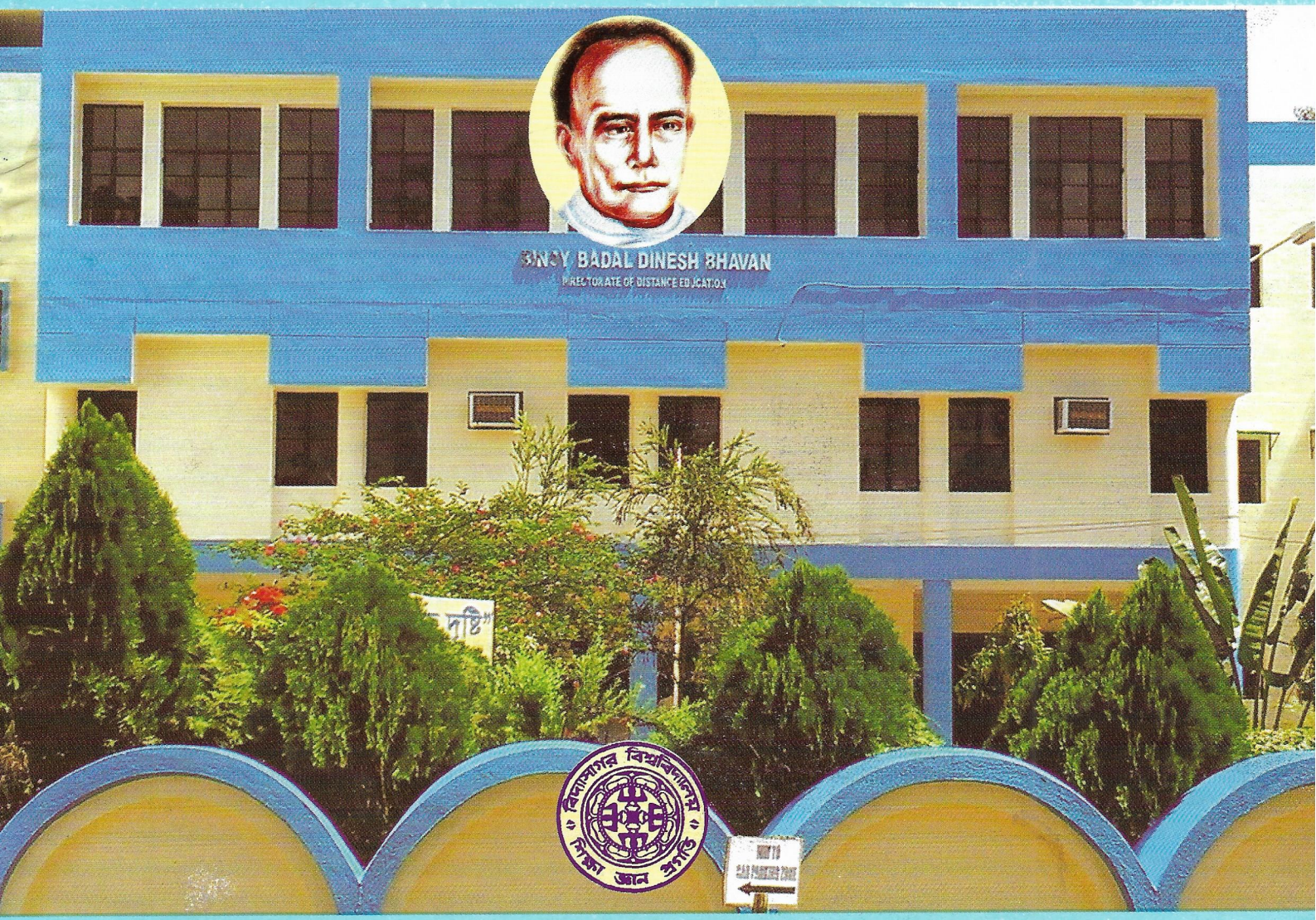
DISTANCE LEARNING MATERIAL



VIDYASAGAR UNIVERSITY

DIRECTORATE OF DISTANCE EDUCATION

MIDNAPORE- 721 102

M. SC. IN APPLIED MATHEMATICS WITH OCEANOLOGY & COMPUTOR PROGRAMMING

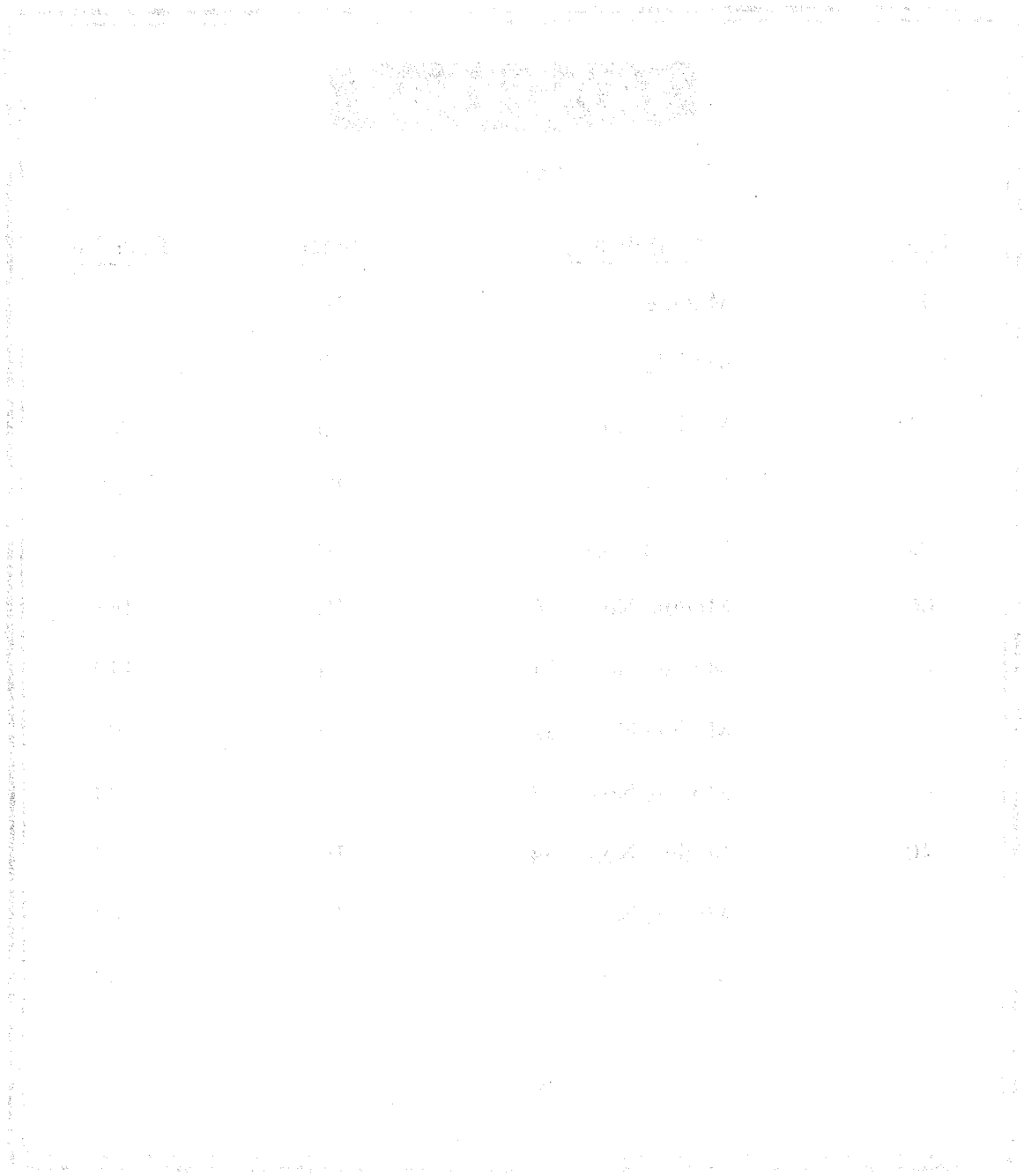
PART - I

Paper : III

Module No. : 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35 & 36

CONTENTS

PART - I



M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III Group-A

Module No. - 25

PROBABILITY AND STATISTICS

(Stochastic Process : Markov Chains)

CONTENT :

The probability models are more realistic than deterministic models in many problems in different branches of sciences, huminities, engineering etc. Observations taken at different time points rather than those taken at a fixed

period of time began to engage the attention of probabilities. This led to a new concept of indeterminism in dynamic studies. This has been called dynamic indeterminism. Many situations occur in physical and life sciences are studied now not only as a random phenomenon but also as one changing with time or space. Similar considerations are also made in other areas, such as, social sciences, engineering and management and so on. The scope of applications of random variables which are functions of time or space or both has been on the increase.

Objectives:

- Stochastic process
- Markov chain
- Transition probability matrix
- Random walk with absorbing and reflecting barriers
- Probability distribution
- Order of Markov chain
- Markov chain as graphs
- Chapman-Kolmogorov equation
- Classification of states and chains
- Closed state
- Irreducible chain
- Persistent and transient states
- Exercise

1.1 Stochastic Process

Families of random variables $\{x(t), t \in T\}$ where T is index set which are functions of say, time are known as stochastic processes (or random processes or random functions).

Example 1.1.1 Consider a simple experiment like throwing a fair die. Suppose that X_n is the outcome of the nth throw, $n \geq 1$. Then $\{X_n, n \geq 1\}$ is a family of random variables such that for a distinct value of $n(= 1,2,...)$, one gets a distinct random variable X_n; $\{X_n, n \geq 1\}$ constitutes a stochastic process, known as Bernoulli process.

Example 1.1.2 Consider a random event occurring in time, such as, number of telephone calls received at a switch board. Suppose that $X(t)$ is the random variable which represents the number of incoming calls in an interval $(0,t)$ of duration t units. The number of calls within a fixed interval of specified duration, say, one unit of time, is a random

variable $X(1)$ and the family $\{X(t), t \in T\}$ constitues a stochastic process $T = [0, \infty)$.

1.2 Definitions

Let S_t be the sample space of $X(t)$, which may be finite or infinite. The random variables $X_t, X_{t+r} (r > 0)$ may be dependent or independent. Also, T may be finite or infinite then the stochastic process $\{X(t), t \in T\}$ is a discrete-parameter stochastic process. If T is any finite or infinite interval e.g. $T = \{t : a < t < b\}, T = \{t : 0 \le t < \infty\}$, the process is said to be a continuous parameter stochastic process.

The particular value of $X(t)$ is often called state and the set of all possible values of a single random variable $X(t)$ of a stochastic process $\{X(t), t \in T\}$ is known as its state space.

The system is defined for a continuous range of time and we say that we have a family of r.v. in continuous time. A stochastic process in continuous time may have either a discrete or a continuous state space. e.g., suppose that $X(t)$ gives the number of incoming calls at a switchboard in an interval $(0, t)$. Here the state space of $X(t)$ is discrete though $X(t)$ is defined for a continuous range of time.

We have a process in continuous time having a discrete state space. Suppose that $X(t)$ represents the maximum temperature at a particular place in $(0, t)$, then the set of possible values of $X(t)$ is continuous. Here we have a system in continuous time a continuous state space.

We have assumed that the values assumed by the random variable $X(t)$ are one-dimensional, but the process $\{X(t)\}$ may be multi-dimensional. Consider $X(t) = (X_1(t), X_2(t))$, where X_1 represents the maximum and X_2 that minimum temperature at a place in an interval of time $(0, t)$. This is a two-dimensional stochastic process in continuous time having continuous state space. One can similarly have multi-dimenstional process. One-dimensional processes can be classified, in general, into the following four types of processes :

(i) Discrete time, discrete state space.

(ii) Discrete time, continuous state space.

(iii) Continuous time, discrete state space.

(iv) Continuous time, Continuous state space.

1.3 Markov Chains

Let us consider a simple coin tossing experiment repeated for a number of times. The possible outcomes at

each trial are two : head with probability say, p and tail with probability q, $p+q=1$. Let us denote head by 1 and tail by 0 and the random variable denoting the result of the nth toss by X_n. Then for $n=1, 2, 3, \ldots\ldots$

$$P(X_n = 1) = p, P(X_n = 0) = q.$$

Thus we have a sequence of random variables $X_1, X_2, \ldots\ldots$ The trials are independent and the result of the nth trial does not depend in any way on the previous trials numbered $1, 2, \ldots, n-1$. The random variables are independent.

Consider now the random variable given by the partial sum $S_n = X_1 + X_2 + \ldots\ldots + X_n$. The sum S_n gives the accumulated number of heads in the first n trials and its possible values are $0, 1, 2, \ldots\ldots, n$.

We have $S_{n+1} = S_n + X_{n+1}$. Given that $S_n = j, j = 0, 1, 2, \ldots, n$, the random variable S_{n+1} can assume only two possible values : $S_{n+1} = j$ with probability q and $S_{n+1} = j+1$ with probability p; these probabilities are not at all affected by the values of the variables $S_1, S_2, \ldots\ldots, S_{n-1}$.

Thus

$$P(S_{n+1} = j+1 / S_n = j) = p$$

$$P(S_{n+1} = j / S_n = j) = q.$$

This is an example of Markov Chain, a case of simple dependence that the outcome of $(n+1)$st trial depends directly on that of nth trial and only on it. The conditional probability of S_{n+1} given by S_n depends on the value of S_n and the manner in which the value of S_n was reached is of no consequence.

Now we present the formal definition of Markov chain.

Definition The stochastic process $\{X_n, n = 0,1,2,\ldots\ldots\}$ is called a Markov chain, if for $j, k, i_1, i_2, \ldots, i_{n-1} \in N$ (the set of natural numbers),

$$P(X_n = k / X_{n-1} = j, X_{n-2} = i_1, \ldots, X_0 = i_{n-1})$$

$$= P(X_n = k / X_{n-1} = j) = p_{jk} \text{ (say)}, \qquad (1.1)$$

whenever the first member is defined.

The outcomes are called the states of the Markov Chain. If X_n has outcome j, i.e. $(X_n = j)$, the process is said to be at state j at nth trial. To a pair of states (j, k) at the two successive trials (say, nth and $(n+1)$st trials) there

is an associated conditional probability p_{jk}. It is the probability of transition from the state j at nth trial to the state k at $(n+1)$ st tiral. The transition probabilities p_{jk} are basic to the study of the structure of the Markov Chain.

The transition probability may or may not be independent of n. If the transition probability p_{jk} is independent of n, the Markov Chain is said to be homogeneous (or to have stationary transition probabilities). If it is dependent on n, the chain is said to be non-homogeneous. The transition probability p_{jk} refers to the state (j, k) at two successive trials (say, nth and $(n+1)$st trial), the transition is one-step and p_{jk} is called one-step or unit step transition probability.

