the function $f(x)$ by fitting quadratics through sets of three points. This is illustrated graphically in Figure 8.3.

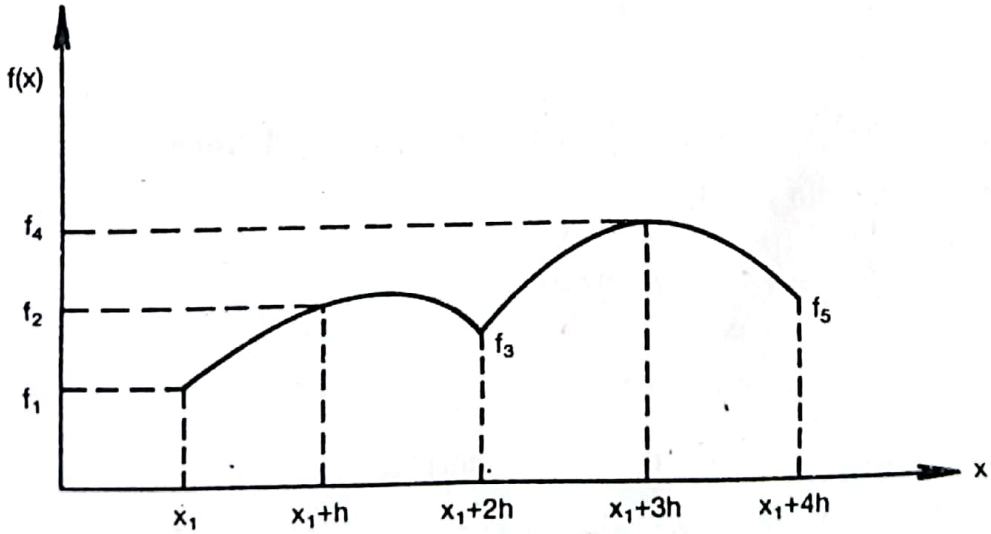


Fig. 8.3 Illustrating Simpson's rule.

Using a quadratic interpolation polynomial we get

$$S = \int_{x_1}^{x_1+nh} f(x)\,dx = \sum_{i=1,3,\ldots,n-2} \int_{x_i}^{x_i+2h} \left[f_i + \frac{\Delta f_i}{h}(x - x_i) + \frac{\Delta^2 f_i}{2h^2}(x - x_i)(x - x_i - h) \right] dx \tag{8.8}$$

Integrating the above expression we obtain

$$S = \sum_{i=1,3,5,\ldots,n-2} h/3\{f_i + 4f_{i+1} + f_{i+2}\} \tag{8.9}$$

$$= h/3\ \{f_1 + 4f_2 + 2f_3 + 4f_4 + 2f_5 + 4f_6 + \ldots + f_{n+1}\} \tag{8.10}$$

In using the above formula it is implied that f is tabulated at an odd number of points.

Example 8.3

Integrate the function tabulated below using Simpson's rule.

x	-0.6	-0.5	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3
$f(x)$	4	2	3	8	4	-2	2	3	5	8

Observe that the function is tabulated at 10 points.

Simpson's rule requires an odd number of points. We will use trapezoidal rule for the first two points, and Simpson's rule of the rest.

$$\int_{-0.6}^{0.3} f(x)\,dx = \frac{h}{2}(4 + 2) + \frac{h}{3}(2 + 12 + 16 + 16 - 4 + 8 + 6 + 20 + 8)$$

$$= \frac{0.1}{2}(6) + \frac{0.1}{3}(84) = 3.1$$

Example 8.4

Integrate $5x^3 - 3x^2 + 2x + 1$ from $x = -1$ to $x = 1$ using Simpson's rule with $h = 1$.

$$\text{Integral} = \frac{1}{3}[f(-1) + 4f(0) + f(1)] \quad \text{as } h = 1$$

$$= \frac{1}{3}(-5 - 3 - 2 + 1 + 4 + 5 - 3 + 2 + 1) = 0$$

The exact value of the integral is also 0 as it should be because Simpson's rule is exact up to a cubic.

The Simpson formula may also be derived by a technique called the

CS CamScanner

method of undetermined coefficients. In this method we seek an integration formula of the type

$$S = af_1 + bf_2 + cf_3 \tag{8.11}$$

where f_1, f_2 and f_3 are the values of the function being integrated at the three points x_1, $x_1 + h$ and $x_1 + 2h$ and a, b, c are constants to be determined. As Simpson's integration formula is based on fitting a quadratic through the points (x_1, f_1), $(x_1 + h, f_2)$, $(x_1 + 2h, f_3)$ we assume that it must lead to the exact integration of polynomials up to a quadratic. In other words the formula should be exact for a constant (zero order polynomial), straight line (first order polynomial) and a quadratic (second order polynomial). The coefficients a, b and c are chosen to ensure this. For simplifying the algebra the axis is shifted to make x_1 the origin. If a constant 1 is integrated Equation (8.11) gives