In general case, if the states (j, k) at two non-successive trials, say, state j at the nth trial and state k at the $(n+m)$ th trial. The corresponding transition probability is then called m-step transition probability and is denoted by $p_{jk}^{(m)}$ i.e.,

$$p_{jk}^{(m)} = P(X_{n+m} = k / X_n = j). \tag{1.2}$$

1.3.1 Transition probability matrix

It may be noted that the transition probabilities p_{jk} satisfy

$$p_{jk} \geq 0, \sum_k p_{jk} = 1 \text{ for all } j. \tag{1.3}$$

These probabilities may be written in the matrix form as

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & \cdots \\ p_{21} & p_{22} & p_{23} & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}. \tag{1.4}$$

This matrix is called the transition probability matrix or matrix of transition probabilities (t.p.m) of the Markov Chain. The matrix P is a stochastic matrix, i.e. a square matrix with non-negative elements and unit row sums.

Example 1.3.1 Random Walk with absorbing barriers (Gambler's ruin problem).

A particle performs a random walk with absorbing barriers, say, at 0 and k. Whenever it is at any position r $(0<r<k)$, it moves to $r+1$ with probability p or to $(r-1)$ with probability q, $p+q=1$. But, as soon as it reaches 0 or k it remains there itself. Let X_n be the position of the particle after n moves. The different states of X_n are the

different positions of the particle. $\{X_n\}$ is a Markov Chain whose unit-step transition probabilities are given by

$$\left.\begin{aligned} P(X_n = r+1 / X_{n-1} = r) &= p \\ P(X_n = r-1 / X_{n-1} = r) &= q \end{aligned}\right\} 0 < r < k$$

$$P(X_n = 0 / X_{n-1} = 0) = 1$$

$$P(X_n = k / X_{n-1} = k) = 1$$

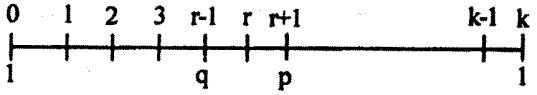


Fig. 1.1

and $P(X_n = r / X_{n-1} = r) = 0,\ 0 < r < k.$

The transition matrix is given by

States of X_n

States of X_{n-1}	0	1	2	3	...0	$k-1$	k
0	1	0	0	0	...0	0	0
1	q	0	p	0	...0	0	0
2	0	q	0	p	...0	0	0
3	0	0	q	0	...0	0	0
⋮	...	...	...	...	...	...	...
$k-1$	0	0	0	0	...q	0	p
k	0	0	0	0	...0	0	1

Table 1.1 : Transition probability matrix of Gambler's ruin problem.

Example 1.3.2 Random Walk between reflecting barriers

Consider that a particle may be at any position $r, r = 0,1,\ldots,k(\geq 1)$ of the x-axis. From state r it moves to state $r+1, 1 \leq r \leq k-1$ with probability p and to state $r-1$ with probability q. As soon as it reaches state 0 it remains there with probabillity a and is reflected to state 1 with probability $1-a(0 < a < 1)$; if it reaches state k it remains there with probability b and is reflected to $k-1$ with probability $1-b$ $(0<b<1)$. Then $\{X_n\}$, where X_n is the position of the particle after n steps or moves, is a Marcov Chain with state space $S = \{0,1,\ldots..,k\}$. The transition probabilities are given by

$$\left.\begin{aligned} P(X_n = r+1 / X_{n-1} = r) &= p \\ P(X_n = r-1 / X_{n-1} = r) &= q \\ P(X_n = r / X_{n-1} = r) &= 0 \end{aligned}\right\} 0 < r < k.$$

$$P(X_n = 0 / X_{n-1} = 0) = a$$

$$P(X_n = 1 / X_{n-1} = 0) = 1-a$$

$$P(X_n = k-1 / X_{n-1} = k) = b$$

$$P(X_n = k / X_{n-1} = k) = 1-b.$$

The transition matrix is shown below

$$P = \begin{array}{c} \\ \text{States of } X_{n-1} \end{array} \begin{array}{c} \text{States of } X_n \\ \begin{array}{cccccccc} & 0 & 1 & 2 & .... & k-1 & k \\ 0 & a & 1-a & 0 & ...0 & 0 & 0 \\ 1 & q & 0 & p & ...0 & 0 & 0 \\ \vdots & ... & ... & ... & ... & ... & ... \\ k-1 & 0 & 0 & 0 & q & 0 & p \\ k & 0 & 0 & 0 & 0 & 1-b & b \end{array} \end{array}$$

Table 1.2 : Transition matrix for random walk between reflecting barriers.

Note : If a =1, then 0 is an absorbing barrier and if a=0 then 0 is a reflecting barrier, if 0<a<1, 0 is an elastic barrier. Similar is the case with state k. The case when both 0 and k are absorbing barriers corrsponds to the familiar Gambler's ruin problem (with total capital between the two gamblers is k).

Example 1.3.3. Partial sum of independent random variables :

Consider a series of coin tossing experiments, where the outcomes of nth trial are denoted by 1 (for a head) and 0 (for a tail). Let X_n be the random variable denoting the outcome of nth trial and $S_n = X_1 + X_2 + ... + X_n$ be the nth partial sum. The possible values of S_n are 0, 1, ..., n i.e., the states of S_n are r, r =0, 1, ..., n, $\{S_n, n \geq 0\}$ is a Markov chain with transition matrix is given below

$$P = \text{States of } X_{n-1} \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ r-1 \\ r \\ \vdots \end{array} \begin{array}{c} \text{States of } S_n \\ \begin{array}{ccccccc} 0 & 1 & 2 & \dots & r-1 & r & \dots \end{array} \\ \begin{bmatrix} q & p & 0 & \dots & 0 & 0 & \dots \\ 0 & q & p & \dots & 0 & 0 & \dots \\ 0 & 0 & q & \dots & 0 & 0 & \dots \\ & & & & & & \\ 0 & 0 & 0 & \dots & q & p & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \end{array}$$

Mathematically,

$$p_{jk} = P(S_n = k / S_{n-1} = j)$$

$$= \begin{cases} p, & k = j+1 \\ q, & k = j \\ 0, & \textit{otherwise.} \end{cases}$$

1.3.2 Probability Distribution

The probability distribution of the random variables $X_r, X_{r+1}, \dots\dots, X_{r+n}$ can be computed in terms of the transition probabilities p_{jk} and the initial distribution of X_r. Suppose $r = 0$, then

$$P(X_0 = a, X_1 = b, \dots\dots, X_{n-2} = i, X_{n-1} = j, X_n = k)$$

$$= P(X_n = k / X_{n-1} = j, \dots\dots, X_0 = a)\, P(X_{n-1} = j, \dots\dots, X_0 = a)$$

$$= P(X_n = k / X_{n-1} = j)\, P(X_{n-1} = j, \dots\dots, X_0 = a)$$

[Since $\{X_n\}$ is a Markov Chain.]

$$= P(X_n = k / X_{n-1} = j)\, P(X_{n-1} = j / X_{n-2} = i)\, P(X_{n-2} = i, \dots\dots, X_0 = a)$$

$$= P(X_n = k / X_{n-1} = j)\, P(X_{n-1} = j / X_{n-2} = i) \dots P(X_1 = b / X_0 = a)\, P(X_0 = a)$$

$$= P(X_0 = a)\, p_{ab} \dots\dots p_{ij}\, p_{jk}.$$

Thus

$$P(X_r = a, X_{r+1} = b, \ldots, X_{r+n-2} = i, X_{r+n-1} = j, X_{r+n} = k)$$
$$= P(X_r = a)\, p_{ab} \ldots p_{ij}\, p_{jk}.$$

Example 1.3.4 Let $\{X_n, n \geq 0\}$ be a Marmov Chain with three states 0, 1, 2 and with transition matrix

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 3/4 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 \\ 0 & 3/4 & 1/4 \end{bmatrix} \end{array}$$ and the initial distribution $P(X_0 = i) = 1/3, i = 0,1,2$

We have

$$P(X_1 = 1 / X_0 = 2) = 3/4$$

$$P(X_2 = 2 / X_1 = 1) = 1/4$$

$$P(X_2 = 2, X_1 = 1 / X_0 = 2) = P(X_2 = 2 / X_1 = 1)\, P(X_2 = 1 / X_0 = 2)$$

$$= \frac{1}{4} \cdot \frac{3}{4} = \frac{3}{16}.$$

$$P(X_2 = 2, X_1 = 1, X_0 = 2) = P(X_2 = 2, X_1 = 1 / X_0 = 2)\, P(X_0 = 2)$$

$$= \frac{3}{16} \cdot \frac{1}{3} = \frac{1}{16}.$$

$$P(X_3 = 1, X_2 = 2, X_1 = 1, X_0 = 2)$$
$$= P(X_3 = 1 / X_2 = 2, X_1 = 1, X_0 = 2)\, P(X_2 = 2, X_1 = 1, X_0 = 2)$$
$$= P(X_3 = 1 / X_2 = 2) \cdot \frac{1}{16} = \frac{3}{4} \cdot \frac{1}{16} = \frac{3}{64}.$$

Note. The matrix of transition probabilities together with the initial distribution completely specifies a Markov Chain $\{X_n, n = 0,1,2,\ldots\}$.

Theorem 1.3.1 (General existence theorem of Markov Chains)

Given the set N and the sequence of stochastic matrices ${}^{(n)}P$, there exists a Markov Chain $\{X_n, n \geq 0\}$ with state space N and transition probability matrix ${}^{(n)}P$.

1.3.3 Strong Markov property

Let N be a stopping time (is also a random variables) for a Markov Chain $\{X_n, n > 0\}$ and let A and B be two events relating to X_n and happening, prior and posterior respectively to N. Then

$$P(B / X_N = i, A) = P(B / X_N = i). \tag{1.5}$$

This is called the strong Markov property. It shows that if N is a stopping time for a Markov Chain $\{X_n, n > 0\}$, then the evolution of the chain starts afresh from the state reached at time N.

Every discrete time Markov Chain $\{X_n, n \geq 0\}$ possesses the strong Markov property.

1.3.4 Order of Markov Chain

A Markov Chain $\{X_n\}$ is said to be of order s (s=1, 2, 3,.....) if for all n,

$$P(X_n = k / X_{n-1} = j, X_{n-2} = j_1, \ldots\ldots, X_{n-s} = j_{s-1}, \ldots\ldots)$$

$$= P(X_n = k / X_{n-1} = j, \ldots\ldots, X_{n-s} = j_{s-1}) \tag{1.6}$$

whenever the l.h.s is defined.

A Markov Chain $\{X_n\}$ is said to be of order one (or simply a Markov Chain) if

$$P(X_n = k / X_{n-1} = j, X_{n-2} = j_1, \ldots.)$$

$$= P(X_n = k / X_{n-1} = j) = p_{jk}$$

whenever $P(X_{n-1} = j, X_{n-2} = j_1, \ldots.) > 0$.

Unless explicity stated otherwise, we shall mean by Markov Chain, a chain of order one. A chain is said to be of order zero if $p_{ik} = p_k$ for all j. This implies independence of X_n and X_{n-1}.

1.3.5 Markov Chains as graphs

The states of a Markov Chain may be represented by the vertices of the graph and one step transitions between states by directed arcs, if $i \to j$, then vertices i and j are joined by a directed are with arrow towards j, the value of p_{ij} which is the weight of the directed arc (i,j). If $V = (1,2,\ldots,m)$ is the set of vertices corresponding the state space of the chain and E is the set of directed arcs between these vertices, then the graph $G=(V, E)$ is the directed graph or digraph or transition graph of the chain. A digraph such that its arc weights are positive and sum of the arc weights of the arcs from each node is unity is called a stochastic graph. The digraph or transition graph of a Markov Chain is a stochastic graph.

A transition graph is of great aid in visualizing a Markov chain, it is an useful tool in studying the properties of the chain. e.g., the graph of the Chain of the transition matrix

$$P = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 3/4 & 1/4 & 0 \\ 1/4 & 1/2 & 1/4 \\ 0 & 3/4 & 1/4 \end{bmatrix} \end{array} \tag{1.7}$$

is

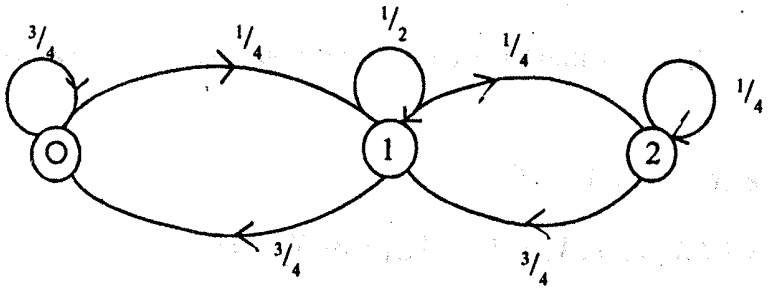


Fig. 1.2 : The transition graph of the Markov Chain of transition matrix (1.7).

1.4 Higher Transition Probabilites

The m-step transition probability is denoted by

$$P(X_{m+n} = k / X_n = j) = p_{jk}^{(m)}. \tag{1.8}$$

$p_{jk}^{(m)}$ gives the probability that from the state j at nth trial, the state k is reached at $(m+n)$th trial in m steps,

i.e. the probability of transition from the state j to the state k in exactly m steps. The number n does not occur in the r.h.s of (1.8) and the chain is called homogeneous otherwise it is called non-homogeneous.

Chapman-Kolmogorov Equation :

Let $p_{jk}^{(m)}$ be the m-step transition probability from state j to the state k. Then

$$p_{jk}^{(m+n)} = \sum_r p_{rk}^{(n)} p_{jr}^{(m)} = \sum_r p_{jr}^{(n)} p_{rk}^{(m)} \tag{1.9}$$

i.e., $P^{m+n} = P^n P^m = P^m P^n$, (1.10)

where $P^n = \left(p_{ij}^{(n)}\right)$.

Proof. The m-step transition probability is defined as

$$P\left(X_{m+n} = k / X_n = j\right) = p_{jk}^{(m)}.$$

The one-step transition probabilities $p_{jk}^{(1)}$ are denoted by p_{jk}. The 2-step transition probability $p_{jk}^{(2)}$ is given by

$$p_{jk}^{(2)} = P\left(X_{n+2} = k / X_n = j\right).$$

The state k can be reached from the state j in two steps through some intermediate state r. Consider a fixed value of r, we have

$$P\left(X_{n+2} = k, X_{n+1} = r / X_n = j\right)$$

$$= P\left(X_{n+2} = k / X_{n+1} = r, X_n = j\right) P\left(X_{n+1} = r / X_n = j\right)$$

$$= p_{rk}^{(1)} p_{jr}^{(1)} = p_{jr} p_{rk}.$$

Since these intermediate states r can assume the values r=1, 2, ..., we have

$$p_{jk}^{(2)} = P\left(X_{n+2} = k / X_n = j\right) = \sum_r P\left(X_{n+2} = k, X_{n+1} = r / X_n = j\right)$$

$$= \sum_r p_{jr} p_{rk} \text{ (summing over for all the intermediate states).}$$

By similar method,

$$p_{jk}^{(m+1} = P\left(X_{m+n+1} = k / X_n = j\right)$$

$$= \sum_r P\left(X_{+m+1} = k / X_{n+m} = r\right) P\left(X_{n+m} = r / X_n = j\right)$$

$$= \sum_r p_{rk}\, p_{jr}^{(m)}.$$

Similarly, we get

$$p_{jk}^{(m+1)} = \sum_r p_{jr} p_{rk}^{(m)}.$$

In general, we have

$$p_{jk}^{(m+n)} = \sum_r p_{rk}^{(n)} p_{jr}^{(m)} = \sum_r p_{jr}^{(n)} p_{rk}^{(m)}. \tag{1.11}$$

Denoting by $P^n = \left(p_{ij}^{(n)}\right)$ the matrix of n-step transition probabilities, the equation (1.11) can be written as

$$P^{m+n} = P^n P^m. \tag{1.12}$$

The equations (1.11) and (1.12) are called Chapman - Kolmogorov equations and these characterise Markov Chains.

Example 1.4.1 Suppose that probability of a dry day (state 0) following a rainy day (state 1) is $\frac{1}{3}$ and that the probability of a rainy day following a dry day is $\frac{1}{2}$ and transition probability matrix is

$$p = \begin{matrix} \\ 0 \\ 1 \end{matrix} \begin{matrix} \begin{matrix} 0 & 1 \end{matrix} \\ \begin{bmatrix} 1/2 & 1/2 \\ 1/3 & 2/3 \end{bmatrix} \end{matrix}.$$

If May 10 is a dry day then waht are the probabilities that May 12 and May 14 are dry days?

Solution : We have $P = \begin{pmatrix} 1/2 & 1/2 \\ 1/3 & 2/3 \end{pmatrix}$.

Then $P^2 = P \cdot P = \begin{pmatrix} 5/12 & 7/12 \\ 7/18 & 11/18 \end{pmatrix}, P^4 = \begin{pmatrix} 173/432 & 259/432 \\ 259/648 & 389/648 \end{pmatrix}$.

If May 10 is a dry day, the probability that May 12 a dry day is $p_{00}^{(2)} = 5/12$ and that May 14 a dry day is $p_{00}^{(4)} = 173/432$.

Example 1.4.2 Consider a communication system which transmits the two digits 0 and 1 through several stages. Let $X_n, n \geq 1$ be the digit leaving the nth stage of system and X_0 be the digit entering the first stage (leaving the 0th stage). At each stage there is a constant probability q that the digit which enters will be transmitted unchanged (i.e., the digit will remain unchanged when it leaves), and probability p otherwise (i.e. the digit changes when it leaves), $p+q=1$. Find the one-step transition matrix P, and n–step transition matrix P^n. Also find P^n when $n \to \infty$.

Solution. Here $\{X_n, n \geq 0\}$ is a homogeneous two-state Markov Chain with unit - step transition matrix

$$P = \begin{pmatrix} q & p \\ p & q \end{pmatrix}.$$

$$P = \begin{pmatrix} q & p \\ p & q \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(p+q)+\frac{1}{2}(q-p) & \frac{1}{2}(p+q)-\frac{1}{2}(q-p) \\ \frac{1}{2}(p+q)+\frac{1}{2}(q-p) & \frac{1}{2}(p+q)+\frac{1}{2}(q-p) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p) & \frac{1}{2}-\frac{1}{2}(q-p) \\ \frac{1}{2}-\frac{1}{2}(q-p) & \frac{1}{2}+\frac{1}{2}(q-p) \end{pmatrix}.$$

Now, $P^2 = \begin{pmatrix} q & p \\ p & q \end{pmatrix}\begin{pmatrix} q & p \\ p & q \end{pmatrix} = \begin{pmatrix} p^2+q^2 & 2pq \\ 2pq & p^2+q^2 \end{pmatrix}$

$$= \begin{pmatrix} \frac{1}{2}(q+p)^2+\frac{1}{2}(q-p)^2 & \frac{1}{2}(q+p)^2-\frac{1}{2}(q-p)^2 \\ \frac{1}{2}(q+p)^2-\frac{1}{2}(q-p)^2 & \frac{1}{2}(q+p)^2+\frac{1}{2}(q-p)^2 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p)^2 & \frac{1}{2}-\frac{1}{2}(q-p)^2 \\ \frac{1}{2}-\frac{1}{2}(q-p)^2 & \frac{1}{2}+\frac{1}{2}(q-p)^2 \end{pmatrix}$$

Let $P^m = \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p)^m & \frac{1}{2}-\frac{1}{2}(q-p)^m \\ \frac{1}{2}-\frac{1}{2}(q-p)^m & \frac{1}{2}+\frac{1}{2}(q-p)^m \end{pmatrix}$

Now, $P^{m+1} = \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p)^m & \frac{1}{2}-\frac{1}{2}(q-p)^m \\ \frac{1}{2}-\frac{1}{2}(q-p)^m & \frac{1}{2}+\frac{1}{2}(q-p)^m \end{pmatrix} \begin{pmatrix} q & p \\ p & q \end{pmatrix}$

$$= \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p)^{m+1} & \frac{1}{2}-\frac{1}{2}(q-p)^{m+1} \\ \frac{1}{2}-\frac{1}{2}(q-p)^{m+1} & \frac{1}{2}+\frac{1}{2}(q-p)^{m+1} \end{pmatrix}.$$

This is true for $n=m+1$. Hence by mathematical induction

$$P^n = \begin{pmatrix} \frac{1}{2}+\frac{1}{2}(q-p)^n & \frac{1}{2}-\frac{1}{2}(q-p)^n \\ \frac{1}{2}-\frac{1}{2}(q-p)^n & \frac{1}{2}+\frac{1}{2}(q-p)^n \end{pmatrix}.$$

Here $p_{00}^{(n)} = p_{11}^{(n)} = \frac{1}{2}+\frac{1}{2}(q-p)^n$

and $p_{10}^{(n)} = p_{01}^{(n)} = \frac{1}{2}-\frac{1}{2}(q-p)^n$.

As $n \to \infty, p_{11}^{(n)} = p_{00}^{(n)} = \frac{1}{2}$ and $p_{01}^{(n)} = p_{10}^{(n)} = \frac{1}{2}$

[as $q-p<1$].

1.5 Classification of States and Chains

1.5.1 Communication relations

If $p_{ij}^{(n)} > 0$ for some $n \geq 1$, then we say that state j can be reached or state j is accessible from state i. This relation is denoted by $i \to j$. Conversely, if for all n, $p_{ij}^{(n)} = 0$, then j is not accessible from i and it is denoted by $i \not\to j$.

If two states i and j are such that each is accessible from the other then we say that the two states communicate and it is denoted by $i \leftrightarrow j$. Then there exist integers $m(\geq 1)$ and $n(\geq 1)$ such that $p_{ij}^{(n)} > 0$ and $p_{ji}^{(m)} > 0$,

Obviously, the relation $\leftrightarrow$ is symmetric.

Theorem 1.5.1 The communicate relation $\leftrightarrow$ is transitive.

Proof. From Chapman - Kolmogorov equation

$$p_{ik}^{(m+n)} = \sum_r p_{ir}^{(m)} p_{rk}^{(n)}$$

or, $p_{ik}^{(m+n)} \geq p_{ij}^{(m)} p_{jk}^{(n)}$.

If $i \to j$ and $j \to k$ then $p_{ij}^{(m)} > 0$ and $p_{jk}^{(n)} > 0$.

Then $p_{ik}^{(m+n)} > 0$, implies $i \to k$.

Again $p_{ki}^{(m+n)} \geq p_{kj}^{(m)} p_{ji}^{(n)}$.

If $k \to j$ and $j \to i$ then by similar way $p_{ki}^{(m+n)} > 0$.

Hence $i \leftrightarrow j$ and $j \leftrightarrow k$ implies $i \leftrightarrow k$, i.e., $\leftrightarrow$ is transitive.

1.5.2 Class property

A class of states is a subset of the state space such that every state of the class communicates with every other and there is no other state outside the class which communicates with all other states in the class. A property defined for all states of a chain is a class property if its possession by one state in a class implies its possession by all states of the same class.

The state i is a return state if $p_{ii}^{(n)} > 0$ for some $n \geq 1$.

The period d_i of a return to state i is defined as the greatest common divisor of all m such that $p_{ii}^{(m)} > 0$ i.e.

$$d_i = G.C.D.\{m: p_{ii}^{(m)} > 0\}.$$

[G.C.D. means greatest common divisor.]

This state i is said to be aperiodic if $d_i = 1$ and periodic if $d_i > 1$. Clearly, state i is aperiodic if $p_{ii} \neq 0$.

Closed state : If C is a set of states such that no state outside C can be reached from any state in C, then C is said to be closed.

If C is closed and $j \in C$ while $k \notin C$, then $p_{jk}^{(n)} = 0$ for all n, i.e. C is closed iff $\sum_{j \in C} p_{ij} = 1$ for every $i \in C$.

A closed state may contain one or more states. If a closed set contains only one state j then state j is said to be absorbing i.e. j is absorbing iff $p_{jj} = 1, p_{jk} = 0, k \neq j$.

In Gambler's ruin problem states 0 and k are absorbing.

Every finite Markov Chain contains at least one closed set i.e. the set of all states or the state space.

Irreducible Chain : If the chain does not contain any other proper closed subset other than the state space, then the chain is called irreducible; the t.p.m. of irreducible chain is an irreducible matrix.

In an irreducible Markov chain every state can be reached from every other state.

Chains which are not irreducible are said to be reducible or non-irreducible, the t.p.m. is reducible.

Notations :

Let $f_{jk}^{(n)}$ be the probability that it reaches from the state j to the state k for the first time at the nth step (or after n transitions) and let $p_{jk}^{(n)}$ be the probability that it reaches state k (not necessarily for the first time) after n transitions.

Let F_{jk} denote the probability that starting with state j the system will ever reach state k.

That is, $F_{jk} = \sum_{n=1}^{\infty} f_{jk}^{(n)}$. (1.13)

The mean time from state j to state k is given by

$$\mu_{jk} = \sum_{n=1}^{\infty} n f_{jk}^{(n)}. \tag{1.14}$$

The mean recurrence time for the state j is

$$\mu_{jj} = \sum_{n=1}^{\infty} n f_{jj}^{(n)}. \tag{1.15}$$

Theorem 1.5.2 (First Entrance Theorem).

For any states j and k

$$p_{jk}^{(n)} = \sum_{r=0}^{n} f_{jk}^{(r)} p_{kk}^{(n-r)}, n \geq 1 \text{ with} \tag{1.16}$$

$$p_{kk}^{(0)} = 1, f_{jk}^{(0)} = 0, f_{jk}^{(1)} = p_{jk}.$$

Proof. We have $p_{jk}^{(n)}$ = probability that the system will pass from state j to state k in n steps

$$= \sum_{r=0}^{n} P \text{ (first return to } k \text{ occurs at } r\text{th step from state } j)$$

$\times P$ (the system is in k at nth step/ it was in state k at the rth step)

$$= \sum_{r=0}^{n} f_{jk}^{(r)} P\left(X_n = k / X_r = k\right)$$

$$= \sum_{r=0}^{n} f_{jk}^{(r)} p_{kk}^{(n-r)}$$

Therefore,

$$p_{jk}^{(n)} = \sum_{r=0}^{n} f_{jk}^{(r)} p_{kk}^{(n-r)}.$$

Note :

The above result can also be written as

$$p_{jk}^{(n)} = \sum_{r=0}^{n-1} f_{jk}^{(r)} p_{kk}^{(n-r)} + f_{jk}^{(n)} \left[\because p_{kk}^{(0)} = 1\right]$$

or, $$f_{jk}^{(n)} = p_{jk}^{(n)} - \sum_{r=0}^{n-1} f_{jk}^{(r)} p_{kk}^{(n-r)}. \tag{1.17}$$

1.5.3 Transient and Persistent (recurrent) states.

A state j is said to be persistent (or recurrent) if return to j is certain, i.e. $F_{jj} = 1$ (i.e., return to state j is certain).

A state j is called transient (or non-recurrent) if return to j is un-certain, i.e., $F_{jj} < 1$.

A persistent state j is said to be null-persistent if $\mu_{jj} = \infty$, i.e., if the mean recurrence time is infinite, and is said to be non-full (or positive) persistent if $\mu_{jj} < \infty$.

A persistent state j is called ergodic if it is not a null-state and is non-periodic.

Note. If the state j is transient then $p_{jj}^{(n)} \to 0$ as $n \to \infty$ because the return to j is uncertain.

Example 1.5.1 Let $\{X_n, n \geq 0\}$ be a Markov chain having state space $S = \{1, 2, 3, 4\}$ and transition matrix

$$P = \begin{pmatrix} 1/3 & 2/3 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 1/2 \end{pmatrix}.$$

Identify the states as transient, persistent, ergodic.

Solution : Here $f_{11}^{(1)} = \frac{1}{3}, f_{11}^{(2)} = \frac{2}{3}$,

$f_{11}^{(n)} = 0, n \geq 3$ and $F_{11} = \frac{1}{3} + \frac{2}{3} = 1$. So that state 1 is persistent.

$f_{33}^{(1)} = \frac{1}{2}, f_{33}^{(n)} = 0, n \geq 2.$

So $F_{33} = \frac{1}{2} < 1$. Therefore the state 3 is transient.

$f_{44}^{(1)} = \frac{1}{2}, f_{44}^{(n)} = 0,\ n \geq 2$ so that $F_{44} = \frac{1}{2} < 1$. Thus state 4 is also transient.

Further, $\mu_{11} = 1 \cdot \frac{1}{3} + 2 \cdot \frac{2}{3} = \frac{5}{3}$. The state is non-null persistent. Again $p_{11} = \frac{1}{3} > 0$, so that state 1 is aperiodic, state 1 is ergodic.

$$f_{22}^{(1)} = 0, f_{22}^{(2)} = 1 \cdot \frac{2}{3}, f_{22}^{(3)} = 1 \cdot \frac{1}{3} \cdot \frac{2}{3}, f_{22}^{(4)} = 1 \cdot \left(\frac{1}{3}\right)^2 \cdot \frac{2}{3}$$

$$\ldots\ldots, f_{22}^{(n)} = 1 \cdot \left(\frac{1}{3}\right)^{n-2} \cdot \frac{2}{3}, n \geq 2.$$

$$\therefore F_{22} = \sum_{n=1}^{\infty} f_{22}^{(n)} = \sum_{n=2}^{\infty} \left(\frac{1}{3}\right)^{n-2} \cdot \frac{2}{3} = 1.$$

Thus state 2 is persistent.