$$S_0 = a + b + c$$

as

$$f_1 = f_2 = f_3 = 1$$

By integrating a constant analytically we obtain

$$\int_0^{2h} 1\,dx = x\Big]_0^{2h} = 2h$$

Thus

$$S_0 = a + b + c = 2h \tag{8.12}$$

For a linear function x Equation (8.11) gives

$$S_1 = a \cdot 0 + b \cdot h + c \cdot 2h$$

Analytically we obtain

$$\int_0^{2h} x\,dx = \frac{x^2}{2}\Big]_0^{2h} = 2h^2$$

Thus

$$S_1 = bh + 2ch = 2h^2 \tag{8.13}$$

For a quadratic x^2 Equation (8.11) gives

$$S_2 = a \cdot 0 + b \cdot h^2 + c \cdot 4h^2$$

Analytically we obtain

$$\int_0^{2h} x^2\,dx = \frac{x^3}{3}\Big]_0^{2h} = \frac{8}{3}h^3$$

Thus

$$S_2 = bh^2 + 4ch^2 = \frac{8}{3}h^3 \tag{8.14}$$

Solving Equations (8.12), (8.13) and (8.14) for a, b and c we obtain

CamScanner

$$a = \frac{h}{3}, \quad b = \frac{4h}{3} \quad \text{and} \quad c = \frac{h}{3}$$

Thus Equation (8.11) may be written as

$$S = \frac{h}{3}(f_1 + 4f_2 + f_3) \tag{8.15}$$

The above equation is the same as Equation (8.9) which was obtained using a quadratic interpolation polynomial.

Incidentally it may be observed that Simpson's rule is exact for a cubic even though it was derived for a quadratic. This may be seen by applying Equation (8.15) to a cubic, namely, $f = x^3$. From (8.15) we obtain

$$S_3 = \frac{h}{3}(0 + 4h^3 + 8h^3) = 4h^4$$

Analytically we obtain

$$\int_0^{2h} x^3\,dx = 4h^4. \tag{8.16}$$

For a quartic, Equation (8.15) is not exact. The error is

$$\int_0^{2h} x^4\,dx - S_4 = \frac{32}{5}h^5 - \frac{h}{3}(0 + 4h^4 + 16h^4) = \frac{-4h^5}{15} \tag{8.17}$$

For higher order polynomials the error is given by higher powers of h.

Other rules similar to Simpson's rule may be derived based on fitting higher order polynomials through a large number of points. Such formulae have been derived in the literature and are called *Newton-Cotes* formulae. One may refer to the book by Ralston for these formulae.

8.5 Errors in Integration Formulae

Given a function $f(x)$ over an interval a valid problem would be to decide whether to apply trapezoidal rule, Simpson's rule or a rule based on fitting a higher order polynomial. Once this decision is made the next problem would be the choice of the interval h. In order to make these decisions it is necessary to obtain formulae for the truncation error. The error in using Trapezoidal rule may be derived as follows:

Expanding $f(x)$ in the interval $[0, h]$ in a Taylor series we get

$$f(x) = f(0) + xf'(0) + \frac{x^2}{2}f''(0) + \frac{x^3}{3!}f'''(0) + \ldots$$

$$\int_0^h f(x)\,dx = hf(0) + \frac{h^2}{2}f'(0) + \frac{h^3}{3!}f''(0) + \frac{h^4}{4!}f'''(0) + \ldots \tag{8.18}$$

$$a = \frac{h}{3}, \quad b = \frac{4h}{3} \quad \text{and} \quad c = \frac{h}{3}$$

Thus Equation (8.11) may be written as

$$S = \frac{h}{3}(f_1 + 4f_2 + f_3) \tag{8.15}$$

The above equation is the same as Equation (8.9) which was obtained using a quadratic interpolation polynomial.

Incidentally it may be observed that Simpson's rule is exact for a cubic even though it was derived for a quadratic. This may be seen by applying Equation (8.15) to a cubic, namely, $f = x^3$. From (8.15) we obtain

$$S_3 = \frac{h}{3}(0 + 4h^3 + 8h^3) = 4h^4$$

Analytically we obtain:

$$\int_0^{2h} x^3\,dx = 4h^4. \tag{8.16}$$

For a quartic, Equation (8.15) is not exact. The error is

$$\int_0^{2h} x^4\,dx - S_4 = \frac{32}{5}h^5 - \frac{h}{3}(0 + 4h^4 + 16h^4) = \frac{-4h^5}{15} \tag{8.17}$$

For higher order polynomials the error is given by higher powers of h.

Other rules similar to Simpson's rule may be derived based on fitting higher order polynomials through a large number of points. Such formulae have been derived in the literature and are called *Newton-Cotes* formulae. One may refer to the book by Ralston for these formulae.

8.5 Errors in Integration Formulae

Given a function $f(x)$ over an interval a valid problem would be to decide whether to apply trapezoidal rule, Simpson's rule or a rule based on fitting a higher order polynomial. Once this decision is made the next problem would be the choice of the interval h. In order to make these decisions it is necessary to obtain formulae for the truncation error. The error in using Trapezoidal rule may be derived as follows:

Expanding $f(x)$ in the interval $[0, h]$ in a Taylor series we get

$$f(x) = f(0) + xf'(0) + \frac{x^2}{2}f''(0) + \frac{x^3}{3!}f'''(0) + \ldots$$

$$\int_0^h f(x)\,dx = hf(0) + \frac{h^2}{2}f'(0) + \frac{h^3}{3!}f''(0) + \frac{h^4}{4!}f'''(0) + \ldots \tag{8.18}$$

CamScanner