Now, $\mu_{22} = \sum_{k=1}^{\infty} k f_{22}^{(k)} = \sum_{k=1}^{\infty} k \left(\frac{1}{3}\right)^{k-2} \cdot \frac{2}{3} = 2 \sum_{k=2}^{\infty} k \left(\frac{1}{3}\right)^{k-1} = \frac{5}{2}$

Therefore, state 2 is non-ull persistent. It is also aperiodic and hence ergodic.

Theorem 1.5.3 In an irreducible chain, all the states are of the same type. They are either all transient, all persistent null, or all persistent non-null. All the states are aperiodic and in the latter case they all have the same period.

Proof. Since the chain is irreducible, every state can be reached from every other state. If i, j are any two states, then i can be reached from j and j from i, i.e.

$$p_{ij}^{(N)} = \alpha > 0 \text{ for some } N \geq 1$$

and $p_{ji}^{(N)} = \beta > 0$ or some $M \geq 1$.

We have

$$p_{jk}^{(m+n)} = \sum_r p_{jr}^{(m)} p_{rk}^{(n)} \geq p_{jr}^{(m)} p_{rk}^{(n)} \text{ for each } r.$$

Hence $p_{ii}^{(n+N+M)} \geq p_{ij}^{(n)} p_{jj}^{(n)} p_{ji}^{(M)} = \alpha\beta\, p_{jj}^{(n)}$ (1.18)

and $p_{jj}^{(n+N+M)} \geq p_{ji}^{(M)} p_{ii}^{(n)} p_{ij}^{(N)} = \alpha\beta\, p_{ii}^{(n)}$. (1.19)

From above it is clear that the two series $\sum_n p_{ii}^{(n)}$ and $\sum_n p_{jj}^{(n)}$ coverage or diverge together. Thus the two states i, j are either both transient or both persistent.

Suppose that i is persistent null, then $p_{ii}^{(n)} \to 0$ as $n \to \infty$, then from (1.18), $p_{jj}^{(n)} \to 0$ as $n \to \infty$, so that j

is also persistent null i.e., they are both persistent null.

Suppose that i is persistent non-null and has period t, then $p_{ii}^{(n)} > 0$ whenever n is a multiple of t. Now

$p_{ii}^{(N+M)} \geq p_{ij}^{(N)} p_{ji}^{(M)} = \alpha\beta > 0$, so that

$(N + M)$ is a multiple of t. From (1.19)

$p_{jj}^{(n+N+M)} \geq \alpha\beta p_{ii}^{(n)} > 0.$

Thus $(n + N + M)$ is a multiple of t and so t is the period of the state j also.

Theorem 1.5.4 State j is persistent iff

$$\sum_{n=0}^{\infty} p_{jj}^{(n)} = \infty. \tag{1.20}$$

Proof. Let $P_{jj}(s) = \sum_{n=0}^{\infty} p_{jj}^{(n)} s^n = 1 + \sum_{n=1}^{\infty} p_{jj}^{(n)} s^n, |s| \leq 1$

and $F_{jj}(s) = \sum_{n=0}^{\infty} f_{jj}^{(n)} s^n = \sum_{n=1}^{\infty} f_{jj}^{(n)} s^n, |s| \leq 1$

$[\because p_{jj}^{(0)} = 1$ and $f_{jj}^{(0)} = 0.]$

be the generating functions of the sequences

$\{p_{jj}^{(n)}\}$ and $\{f_{jj}^{(n)}\}$ respectively.

We know that

$$p_{jj}^{(n)} = \sum_{r=0}^{n} f_{jj}^{(r)} p_{jj}^{(n-r)}. \tag{1.21}$$

Multiplying both sides by s^n and adding for all $n \geq 1$, we get

$$\sum_{n=1}^{\infty} p_{jj}^{(n)} s^n = \sum_{n=1}^{\infty} \sum_{r=0}^{n} f_{jj}^{(r)} p_{jj}^{(n-r)} s^n$$

or, $$P_{jj}(s) - 1 = F_{jj}(s) P_{jj}(s). \tag{1.22}$$

The r.h.s. is immediately obtained by considering the fact that the r.h.s. of (1.21) is a convolution of $\{f_{jj}\}$

and $\{p_{jj}\}$ and that the generating function of the convolution is the product of the two generating functions. Thus we have

$$P_{jj}(s) = \frac{1}{1 - F_{jj}(s)}, |s| \leq 1. \tag{1.23}$$

If the state j is persistent then $F_{jj} = 1$.

$$\therefore \lim_{s \to 1} F_{jj}(s) = 1.$$

Thus $\lim_{s \to 1} P_{jj}(s) \to \infty$ or $\sum_{n=0}^{\infty} p_{jj}^{(n)} = \infty$.

Conversely, if the state is transient then

$$F_{jj} < 1, \text{ i.e., } \lim_{s \to 1} F_{jj}(s) < 1$$

or, $\lim_{s \to 1} P_{jj}(s) < \infty$, $\sum_{n=0}^{\infty} p_{jj}^{(n)} < \infty$.

Remarks :

1. State j is transient if $\sum p_{jj}^{(n)} < \infty$; this implies that if j is transient then $p_{jj}^{(n)} \to 0$ as $n \to \infty$.
2. The state space of a finite Markov Chian must contain at least one persistent state.
3. If k is a transient state and j is an arbitrary state then $\sum p_{jk}^{(n)}$ converges and $\lim_{n \to \infty} p_{jk}^{(n)} \to 0$.
4. If a Markov Chain having a set of transient states T, starts in a transient state, then with probability 1, it stays at the transient set of states T only a finite number of times after which it enters a recurrent state where it remains forever.

Example 1.5.2 Consider the Markov Chain with t.p.m.

$$P = \begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ ½ & 0 & ½ \\ 0 & 1 & 0 \end{bmatrix} \end{array}.$$

Test whether the states are periodic and persistent.

Solution : The chain is irreducible as the matrix P is irreducible.

We have $P^2 = P \cdot P = \begin{bmatrix} ½ & 0 & ½ \\ 0 & 1 & 0 \\ ½ & 0 & ½ \end{bmatrix}, P^3 = P.$

In general, $P^{2n} = P^2, P^{2n+1} = P.$

Therefore, $p_{ii}^{(2n)} > 0, p_{ii}^{(2n+1)} = 0$ for all i.

The states are periodic with period 2.

Also, $f_{11}^{(1)} = 0, f_{11}^{(2)} = 1. F_{11} = \sum_{n=1}^{\infty} f_{11}^{(n)} = 1,$ i.e. state 1 is persistent and hence the other states 0 and 2 are also persistent.

Now, $\mu_{11} = \sum_{n=1}^{\infty} n f_{11}^{(n)} = 2,$

i.e. state 1 is non-null. Thus the states of the chain are periodic and persistent non-null.

1.6 Markov Chains with Continuous State Space

We have discussed Markov Chains $\{X_n, n = 0,1,2,....\}$ with discrete state space, i.e. with $0, \pm 1, \pm 2, ...$ as possible values of X_n. Here we consider chains $\{X_n\}$ with continuous state space, i.e. with $(-\infty, \infty)$ as possible range of values of X_n. We shall have to use either probability distribution (d.f.) or probability density function (p.d.f), when this exists, in place of probability mass functions.

Definition : If for all m and for all possible values of X_m in $(-\infty, \infty)$

$$P(X_{m+1} \le x / X_m = y, X_{m-1} = y_1, ..., X_0 = y_m)$$

$$= P(X_{m+1} \le x / X_m = y) \quad (1.24)$$

then $\{X_m, m > 0\}$ is said to constitute a Markov Chain with continuous stape space. If the conditional d.f. as given by (1.24) is independent of m, then the Chain is homogeneous and equation (1.24) gives one-step transition probability d.f. More generally, the n-step transition probability d.f. is defined by

$$P(X_{m+n} \le x / X_m = y, X_{m-1} = y_1, \ldots, X_0 = y_m)$$

$$= P(X_{m+n} \le x / X_m = y)$$

$$= P_n(y; x). \tag{1.25}$$

Denote $P(y;x) = P_1(y;x) = P(X_{m+1} \le x / X_m = y)$

$= P(X_{n+1} \le x / X_n = y)$.

Let $P_n(x) = P(X_n \le x)$. The transition d.f. $P_n(y;x)$ and the initial distribution $P(X_0 \le x) = P_0(x)$ can unquiely determine $P_n(x)$. The Chapman – Kolmogorov equation takes the form

$$P_{n+m}(y;x) = \int_{-\infty}^{\infty} d_z P_n(y;z)\, P_m(z;x), m,n \ge 0 \tag{1.26}$$

which correspondens to $p_{jk}^{(n+m)} = \sum_s p_{js}^{(n)}\, p_{sk}^{(m)}$ for Markov Chains with discrete state space.

For n=1, equation (1.26) becomes

$$P_{n+1}(y;x) = \int d_z\, P_n(y;z)\, P_1(z;x)$$

$$= \int d_z\, P_n(y;z)\, P(z;x) \tag{1.27}$$

Suppose that as $n \to \infty$, $P_n(y;x)$ tends to a limit $P(x)$ indepdendent of the initial value.

Then the limiting distribution $P(x)$ satisfies the integral equation

$$P(x) = \int d_z P(z)\, P(z;x) = \int P(z;x)\, dP(z) \tag{1.28}$$

In the above relations, distribution functions can be replaced by p.d.f.'s when these exist.

1.7 Unit Summary

The stochastic process is introduced in this unit. The Markov Chain is defined and several properties are discussed here. Different types of states and Chains are defined with examples.

1.8 Self Assessment Questions

1.1. Define stochastic process with example. Classify it with respect to state space and time.

1.2. Define Markov Chain with example. What do you mean by state and transition probability? What do you mean transition matrix?

1.3. State Gambler's ruin problem and write transition matrix for it.

1.4. Write transition matrix for the problem of random walk between reflecting barriers.

1.5. Define order of a Markov Chain. Discuss how a Markov Chain can be represented as a graph.

1.6. State and prove Chapman-Kolmogorov equation.

1.7. Suppose that probability of a dry day following a rainy day is $\frac{2}{3}$ and that the probability of a rainy day following a dry day is $\frac{1}{2}$ and t.p.m.

$P = \begin{pmatrix} 1/3 & 2/3 \\ 1/2 & 1/2 \end{pmatrix}$. If June 2 is a dry day then find the probability that June 4 and June 6 are dry day.

1.8. Define the following :

(i) accessible state, (*ii*) return state,

(ii) periodic state (*iv*) aperiodic state (*v*) closed state,

(vi) irreducile chain, (vii) perisistent state

(viii) transient state and (ix) ergodic.

1.9. State and prove first entrance theorem.

1.10. Prove that the state j is persistent iff

$$\sum_{n=0}^{\infty} p_{jj}^{(n)} = \infty.$$

1.11. Suggested Further Reading

1. J. Medhi, Stochastic Processes (2e), New Age International Publishers.

2. P. Mukhopadhyay, Mathematical Statistics, New Central Book Agency.

3. G. Klimov, Probability Theory and Mathematical Statistics, Mir Publishers Moscow.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III Group-A

Module No. - 26

PROBABILITY AND STATISTICS

(Markov Process)

CONTENT :

In this unit, Markov process with discrete and continuous state space are introduced. Poisson process has many applications in real life situations where uncertainty occurs. Many problems can deal with birth and death process. Wiener process and branching process has also many applications.

Objectives:

- Poisson process
- Pure birth process and its p.g.f.
- Birth and death process, p.g.f., mean probability size, extinction probability
- Wiener process
- Branching process

2.1 Discrete State Space : Poisson Process

Poisson process is a stochastic process in *continuous time and with discrete state space*. Let N(t) be the number of occurrences of the event E in an interval (o, t). Let $p_n(t)$ be the probability that the random variable $N(t)$ assumes the value n i.e.,

$$p_n(t) = P(N(t) = n) \tag{2.1}$$

This probability is a function of time t. Since the only possible values of n are $n = 0, 1, 2, ...$

$$\sum_{n=0}^{\infty} p_n(t) = 1 \tag{2.2}$$

Thus $\{p_n(t)\}$ represents the probability distribution of the random variable $N(t)$ for every value of t. The family of random variables $\{N(t), t \geq 0\}$ is a stochastic process. Here the time t is continuous, the state space of $N(t)$ is discrete and integral-valued and the process is integral-valued.

Under contain conditions, the number of telephone calls, arrival of customers for service at a counter, number of accident at a certain place, etc., follows Poisson process.

Postulates for Poisson Process

1. **Independence :** $N(t)$ is independent of the number of occurrences of the event E is an interval prior to the interval $(0,t)$.

2. **Homogeneity in time :** $p_n(t)$ depend only on the length t of the interval and is independent of where this interval is situated.

3. **Regularity :** In an interval of infinitesimal length has h, the probability of exactly one occurrence is $\lambda h + o(h)$ and that of more than one occurrence is of $o(h)$, where $o(h)$ means, $\frac{o(h)}{\lambda} \to 0$ as $h \to 0$.

In other words, if the interval $(t.t+h)$ is of short duration h, then

(i) probability of one occurrence $p_1(h) = \lambda h + o(h)$, (2.3)

(ii) probability of k $(k = 2, 3, ...)$ occurrences is of $o(h)$, i.e. $\sum_{k=2}^{\infty} p_k(h) = o(h)$, (2.4)

(iii) Since

$$\sum_{k=0}^{\infty} p_k(h) = 1.$$

Therefore, $p_o(h) = 1 - \sum_{k=1}^{\infty} p_k(h) = 1 - \lambda h + o(h)$. (2.5)

Theorem 2.1.1 Under above postulates, $N(t)$ follows Poisson distribution with mean λt, i.e. $p_n(t)$ is given by the Poisson law.

$$p_n(t) = \frac{\bar{e}^{\lambda}(\lambda t)^n}{n!}, n = 0,1,2,... \quad (2.6)$$

Proof. Let $p_n(t+h)$ be the probability of occurrence of $n(\geq 1)$ events in the time interval $(o, t+h)$.

Fig. 2.1

Therefore,

$p_n(t+h)$ = P (n events occur in (o,t) and no events occur in $(t,t+h)$

+ $P\big((n-1)$ events occur in (o,t) and 1 event occurs in $(t,t+h)\big)$

+ $P\big((n-2)$ events occur in (o,t) and 2 events occur in $(t,t+h)\big)$

+ .....

+ P (no events occur in (o,t) and n events occur in $(t,t+h)\big)$.

$$= \{p_n(t)(1-\lambda h) + o(h)\} + \{p_{n-1}(t)(\lambda h) + o(h)\} + \{o(h) + ... + o(h)\}$$

$$= p_n(t)(1-\lambda h) + p_{n-1}(t)(\lambda h) + o(h), n \geq 1$$

$$\text{or, } \frac{p_n(t+h) - p_n(t)}{h} = -p_n(t) + \lambda p_{n-1}(t) + \frac{o(h)}{h}$$

As $h \to 0$,

$$p_n'(t) = -\lambda p_n(t) + \lambda p_{n-1}(t), n \geq 1. \tag{2.7}$$

For $n = 0$,

$$p_0(t+h) = P(\text{no events occur in } (o,t) \text{ and no events occur in } (t, t+h))$$

$$= p_o(t)(1-\lambda h) + o(h)$$

$$\text{or, } \frac{p_o(t+h) - p_o(t)}{h} = -\lambda p_o(t) + \frac{o(h)}{h}$$

As $h \to 0$,

$$p_o'(t) = -\lambda p_o(t). \tag{2.8}$$

Initial condition

$p_o(0) = 1$ and $p_n(0) = 0$, for $n \geq 1$.

From (2.8), $\frac{p_o'(t)}{p_o(t)} = \lambda$.

Integrating, we get

$$\log p_o(t) = -\lambda t + \log C_1$$

$$\text{or, } p_o(t) = C_1 e^{-\lambda t}$$

As $p_o(0) = 1$, $C_1 = 1$

$$\therefore p_o(t) = e^{\lambda t}. \tag{2,9}$$

From (2.7), we have for $n=1$.

$$p_1'(t) = -\lambda p_1(t) + \lambda p_o(t)$$

$$\text{or, } p_1'(t) + \lambda p_1(t) = \lambda e^{\lambda t}.$$

This is linear equation of $p_1(t)$ and I.F. is $e^{\lambda t}$.

Multiplying above equation by $e^{\lambda t}$, we get

$$\frac{d}{dt}\left(p_1(t)e^{\lambda t}\right) = \lambda$$

Interesting, we get

$$p_1(t)e^{\lambda t} = \lambda t + C_2$$

or, $p_1(t) = (\lambda t + C_2)\bar{e}^{\lambda t}$.

As $p_1(0) = 0,\ C_2 = 0.$

Hence

$$p_1(t) = \frac{\lambda t\, \bar{e}^{\lambda t}}{1!}$$

Let

$$p_m(t) = \frac{(\lambda t)^m \bar{e}^{\lambda t}}{m!} \qquad (2.10)$$

be the solution of (2.7).

Therefore, for $n = m+1$, (2.7) becomes .

$$p'_{m+1}(t) + \lambda p_{m+1}(t) = \lambda p_m(t)$$

$$= \frac{\lambda(\lambda t)^m \bar{e}^{\lambda t}}{m!}$$

This is also a linear equation of $p_{m+1}(t)$ and its I.F. is $e^{\lambda t}$. Multiplying above equation by $e^{\lambda t}$, we get

$$\frac{d}{dt}\left(p_{m+1}(t)e^{\lambda t}\right) = \frac{\lambda(\lambda t)^m}{m!}.$$

Integrating, we get

$$p_{m+1}(t)e^{\lambda t} = \frac{(\lambda t)^{m+1}}{(m+1)m!} + C_3$$

or $p_{m+1}(t) = \left(\frac{(\lambda t)^{m+1}}{(m+1)!} + C_3 \right) \bar{e}^{\lambda t}$.

As $p_{m+1}(0) = 0, \; C_3 = 0.$

$$\therefore p_{m+1}(t) = \frac{(\lambda t)^{m+1}}{(m+1)!} \bar{e}^{\lambda t}.$$

Hence

$$p_n(t) = \frac{(\lambda t)^n \bar{e}^{\lambda t}}{n!}, \; n = 0,1,2,...$$

Note: The mean and variance of Poisson process with parameter λt are λt. The mean number of occurrences per unit time (t=1), i.e., in an interval of unit length is λ. The mean rate λ per unit time is known as the parameter of the Poisson process.

The mean and variance of $N(t)$ are functions of t; in fact, its distribution is dependent on t.

Postulate 1 implies that Poisson process is Markovian; postulate 2 that Poisson process is time-homogeneous; postulate 3 that in an infinitesimal interval of length h, the probabioity of exactly one occurrence is approximately proportional to the length h of that interval and that of the simultaneous occurrence of two or more events is extremely small.

Poisson process has independent as well as stationary increments. Again for every t, future increments of a Poisson process are independent of the process generated.

2.1.1 Properties of Poisson Process

Property 2.1.1 (Additive Property): Sum of two independent Poisson processes is a Poisson process.

Proof. Let $N_1(t)$ and $N_2(t)$ be two Poisson processes with parameters λ_1, λ_2 respectively and let

$$N(t) = N_1(t) + N_2(t).$$

The probability generating function (*p.g.f.*) of $N_i(t)$, $(i = 1,2)$ is

$$E\left(s^{Ni(t)}\right) = e^{\lambda i(s-1)t}.$$

The *p.g.f.* of $N(t)$ is

$$E\left(s^{N(t)}\right) = E\left(s^{N_1(t)+N_2(t)}\right)$$

$$= E\left(s^{N_1(t)}\right)E\left(s^{N_2(t)}\right).$$

[Since $N_1(t)$ and $N_2(t)$ are independent].

Now, $E\left(s^{N(t)}\right) = e^{\lambda_1(s-1)t}.e^{\lambda_2(s-1)t}$

$$= e^{(\lambda_1+\lambda_2)(s-1)t}.$$

Thus $N(t)$ is a Poisson process with parameter $\lambda_1+\lambda_2$.

Property 2.1.2 If $\{N(t)\}$ is a Poisson process then the correlation (auto-correlation) coefficient between $N(t)$ and $N(t+s)$ is $\left\{\frac{t}{t+s}\right\}^{1/2}$.

Proof. Let λ be the parameter of the process, then

mean= $E(N(T)) = \lambda T$, $Var(N(T)) = \lambda T$,

$E(N^2(T)) = \lambda T + (\lambda T)^2$ for $T = t$ and $t+s$.

Since $N(t)$ and $\{N(t+s)-N(t)\}$ are independent, $\{N(t), t \geq 0\}$ being a Poisson process.

$$E\{N(t)N(t+s)\} = E[N(t)\{N(t+s)-N(t)+N(t)\}]$$

$$= E\{N^2(t)\} + E\{N(t)\}E\{N(t+s)-N(t)\}$$

Hence

$$E\{N(t)N(t+s)\} = (\lambda t + \lambda^2 t^2) + \lambda t \cdot \lambda s.$$

Thus the autocovariance between $N(t)$ and $N(t+s)$ is given by

$$Cov(t,t+s) = E\{N(t)N(t+s)\} - E\{N(t)\}E\{N(t+s)\}$$

$$= (\lambda t + \lambda^2 t^2 + \lambda^2 ts) - \lambda t(\lambda t + \lambda s) = \lambda t.$$

Hence the autocorrelation is

$$\rho(t,t+s)=\frac{Cov(t,t+s)}{\sqrt{Var\,N(t)}\sqrt{Var\,N(t+s)}}$$

$$=\sqrt{\left(\frac{t}{t+s}\right)}.$$

2.2 Pure Birth Process

Let λ be the rate of birth in a population. Here the probability that k events occur in the interval $(t, t+h)$ given that n events occurred in the interval $(0, t)$ is given by

$$p_k(h)=\begin{cases}\lambda h+O(h) & , \quad k=1\\ O(h) & , \quad k\geq 2\\ 1-\lambda h+O(h), & k=0.\end{cases}$$

$p_k(h)$ is independent of n as well as t. We generalize the process by considering that λ is not constant but is a function of n or t or both; the resulting process will still be Markovian in nature.

Now, we consider the λ is a function of n, the population size. We assume that

$$p_k(h)=\begin{cases}\lambda_n h+O(h) & , \quad k=1\\ O(h) & , \quad k\geq 2\\ 1-\lambda_n h+O(h), & k=0.\end{cases}$$

Now, $p_n(t+h)$ = probability of having n person in the system in an interval $(0, t+h)$.

= probability of n persons in $(0, t)$ × probability of no birth in $(t, t+h)$.

+ probability of $(n-1)$ persons in $(0, t)$ × probability of one birth in $(t, t+h)$.

+ probability of $(n-i)$ persons in $(0, t)$ × probability of one birth in $(t, t+h)$, $i \geq 2$.

$$=p_n(t)(1-\lambda_n h)+p_{n-1}(t)\lambda_{n-1}h+O(h), n\geq 1.$$

or, $$\frac{p_n(t+h)-p_n(t)}{h}=-\lambda_n p_n(t)+p_{n-1}(t)\lambda_{n-1}+\frac{O(h)}{h}.$$

As $h \to 0$

$$p_n'(t)=-\lambda_n p_n(t)+\lambda_{n-1}p_{n-1}(t),\ n\geq 1.$$

For n = 0,

$p_0(t+h)$ = probability of no persons in $(0, t)$ × probability of no birth in $(t, t+h)$

$$= p_0(t)(1-\lambda_0 h) + O(h)$$

or, $$\frac{p_0(t+h)-p_0(t)}{h} = -\lambda_0 p_0(t) + \frac{O(h)}{h}$$

As $h \to 0, p_0'(t) = -\lambda_0 p_0(t).$ (2.12)

In particular, if $\lambda_n = n \cdot \lambda$ then this pure birth process is called Yule-Furry process.

In this case,

$$p_n'(t) = -n\lambda p_n(t) + (n-1)\lambda p_{n-1}(t), n \geq 1$$

and $p_0'(t) = 0.$ (2.13)

Let the initial conditions be

$p_1(0) = 1, p_i(0) = 0$ for $i \neq 1$, the process started with only one member at time $t = 0$.

For $n = 1$,

$$p_1'(t) = -\lambda p_1(t)$$

or, $$\frac{dp_1(t)}{p_1(t)} = -\lambda dt$$

Integrating,

$$\log p_1(t) = -\lambda t + \log c_1$$

or, $p_1(t) = c_1 \bar{e}^{\lambda t}$.

Putting $p_1(0) = 1$ we have $c_1 = 1$

$$\therefore p_1(t) = \bar{e}^{\lambda t}.$$

For $n = 2$, we have

$$p_2'(t) = -2\lambda p_2(t) + \lambda p_1(t)$$

or, $$p_2'(t) + 2\lambda p_2(t) = \lambda p_1(t) = \lambda \bar{e}^{\lambda t}$$

This is a linear equation of $p_2(t)$ and its I.F. $= e^{2\lambda t}$.

Multiplying above equation by $e^{2\lambda t}$, we get

$$\frac{d}{dt}\left(p_2(t) e^{2\lambda t}\right) = \lambda e^{\lambda t}.$$

Integrating, we obtain

$$p_2(t)e^{2\lambda t} = \int \lambda e^{\lambda t} dt = e^{\lambda t} + c_2.$$

Since $p_2(0) = 0$. Therefore, $c_2 = -1$.

Thus, $p_2(t) = \bar{e}^{2\lambda t}\left(e^{\lambda t} - 1\right) = \bar{e}^{\lambda t}\left(1 - \bar{e}^{\lambda t}\right)$.

Therefore, $p_n(t) = \bar{e}^{\lambda t}\left(1 - \bar{e}^{\lambda t}\right)^{n-1}$ holds for $n = 1, 2$. (2.14)

Let (2.14) be true for $n = m$.

Then

$$p_m(t) = \bar{e}^{\lambda t}\left(1 - \bar{e}^{\lambda t}\right)^{m-1}. \quad (2.15)$$

From (2.13), for $n = m + 1$,

$$p'_{m+1}(t) = -(m+1)\lambda\, p_{m+1}(t) + m\lambda\, p_m(t)$$

or, $p'_{m+1}(t) + (m+1)\lambda\, p_{m+1}(t) = m\lambda\, p_m(t) = m\lambda \bar{e}^{\lambda t}\left(1 - \bar{e}^{\lambda t}\right)^{m-1}$

which is also a linear equation of $p_{m+1}(t)$ and I.F. is $e^{(m+1)\lambda t}$. Multiplying the above equation by $e^{(m+1)\lambda t}$ we get

$$\frac{d}{dt}\left(p_{m+1}(t)e^{(m+1)\lambda t}\right) = m\lambda e^{m\lambda t}\left(1 - \bar{e}^{\lambda t}\right)^{m-1}.$$

Integrating, we have

$$p_{m+1}(t)e^{(m+1)\lambda t} = \int m\lambda e^{\lambda t}\left(e^{\lambda t} - 1\right)^{m-1} dt$$

Putting $e^{\lambda t} - 1 = z$. Then $\lambda e^{\lambda t} dt = dz$.

$$p_{m+1}(t)e^{(m+1)\lambda t} = \int m\, z^{m-1} dz = z^m + c_3.$$

$$= \left(e^{\lambda t} - 1\right)^m + c_3.$$

Since $p_{m+1}(0) = 0$, $c_3 = 0$.

Hence $p_{m+1}(t) = \left(e^{\lambda t} - 1\right)^m \bar{e}^{(m+1)\lambda t}$

$$= \bar{e}^{\lambda t}\left(1-\bar{e}^{\lambda t}\right)^{m}.$$

Therefore, the process, i.e. equation (2.14) holds for $n=m+1$. Hence by mathematical induction

$$p_n(t) = \bar{e}^{\lambda t}\left(1-\bar{e}^{\lambda t}\right)^{n-1}, n \geq 1. \tag{2.16}$$

Solving, $p_0'(t) = 0$, we get $p_0(t) = 0.[As\ p_0(0) = 0.]$

Note : The probability generating function (*pgf*) is given by

$$P(s,t) = \sum_{n=1}^{\infty} p_n(t)\, s^n = \sum_{n=1}^{\infty} \left\{\bar{e}^{\lambda t}\left(1-\bar{e}^{\lambda t}\right)^{n-1}\right\} s^n$$

$$= \bar{e}^{\lambda t} \cdot s \sum_{n=1}^{\infty} \left(1-\bar{e}^{\lambda t}\right)^{n-1} s^{n-1}$$

$$= \bar{e}^{\lambda t} \cdot s \frac{1}{1-s\left(1-\bar{e}^{\lambda t}\right)} \quad [\text{using the relation } 1+x+x^2+.....=(1-x)^{-1}]$$

$$= \frac{s\bar{e}^{\lambda t}}{1-s\left(1-\bar{e}^{\lambda t}\right)}. \tag{2.17}$$

2.3 Birth and Death Process

In this process the number of individuals will increase as well as decrease. The following assumptions are made to describe the birth and death process.

Assumption :

1. Probability of one birth during $(t, t+h), (h \geq 0)$ is $\lambda_n h + 0(h)$ where n is the number of items in a population at time t and λ_n is the birth rate.
2. Probability of more than one birth during $(t, t+h)$ is $0(h)$. Hence the probability of no birth during $(t, t+h)$ is $1-\lambda_n h + 0(h)$.
3. Probability of one death during $(t, t+h)$ is $\mu_n h + 0(h), \mu_n$ is the death rate.
4. Probability of more than one death during $(t, t+h)$ is $0(h)$. Hence the probability of no death during $(t, t+h)$ is $1-\mu_n h + 0(h)$.

Let $p_n(t)$ be the probability of having population size n at time t.

$p_n(t+h)$ = probability of having population size n at time $t+h$.

= (probability of n items in $(0, t)$ × probability of no birth and no death during $(t, t+h)$)

+ (probability of $(n+1)$ items in $(0, t)$ × probability of no birth and probability of one death during $(t, t+h)$.

+ (probability of $(n-1)$ items in $(0, t)$ × probability of one birth and probability of no death during $(t, t+h)$)

+ (probability of n items in $(0, t)$ × probability of one birth and one death during $(t, t+h)$)

+ 0 (h)

$$= p_n(t)\{1-\lambda_n h+0(h)\}\{1-\mu_n h+0(h)\}$$

$$+p_{n+1}(t)\{1-\lambda_{n+1}h+0(h)\}\{\mu_{n+1}h+0(h)\}$$

$$+p_{n-1}(t)\{\lambda_{n-1}h+0(h)\}\{1-\mu_{n-1}h+0(h)\}$$

$$+p_n(t)\{\lambda_n h+0(h)\}\{\mu_n h+0(h)\}+0(h)$$

$$= p_n(t)\{1-\mu_n h-\lambda_n h\}+p_{n+1}(t)\{\mu_{n+1}h\}+p_{n-1}(t)\{\lambda_{n-1}h\}+0(h)$$

[neglecting second and higher powers of h]

or, $p_n(t+h)-p_n(t) = -h(\lambda_n+\mu_n)p_n(t)+\mu_{n+1}h\,p_{n+1}(t)+\lambda_{n-1}h\,p_{n-1}(t)+o(h).$

Dividing both sides by h and taking $h \to 0$, we get

$$p_n'(t) = -(\lambda_n+\mu_n)p_n(t)+\mu_{n+1}p_{n+1}(t)+\lambda_{n-1}p_{n-1}(t), n \geq 1. \qquad (2.18)$$

For n=0, we have

$$p_0(t+h) = p_0(t)\{1-\lambda_0 h+0(h)\}\{1-\mu_0 h+0(h)\}$$

$$+p_1(t)\{1-\lambda_1 h+0(h)\}\{\mu_1 h+0(h)\}$$

$$= p_0(t)-\lambda_0 h\,p_0(t)+\mu_1 h\,p_1(t)-\mu_0 h\,p_0(t)+0(h)$$

or, $p_0(t+h)-p_0(t) = -\lambda_0 h\,p_0(t)-\mu_0 h p_0(t)+\mu_1 h p_1(t)+0(h)$

or, $$\frac{p_0(t+h)-p_0(t)}{h} = -\lambda_0 p_0(t)-\mu_0 p_0(t)+\mu_1 p_1(t)+\frac{0(h)}{h}.$$

Taking limit $h \to 0$, we get

$$p_0'(t) = -(\lambda_0+\mu_0)p_0(t)+\mu_1 p_1(t). \qquad (2.19)$$

Initial condition :

Suppose, intially there are i members in the system

$$\text{i.e. } p_n(0) = \begin{cases} 0, & n \neq i \\ 1, & n = i. \end{cases} \tag{2.20}$$

The equations (2.18) and (2.19) are the equations of the birth and death process with initial conditions given by (2.20).

Condition of existance of the solution of (2.18) and (2.19)

For arbitrary $\lambda_n \geq 0, \mu_n \geq 0$, there always exists a solution $p_n(t)\ (\geq 0)$ such that $\sum p_n(t) \leq 1$. If λ_n and μ_n are bounded, the solution is unique and $\sum p_n(t) = 1$.

Note. When $\lambda_n = \lambda$ i.e. λ_n is independent of the population sign n, then the increase may be thought of as due to an external source such as immigration.

When $\lambda_n = n\lambda$, we have the case of linear birth, the rate of birth in unit interval being λ per indiviudal.

When $\mu_n = \mu$, decrease may be thought of as due to a factor such as emigration.

When $\mu_n = n\mu$, we have the case of linear death, the rate of death in unit interval being μ per individual.

2.3.1 Solution of linear growth process

(a) **Generating function**

When $\lambda_n = n\lambda$ and $\mu_n = n\mu (n \geq 1)$ then the process is called linear growth process, where $\lambda_0 = 0, \mu_0 = 0$. If $X(t)$ denotes the total number of members at time t, then from (2.18) and (2.19), we have the following differential - difference equations for $p_n(t) = P(X(t) = n)$,

$$p_n'(t) = -n(\lambda + \mu)p_n(t) + \lambda(n-1)p_{n-1}(t) + \mu(n+1)(t), n \geq 1 \tag{2.21}$$

and $$p_0'(t) = \mu p_1(t). \tag{2.22}$$

If the initial population size is i, i.e. $X(0) = i$, then we have the initial condition $p_i(0)=1$ and $p_n(0) = 0, n \neq i$.

Let $P(s,t) = \sum_{n=1}^{\infty} p_n(t)s^n$ be the probability generating function of $\{p_n(t)\}$.

Then $\frac{\partial P}{\partial s} = \sum_{n=1}^{\infty} n\, p_n(t)\, s^{n-1}$ and $\frac{\partial P}{\partial t} = \sum_{n=0}^{\infty} p_0'(t)\, s^n$.

Multiplying (2.21) by s^n and adding for $n = 1, 2, 3, \ldots$ and adding (2.22) with it, we get

$$\frac{\partial P}{\partial t} = -(\lambda+\mu)\sum_{n=1}^{\infty} n\, p_n(t)\, s^n + \lambda\sum_{n=1}^{\infty}(n-1)\, p_{n-1}(t)\, s^n$$

$$+\mu\left\{\sum_{n=1}^{\infty}(n+1)\, p_{n+1}(t)\, s^n + p_1(t)\right\}$$

$$= -(\lambda+\mu)s\sum_{n=1}^{\infty} n\, p_n(t)\, s^{n-1} + \lambda s^2\sum_{n=1}^{\infty} n\, p_n(t) s^{n-1}$$

$$+\mu\sum_{n=1}^{\infty} n\, p_n(t)\, s^{n-1}$$

$$= -(\lambda+\mu)s\frac{\partial P}{\partial s} + \lambda s^2\frac{\partial P}{\partial s} + \mu\frac{\partial P}{\partial s}$$

$$= \left\{\mu - (\lambda+\mu)s + \lambda s^2\right\}\frac{\partial P}{\partial s}.$$

This is a partial differential equation of Lagrangian type. The initial condition is $X(0) = i$.

The Lagrange's equation becomes

$$\frac{dt}{1} = \frac{ds}{-(s-1)(\lambda s-\mu)} = \frac{dp}{0} \quad [\text{Of the form } pP+qQ=R]$$

or, $$dt = \frac{ds}{-(s-1)(\lambda s-\mu)} = \frac{1}{\lambda-\mu}\left[\frac{1}{1-s} + \frac{\lambda}{\lambda s-\mu}\right]ds.$$

Integrating, we get

$$(\lambda-\mu)t = \log\left[(\lambda s-\mu)/(1-s)\right] + \log c$$

or, $$\frac{1-s}{\lambda s-\mu}e^{(\lambda-\mu)t} = c. \qquad (2.23)$$

Again, $\frac{dt}{1} = \frac{dP}{0}$ gives $P = \text{Constant} = g(s)$.

or, $P(s,t) = g(s)$. (2.24)

That is, $\sum_{n=0}^{\infty} p_n(t)s^n = g(s)$

or, $s^i = g(s)$ $[\because p_n(0) = 0 \text{ and } n \neq i, p_i(0) = 1.]$

Hence (2.24) becomes

$$P(s,t) = s^i. \tag{2.25}$$

When $t = 0$ equation (2.23) becomes

$$\frac{1-s}{\lambda s - \mu} = c \quad \text{or,} \quad s = \frac{1+\mu c}{1+\lambda c}.$$

Therefore, (2.25) becomes,

$$P(s,t) = \left(\frac{1+\mu c}{1+\lambda c}\right)^i. \tag{2.26}$$

where $c = \frac{1-s}{\lambda s - \mu} e^{(\lambda-\mu)t}$ (2.27)

Therefore,

$$P(s,t) = \left[\frac{\mu(1-s) - (\mu - \lambda s)\bar{e}^{(\lambda-\mu)t}}{\lambda(1-s) - (\mu - \lambda s)\bar{e}^{(\lambda-\mu)t}}\right]^i$$

$$= \left[\frac{\mu\left\{1 - \bar{e}^{(\lambda-\mu)t}\right\} - \left\{\mu - \lambda\bar{e}^{(\lambda-\mu)t}\right\}s}{\left\{\lambda - \mu\bar{e}^{(\lambda-\mu)t}\right\} - \lambda\left\{1 - \bar{e}^{(\lambda-\mu)t}\right\}s}\right]^i. \tag{2.28}$$

Explicit expression for $p_n(t)$ can be obtained from the above by expanding $P(s,t)$ as a power series in s for i=1 as

$$p_n(t) = \{1 - \alpha(t)\}\{1 - \beta(t)\}\{\beta(t)\}^{n-1}, n = 1,2,.....$$

$$p_0(t) = \alpha(t),$$

where $\alpha(t) = \frac{\mu\left\{e^{(\lambda-\mu)t} - 1\right\}}{\lambda e^{(\lambda-\mu)t} - \mu}$ and

$$\beta(t) = \frac{\lambda\left\{e^{(\lambda-\mu)t} - 1\right\}}{\lambda e^{(\lambda-\mu)t} - \mu}.$$

(b) Mean population size

The mean population size can be obtained from $P(s,t)$ by differentiating it w.r.t. s and substituting $s=1$, i.e.,

$$E\{X(t)\} = \frac{\partial P}{\partial s}\Big)_{s=1} = \sum_{n=1}^{\infty} n\, p_n(t) = \alpha_1(t)\,(say).$$

Similarly,

$$E\{X^2(t)\} = \alpha_2(t) = \sum_{n=1}^{\infty} n^2 p_n(t).$$

Multiplying (2.21) by n and adding for $n = 1, 2, 3, \ldots$ we have

$$\sum_{n=1}^{\infty} n\, p_n'(t) = -(\lambda+\mu)\sum_{n=1}^{\infty} n^2 p_n(t) + \lambda\sum_{n=1}^{\infty} n(n-1)p_{n-1}(t) + \mu\sum_{n=1}^{\infty} n(n+1)p_{n+1}(t). \qquad (2.29)$$

Now,

$$\sum_{n=1}^{\infty} n(n-1)p_{n-1}(t) = \sum_{n=1}^{\infty}\left[(n-1)^2 + (n-1)\right]p_{n-1}(t)$$

$$= \sum_{n=1}^{\infty}(n-1)^2 p_{n-1}(t) + \sum_{n=1}^{\infty}(n-1)p_{n-1}(t)$$

$$= \alpha_2(t) + \alpha_1(t).$$

Again, $\sum_{n=1}^{\infty} n(n+1)p_{n+1}(t) = \sum_{n=1}^{\infty}\left[(n+1)^2 - (n+1)\right]p_{n+1}(t)$

$$= \sum_{n=1}^{\infty}(n+1)^2 p_{n+1}(t) - \sum_{n=1}^{\infty}(n+1)p_{n+1}(t)$$

$$= \{\alpha_2(t) - p_1(t)\} - \{\alpha_1(t) - p_1(t)\} = \alpha_2(t) - \alpha_1(t)$$

and $\sum_{n=1}^{\infty} n\, p_n'(t) = \alpha_1'(t).$

Hence (2.29) becomes

$$\alpha_1'(t) = -(\lambda+\mu)\alpha_2(t) + \lambda\{\alpha_2(t)+\alpha_1(t)\} + \mu\{\alpha_2(t)-\alpha_1(t)\}$$

$$= (\lambda-\mu)\alpha_1(t).$$

Integrating, we get

$$\alpha_1(t) = ce^{(\lambda-\mu)t}.$$

The initial condition gives

$$\alpha_1(0) = \sum_{n=1}^{\infty} n\, p_n(0) = i.$$

$\therefore c = \alpha_1(0) = i.$

Hence

$$\alpha_1(t) = ie^{(\lambda-\mu)t}. \qquad (2.30)$$

Hence the mean population size when initial population size i is $ie^{(\lambda-\mu)t}$.

Limiting case :

As $t \to \infty$, the mean population size $\alpha_1(t) \to 0$ for $\lambda < \mu$ or to ∞ for $\lambda > \mu$ and to the constant value i when $\lambda=\mu$.

(c) Extinction probability

Since $\lambda_0 = 0$, 0 is an absorbing state i.e., once the population size reaches 0, it remains at 0 thereafter. This is the interesting case of extinction of the population.

Suppose the process starts with only one member at time 0, i.e., $X(0)=1$.

Then from (2.28),

$$P(s,t) = \frac{a-bs}{c-ds} = \frac{a}{c}\frac{1-bs/a}{1-ds/c},$$

where

$$a = \mu\left\{1-e^{(\lambda-\mu)t}\right\}$$

$$b = \mu - \lambda e^{(\lambda-\mu)t}$$

$$c = \lambda - \mu e^{(\lambda-\mu)t}$$

$$d = \lambda\left\{1 - e^{(\lambda-\mu)t}\right\}.$$

$$\therefore P(s,t) = \frac{a}{c}\left(1-\frac{bs}{a}\right)\left(1-\frac{ds}{c}\right)^{-1}$$

$$= \frac{a}{c}\left(1-\frac{bs}{a}\right)\left\{1+\frac{ds}{c}+\left(\frac{ds}{c}\right)^2+\left(\frac{ds}{c}\right)^3+....\left(\frac{ds}{c}\right)^{n-1}+\left(\frac{ds}{c}\right)^n+.....\right\}.$$

Coefficient of s^n in $P(s,t)$ is

$$\frac{a}{c}\left\{\left(\frac{d}{c}\right)^n - \frac{b}{a}\left(\frac{d}{c}\right)^{n-1}\right\} = p_n(t) \tag{2.31}$$

and $\frac{a}{c} = p_0(t)$.

Therefore,

$$\lim_{t\to\infty} p_0(t) = \lim_{t\to\infty} \frac{\mu\left\{1-e^{(\lambda-\mu)t}\right\}}{\lambda-\mu e^{(\lambda-\mu)t}}$$

$$= \begin{cases} \mu/\lambda, & \lambda > \mu \\ 1, & \lambda \le \mu \end{cases} \tag{2.32}$$

and $\lim_{t\to\infty} p_n(t) = 0$ for $n \neq 0$. (2.33)

In other words, the probability of ultimate extinction is 1 when $\mu > \lambda$ and is $\frac{\mu}{\lambda}(<1)$ when $\mu < \lambda$.

If initially $X(0) = i$ then the probability of ultimate extinction is $\left(\frac{\mu}{\lambda}\right)^i$ for $\mu < \lambda$.

2.4 Markov Processes with Continuous State Space : Wiener Process

Consider that a (Brownian) particle performs a random walk such that in a small interval of time of duration Δt, the displacement of the particle to the right or to the left is also of small magnitude Δx, the total displacement $X(t)$ of the particle in time t being x. Suppose that the random variable Z_i denotes the length of the ith step taken by the particle in a small interval of time Δt and that

$$P(Z_i = \Delta x) = p \text{ and } P(Z_i = -\Delta x) = q,\ p+q = 1,\ 0<p<1, \text{ where } p \text{ is independent of } x \text{ and } t.$$

Suppose that the interval of length t is divided into n equal subintervals of length Δt and that the displacements $Z_i, i = 1,2,....,n$ in the n steps are mutually independent random variables. Then $n \times (\Delta t) = t$ and the total displacement $X(t)$ is the sum of n i.i.d (independently identically distributed) random variables Z_i, i.e.

$$X(t) = \sum_{i=1}^{n(t)} Z_i, n \equiv n(t) = t / \Delta t.$$

We have $E(Z_i) = Z_i P(Z_i = \Delta x) + Z_i P(Z_i = -\Delta x)$

$$= \Delta x \cdot p - \Delta x \cdot q = (p-q) \cdot \Delta x.$$

$$E(Z_i^2) = (\Delta x)^2 p + (\Delta x)^2 q = (\Delta x)^2.$$

$$\therefore Var(Z_i) = E(Z_i^2) - \{E(Z_i)\}^2 = (\Delta x)^2 - (p-q)^2 (\Delta x)^2$$

$$= 4pq(\Delta x)^2.$$

Hence

$$E\{X(t)\} = nE(Z_i) = t(p-q)\Delta x / \Delta t \qquad (2.34)$$

$$[\because n = t / \Delta t]$$

and $Var\{X(t)\} = n\ Var(Z_i) = 4pqt(\Delta x)^2 / \Delta t.$

To get a meaningful result, as $\Delta x \to 0, \Delta t \to 0$,

We must have

$$\frac{(\Delta x)^2}{\Delta t} \to \text{a limit},$$

$$(p-q) \to \text{a multiple of } \Delta x. \qquad (2.35)$$

We assume that, in an interval of length t, $X(t)$ has mean–value function equal to μt and variance function equal to $t\sigma^2$, that is, we suppose that as $\Delta x \to 0, \Delta t \to 0$ in such a way that (2.35) are satisfied and

$$E\{X(t)\} = \mu t \text{ and } Var\{X(t)\} = t\sigma^2. \tag{2.36}$$

Therefore from (2.34), we have

$$\frac{(p-q)\Delta x}{\Delta t} \to \mu, \frac{4pq(\Delta x)^2}{\Delta t} \to \sigma^2. \tag{2.37}$$

The relation (2.35) and (2.37) will be satisfied when $\Delta x = \sigma\sqrt{\Delta t}$ (2.38)

and $p = \frac{1}{2}\{1 + \mu\sqrt{\Delta t}/\sigma\}, q = \frac{1}{2}\{1 - \mu\sqrt{\Delta t}/\sigma\}.$ (2.39)

Now, since Z_i are i.i.d random variables, the sum $\sum_{i=1}^{n(t)} Z_i = X(t)$ for large $n(t)(\equiv n)$ is assymptotically normal with mean μt and variance $t\sigma^2$ (by central limit theorem for equal components).

Here t represents the length of the interval of time during which the displacement, that takes place is equal to the increment $X(t) - X(0)$. We thus find that for $0 < s < t, \{X(t) - X(s)\}$ is normally distributed with mean $\mu(t-s)$ and variance $\sigma^2(t-s)$. Further, the increments $\{X(s) - X(0)\}$ and $\{X(t) - X(s)\}$ are mutually independent this implies that $\{X(t)\}$ is a Markov process.

We now define a Wiener process or a Brownian motion process as follows:

The stochastic process $\{X(t), t \geq 0\}$ is called a Wiener process (or a Wiener Einstein process or a Brownian motion process) with drift μ and variance parameter σ^2 if

(i) $X(t)$ has independent increments, i.e., for every pair of disjoint intervals of time (s, t) and (u, v), where $s \leq t \leq u \leq v$, the random variables $\{X(t) - X(s)\}$ and $\{X(v) - X(u)\}$ are independent.

(ii) Every increment $\{X(t) - X(s)\}$ is normally distributed with mean $\mu(t-s)$ and variance $\sigma^2(t-s)$.

Since $\{X(t) - X(0)\}$ is normally distributed with mean μt and variance $t\sigma^2$, the transition p.d.f p of a Wiener process is given by

$$p(x_0, x, t)dx = P\{x \leq X(t) < x + dx / X(0) = x_0\}$$

$$= \frac{1}{\sigma\sqrt{2\pi t}} e^{-(x - x_0 - \mu t)^2/(2\sigma^2 t)} dx, -\infty < x < \infty \tag{2.40}$$

A Wiener process $\{X(t), t \geq 0\}$ with $X(0) = 0, \mu = 0, \sigma = 1$ is called a standard Wiener process.

2.4.1 Differential equations for a Wiener process

Let $\{X(t), t \geq 0\}$ be a Wiener process. We can consider the displacement in such a process as being caused by the motion of a particle undergone displacements of small magnitude in a small interval of time. Suppose that $(t - \Delta t, t)$ is an infinitesinmal interval of length Δt and the particle makes in this interval a shift equal to Δx with probability p or a shift equal to $-\Delta x$ with probability $q = 1 - p$. Suppose that p and q are independent of x and t. Let the transition probability that the particle has a displacement from x to $x+\Delta x$ in the interval $(0, t)$, given that it started from x_0 at time 0, be $p(x_0, x; t)\Delta x$.

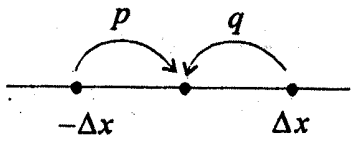
Therefore by Taylor's series

$$p(x_0, x \pm \Delta x; t - \Delta t) = p(x_0, x; t) - \Delta t \frac{\partial p}{\partial t} \pm \Delta x \frac{\partial p}{\partial x} + \frac{1}{2}(\pm \Delta x)^2 \frac{\partial^2 p}{\partial x^2} + 0(\Delta t). \tag{2.41}$$

For simple probability arguments we have

$$p(x_0, x; t)\Delta x = p.\, p(x_0, x - \Delta x; t - \Delta t)\Delta x$$
$$+ q \cdot p(x_0, x + \Delta x; t - \Delta t)\Delta x$$

or, $$p(x_0, x; t) = p.\, p(x_0, x - \Delta x; t - \Delta t)$$
$$+ q \cdot p(x_0, x + \Delta x; t - \Delta t). \tag{2.42}$$



Using (2.41), (2.42) becomes

$$p(x_0, x; t) = p\Big\{p(x_0, x; t) - \Delta t \frac{\partial p}{\partial t} - \Delta x \frac{\partial p}{\partial x}$$
$$+ \frac{1}{2}(\Delta x)^2 \frac{\partial^2 p}{\partial x^2} + 0(\Delta t)\Big\}$$
$$+ q\Big\{p(x_0, x; t) - \Delta t \frac{\partial p}{\partial t} + \Delta x \frac{\partial p}{\partial x} + \frac{1}{2}(-\Delta x)^2 \frac{\partial^2 p}{\partial x^2} + 0(\Delta t)\Big\}$$
$$= p(x_0, x; t) - \Delta t \frac{\partial p}{\partial t} - \Delta x(p - q)\frac{\partial p}{\partial x} + \frac{1}{2}(\Delta x)^2 \frac{\partial^2 p}{\partial x^2} + 0(\Delta t)$$

or, $$\frac{\partial p}{\partial t}+\frac{\Delta x}{\Delta t}(p-q)\frac{\partial p}{\partial x}=\frac{1}{2}\frac{(\Delta x)^2}{\Delta t}\frac{\partial^2 p}{\partial x^2}+\frac{0(\Delta t)}{\Delta t}.$$

Taking limits as $\Delta t \to 0, \Delta x \to 0$ we get grom (2.37), (2.38) and (2.39),

$$p=\frac{1}{2}, q=\frac{1}{2}.$$

Using these limits we get

$$\frac{\partial p}{\partial t}=-\mu\frac{\partial p}{\partial x}+\frac{1}{2}\sigma^2\frac{\partial^2 p}{\partial x^2}, p=p(x_0,x;t). \tag{2.43}$$

The equation is known as the forward diffusion equation of the Wiener process. The backward diffusion equation of the process in the form

$$\frac{\partial p}{\partial t}=\mu\frac{\partial p}{\partial x_0}+\frac{1}{2}\sigma^2\frac{\partial^2 p}{\partial x_0^2}, p=p(x_0,x;t). \tag{2.44}$$

The solution of (2.43) and (2.44) yields $p(x_0,x;t)$ as a normal density of the form given in (2.40).

The equation for a Wiener process with drift $\mu = 0$ is

$$\frac{\partial p}{\partial t}=\frac{1}{2}\sigma^2\frac{\partial^2 p}{\partial x^2} \tag{2.45}$$

which is known as the heat equation.

2.5 Branching Processes

We consider first the discrete time space. Suppose that we start with an initial set of objects or individuals which form the 0th generation - these objects are called ancestors. The offsprings reproduced or the objects generated by the objects of the 0th generation are the direct descendants of the ancestors, and are said to form the first generation; the objects generated by these of the first generation or the direct descendants of the first generation from the second generation and so on, the direct descendants of the rth generation form the $(r+1)$st generation. The number of objects of the rth generation (r=0, 1, 2, ...) is a random variable. We assume that the objects reproduce independently of other objects, i.e. there is no interference.

Let the random variables $X_0, X_1, X_2, \ldots\ldots$ denote the sizes of (or the numbers of objects in) the 0th, 1st, 2nd,..... generations respectively. Let the probability that an object (irrespective of the generation to which it

belongs) generates k similar objects be denoted by p_k, where $p_k \geq 0, k = 1,2,....,$ and $\sum_k p_k = 1.$

The sequence $\{X_n, n = 0,1,2,.....\}$ constitutes a Galton-Watson branching process (or simply a G.W. branching process) with offspring distribution $\{p_k\}$.

The process is also called Bienayame-Galton-Watson process.

2.5.1 Properties of generating functions of branching processes

A Galton-Watson process is a Markov chain $\{X_n, n = 0,1,2,.....\}$ having state space N (set of non-negative integers), such that

$$X_{n+1} = \sum_{r=1}^{X_n} Z_r, \tag{2.46}$$

where Z_r are independently identically distributed (i.i.d) random variable with distribution $\{p_k\}$.

Let

$$P(s) = \sum_k P(Z_r = k)s^k = \sum_k p_k s^k \tag{2.47}$$

be the p.g.f. of $\{Z_r\}$ and let

$$P_n(s) = \sum_k P(X_n = k)s^k, \text{ n = 0, 1, 2, ...} \tag{2.48}$$

be the p.g.f. of $\{X_n\}$.

We assume that $X_0 = 1$; clearly $P_0(s) = s$ and $P_1(s) = P(s)$. The random variables X_1 and Z_r (for any r) both give the same offspring distribution.

Theorem 2.5.1 The generating function $P_n(s)$ satisfy the following relations :

$$P_n(s) = P_{n-1}(p(s)) \tag{2.49}$$

and $$P_n(s) = P(P_{n-1}(s)). \tag{2.50}$$

Proof. We have, for n= 1, 2, ...

$$P(X_n = k) = \sum_{j=0}^{\infty} P(X_n = k / X_{n-1} = j) \cdot P(X_{n-1} = j)$$

$$= \sum_{j=0}^{\infty} P\left(\sum_{r=1}^{j} Z_r = k\right) \cdot P(X_{n-1} = j)$$

$$\therefore P_n(s) = \sum_{k=0}^{\infty} P(X_n = k) s^k$$

$$= \sum_{k=0}^{\infty} s^k \left[\sum_{j=0}^{\infty} P\left(\sum_{r=1}^{j} Z_r = k\right) \cdot P(X_{n-1} = j)\right]$$

$$= \sum_{k=0}^{\infty} P(X_{n-1} = j)\left[\sum_{k=0}^{\infty} P\left(\sum_{r=1}^{j} Z_1 + Z_2 + \ldots\ldots + Z_j = k\right) s^k\right]. \qquad (2.51)$$

The expression within the square bracket, being the p.g.f. of the sum $Z_1 + Z_2 + \ldots\ldots + Z_j$ of j i.i.d random variables each with p.g.f. $P(s)$, equals $[P(s)]^j$. Thus

$$P_n(s) = \sum_{j=0}^{\infty} P(X_{n-1} = j)\,[P(s)]^j$$

or, $P_n(s) = P_{n-1}(P(s))$.

Putting $n = 2, 3, 4, \ldots\ldots$ we get (when $X_0 = 1, P_1 = P$)

$P_2(s) = P_1(P(s)) = P(P(s)), P_3(s) = P_2(P(s)) P_4(s) = P_3(P(s))$, and so on.

Therefore, $P_n(s) = P_{n-1}(P(s)) = P_{n-2}(P(P(s))) = P_{n-2}(P_2(s))$. (2.52)

For $n = 3, P_3(s) = P_1(P_2(s)) = P(P_2(s))$.

Again, $P_n(s) = P_{n-3}(P(P_2(s))) = P_{n-3}(P_3(s))$ and for $n = 4$,

$P_4(s) = P_1(P_3(s)) = P(P_3(s))$.

Thus $P_n(s) = P_{n-k}(P_k(s)), k = 0, 1, 2, \ldots\ldots, n$

and for $k = n-1$,

$P_n(s) = P_1(P_{n-1}(s)) = P(P_{n-1}(s))$.

Note. When $X_0 = i \neq 1$ then (2.50) holds but (2.49) does not hold.

$$p'(1) = E(Z_r) = E(X_1) = m.$$

Theorem 2.5.2. If $m = E(X_1) = \sum_{k=0}^{\infty} k\, p_k$ and

$\sigma^2 = \text{var}(X_1)$ then $E(X_n) = m^n$ and

$$Var(X_n) = \begin{cases} \dfrac{m^{n-1}(m^n - 1)}{m-1}\sigma^2, & \text{if } m \neq 1 \\ n\sigma^2, & \text{if } m = 1. \end{cases}$$

Proof. We have from (2.49),

$$P_n(s) = P_{n-1}(P(s)).$$

$$\therefore P_n'(s) = P_{n-1}'(P(s))\, P'(s). \qquad (2.53)$$

or, $\therefore P_n'(1) = P_{n-1}'(1)\, P'(1)\ [\because P(s) = s]$

$$= m\, P_{n-1}'(1)$$

or, $P_n'(1) = m^2\, P_{n-2}'(1) = m^3\, P_{n-3}'(1) = \ldots\ldots = m^{n-1}\, P'(1) = m^n.$

Thus $E(X_n) = P_n'(1) = m^n.$

Again differentiating (2.53) we get

$$P_n''(s) = P_{n-1}''(P(s))\{P'(s)\}^2 + P_{n-1}'(P(s))\, P''(s).$$

At s=1,

$$P_n''(1) = P_{n-1}''(P(1))\{P'(1)\}^2 + P_{n-1}'(P(1))\, P''(1)$$

$$= P_{n-1}''(1)m^2 + P_{n-1}'(1).A, \text{ where } A = P''(1).$$

$$[\because P'(1) = m \text{ and } P(1) = 1]$$

$$= P_{n-1}''(1)m^2 + m^{n-1}A$$

$$= \left(P''_{n-2}(1)m^2 + m^{n-2}A\right)m^2 + m^{n-1}A$$

$$= P''_{n-2}(1)m^4 + m^n A + m^{n-1}A$$

$$= \left(P''_{n-3}(1)m^2 + m^{n-3}A\right)m^4 + m^n A + m^{n-1}A$$

$$= P''_{n-3}(1)m^6 + m^{n+1}A + m^n A + m^{n-1}A$$

$$= .....$$

$$= P''_0(1)m^{2n} + \left(m^{2n-2} + m^{2n-1} + ... + m^n + m^{n-1}\right)A$$

$$= 0 + Am^{n-1}\left(1 + m + m^2 + ... + m^{n-1}\right) \qquad (2.54)$$

$$[\because P_0(s) = s \text{ or } P''_0(s) = 0]$$

$$= Am^{n-1}\frac{m^n - 1}{m-1} \text{ for } m \neq 1.$$

Now, $P(s) = \sum_k p_k s_k$

$$\therefore P'(s) = \sum_k k\, p_k\, s^{k-1}$$

and $P''(s) = \sum_k k(k-1)\, p_k\, s^{k-2}$.

$$\therefore P''(1) = \sum_k k(k-1)\, p_k = \sum_k k^2 p_k - \sum_k k\, p_k$$

$$= E\left(Z_r^2\right) - E(Z_r).$$

$$\therefore A = P''(1) = E\left(Z_r^2\right) - \left\{E(Z_r)\right\}^2 + \left\{E(Z_r)\right\}^2 - E(Z_r)$$

$$= Var(Z_r) + m^2 - m = \sigma^2 + m(m-1).$$

Similarly,

$$P''_n(1) = E\left(X_n^2\right) - \left\{E(X_n)\right\}^2 + \left\{E(X_n)\right\}^2 - E(X_n)$$

$$= Var(X_n) + \left(m^n\right)^2 - m^n$$

Hence $\text{var}(X_n) + m^n(m^n - 1) = \{\sigma^2 + m(m-1)\}m^{n-1} \cdot \dfrac{m^n - 1}{m-1}$

$$= \sigma^2 m^{n-1} \frac{m^n - 1}{m-1} + m \cdot m^{n-1}(m^n - 1)$$

or, $\text{var}(X_n) = m^{n-1} \dfrac{m^n - 1}{m-1} \sigma^2$ for $m \neq 1$.

when m=1, then from (2.54)

$P_n''(1) = A(1 + 1 + \ldots.. + n \text{ times}) = \text{A.n.}$

In this case, $A = \sigma^2$ and

$$P_n''(1) = \text{var}(X_n) + (1^n)^2 - 1^n = \text{var}(X_n).$$

Therefore $\text{var}(X_n) = n\sigma^2$.

Thus, $Var(X_n) = \begin{cases} \dfrac{m^{n-1}(m^n - 1)}{m-1} \sigma^2, & \text{if } m \neq 1 \\ n\sigma^2, & \text{if } m = 1. \end{cases}$

2.6 Unit Summary

In this unit Poisson process is introduced. Pure birth process and birth and death process are also studied. The probability generating function for birth and death process is determined, where birth and death rates are linear. Mean population size and extinction probability are calculated. An example of continuous time continuous state space Markov process, viz., Wiener process is presented. The differential equation of this process is also established. The Galton-Watson branching process is introduced and studied some of its properties. An exercise is supplied with this unit.

2.7 Self Assessment Questions

2.1 Prove that under certain conditions stated by you the number of telephone calls in a trunk line follows Poisson process.

Find the mean and standard deviation of this process.

2.2 If $N_1(t), N_2(t)$ are two independent Poisson processes with parameters λ_1, λ_2 respectively, then show that

$$P\left(N_1(t) = k \,/\, N_1(t) + N_2(t) = n\right) = n_{c_k} p^k q^{n-k},$$

where $p = \frac{\lambda_1}{\lambda_1 + \lambda_2}, q = \frac{\lambda_2}{\lambda_1 + \lambda_2}$

2.3 The number of accidents in a town follows a Poisson process with a mean of 2 per day and the number X_i of people involved in the ith accident has the distribution

$$P\left(X_i = k\right) = \frac{1}{2^k}, k \geq 1.$$

Find the mean and variance of the number of people involved in accidents per week

2.5 Deduce the probability mass function for pure birth process. Hence deduce the probability generating function for this process.

2.6 State birth and death process. Find the differential-difference equation for birth and death process.

2.7 Find the probability generating function for birth and death process when rate of birth and death are respectively $n\lambda$ and $n\mu$, where n is the population size at any time t. Assume that the initial population size is i.

2.8 Find the probability of ultimate extinction in the case of the linear growth process (birth and death) starting with i individuals at time 0.

2.9 Find the differential equation for Wiener process.

2.10 Show that the generating function $P_n(s)$ for branching process satisfy the following relations:

(i) $P_n(s) = P_{n-1}\left(P(s)\right)$ and

(ii) $P_n(s) = P\left(P_{n-1}(s)\right)$, where $P_1(s) = P(s)$.

2.11 Let $\{X_n, n \geq 0\}$ be a branching process. Show that if $m = E(X_1) = \sum_{k=0}^{\infty} kp_k$ and $\sigma^2 = Var(X_1)$ then $E(X_n) = m^n$ and

$$Var(X_n) = \begin{cases} \dfrac{m^{n-1}(m^n - 1)}{m-1}\sigma^2, & \text{if } m \neq 1 \\ n\sigma^2, & \text{if } m = 1. \end{cases}$$

1.9 Suggested Further Reading

1. J. Medhi, Stochastic Processes (2e), New Age International Publishers.
2. P. Mukhopadhyay, Mathematical Statistics, New Central Book Agency.
3. G. Klimov, Probability Theory and Mathematical Statistics, Mir Publishers. Moscow.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

Paper-III Group-A

Module No. - 27

PROBABILITY AND STATISTICS

(Multiple Correlation and Regression)

CONTENT :

A regression model that involves more than one regressor variable is called a multiple regression model. Fitting of multiple regression equation and analysis of it are discussed in this unit. Multiple regression gives a relationship among the multiple variables and multiple and partial correlation coefficients give the measure of relationship in different situations.

Objectives

- Multiple regression
- Multiple correlation
- Partial correlation
- Regression coefficients
- Linear estimation
- Gauss-Markov linear model
- Exercise

3.1 Multiple Regression

In bivariate regression there is a linear relation between two variables one is taken as dependent variable and one is taken as independent variable. In multiple regression, the linear relation may exists among more than two variables. Here we consider p variables $x_1, x_2, \ldots, x_p$ which are connected by a linear relation. Now our object is to build up a relationship between the 'dependent variable' (called regressand), x_1 and the 'independent variables' (called regressors), $x_2, x_3, \ldots, x_p$, with the idea of using this relationship for predicting the value of the regressand from a knowledge of the values of the regressors. Thus, in estimating the rainfall at a place in a year, it is appropriate to consider the effects of the latitude, the longitude and the altitude of the place on rainfall. Similarly, in estimating the yield of a crop in a year, it is proper to take into account the effects of, say, rainfall average temperature and average humidity, during the period between the sowing and the harvesting of the crop.

Let us assume that the relationship between x_1 and $x_2, x_3, \ldots, x_p$ is, at least in an appropriate sense, given by an equation of the form.

$$X_1 = a + b_2 x_2 + b_3 x_3 \ldots + b_p x_p \qquad (3.1)$$

Our data here will consist of p values, corresponding to the p variables, for each individuals. The values of the variables for the αth individual may be denoted by $(x_{1\alpha}, x_{2\alpha}, \ldots x_{p\alpha}), \alpha = 1,2,\ldots,n$.

We apply the least square method to determine the constants $a, b_2, b_3, \ldots, b_p$.

Let X_1 be the predicted value of x_1 obtained from the equation (3.1). The difference $x_{1\alpha} - X_{1\alpha}$ is the error of estimate corresponding to the αth individual. Thus the sum of square of all errors is $\sum_{\alpha}(x_{1\alpha} - X_{1\alpha})^2$ and let it be

$$E_1 = \sum_{\alpha}\left(x_{1\alpha} - a - b_2 x_{2\alpha} \ldots b_p x_{p\alpha}\right)^2 \tag{3.2}$$

The values of the constants $a, b_2, b_3, \ldots, b_p$ are to be determined such that E_1 is minimum.

The normal equations are

$$\frac{\partial E_1}{\partial a} = 0, \frac{\partial E_1}{\partial b_2} = 0, \frac{\partial E_1}{\partial b_3} = 0, \ldots, \frac{\partial E_1}{\partial b_p} = 0.$$

Differentiating (3.2) partially *w.r.t.* a we get

$$\frac{\partial E_1}{\partial a} = -2\sum_{\alpha}\left(x_{1\alpha} - a - b_2 x_{2\alpha} - \ldots - b_p x_{p\alpha}\right) = 0$$

$$\text{or, } \sum_{\alpha}\left(x_{1\alpha} - a - b_2 x_{2\alpha} - \ldots - b_p x_{p\alpha}\right) = 0$$

$$\text{or, } \sum_{\alpha} x_{1\alpha} = na + b_2 \sum_{\alpha} x_{2\alpha} + \ldots + b_p \sum x_{p\alpha} \tag{3.3}$$

Differentiating (3.2) partially *w.r.t.* b_2, we get

$$\frac{\partial E_1}{\partial b_2} = -2\sum_{\alpha} x_{2\alpha}\left(x_{1\alpha} - a - b_2 x_{2\alpha} - \ldots - b_p x_{p\alpha}\right) = 0$$

$$\text{or, } \sum_{\alpha} x_{2\alpha} x_{1\alpha} = a\sum_{\alpha} x_{2\alpha} + b_2 \sum_{\alpha} x_{2\alpha}^2 + \ldots + b_p \sum x_{2\alpha} x_{p\alpha}. \tag{3.4}$$

Similarly,

$$\sum_{\alpha} x_{3\alpha} x_{1\alpha} = a\sum_{\alpha} x_{3\alpha} + b_2 \sum_{\alpha} x_{3\alpha} x_{2\alpha} + \ldots + b_p \sum_{\alpha} x_{3\alpha} x_{p\alpha}$$

and so on.

Thus the set of normal equations are

$$\left.\begin{aligned}
\sum_{\alpha} x_{1\alpha} &= na + b_2 \sum_{\alpha} x_{2\alpha} + b_3 \sum_{\alpha} x_{3\alpha} + \ldots + b_p \sum_{\alpha} x_{p\alpha} \\
\sum_{\alpha} x_{2\alpha} x_{1\alpha} &= a\sum_{\alpha} x_{2\alpha} + b_2 \sum_{\alpha} x_{2\alpha}^2 + b_3 \sum_{\alpha} x_{2\alpha} x_{3\alpha} + \ldots + b_p \sum_{\alpha} x_{2\alpha} x_{p\alpha} \\
\sum_{\alpha} x_{3\alpha} x_{1\alpha} &= a\sum_{\alpha} x_{3\alpha} + b_2 \sum_{\alpha} x_{3\alpha} x_{2\alpha} + b_3 \sum_{\alpha} x_{3\alpha}^2 + \ldots + b_p \sum_{\alpha} x_{3\alpha} x_{p\alpha} \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
\sum_{\alpha} x_{p\alpha} x_{1\alpha} &= a\sum_{\alpha} x_{p\alpha} + b_2 \sum_{\alpha} x_{p\alpha} x_{2\alpha} + b_3 \sum_{\alpha} x_{p\alpha} x_{3\alpha} + \ldots + b_p \sum_{\alpha} x_{p\alpha}^2
\end{aligned}\right\} \tag{3.5}$$

Dividing (3.3) by n and denoting $\frac{1}{n}\sum x_{1\alpha} = \bar{x}_1, \frac{1}{n}\sum x_{2\alpha} = \bar{x}_2$ and so on, we get

$$\bar{x}_1 = a + b_2\bar{x}_2 + b_3\bar{x}_3 + \ldots + b_p\bar{x}_p, \tag{3.6}$$

which shows incidently that the mean point $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_p)$ necessarily satisfies the prediction equation.

Multiplying (3.6) by $n\bar{x}_2$ and subtracting from the second equation of (3.5) we get

$$\sum_\alpha x_{2\alpha}x_{1\alpha} - \bar{x}_1 \cdot n\bar{x}_2 = b_2\sum_\alpha (x_{2\alpha}^2 - n\bar{x}_2^2) + b_3\sum_\alpha (x_{2\alpha}x_{3\alpha} - n\bar{x}_2\bar{x}_3)$$

$$+ \ldots\ldots + b_p\sum_\alpha (x_{2\alpha}x_{p\alpha} - n\bar{x}_2\bar{x}_p)$$

or, $$S_{21} = b_2 S_{22} + b_3 S_{23} + \ldots\ldots + b_p S_{2p}, \tag{3.7}$$

where $$S_{ij} = \sum_\alpha x_{i\alpha}x_{j\alpha} - n\bar{x}_i\bar{x}_j = \sum_\alpha (x_{i\alpha} - \bar{x}_i)(x_{j\alpha} - \bar{x}_j) \tag{3.8}$$

Similarly, multiplying (3.6) by $n\bar{x}_3, n\bar{x}_4, \ldots, n\bar{x}_p$ and subtracting from the third, fourth, ... pth equation, respectively, of (3.3), we have (p-2) equations determining the b's. Thus

$$\left.\begin{aligned} S_{21} &= b_2S_{22} + b_3S_{23} + \ldots + b_2S_{2p} \\ S_{31} &= b_2S_{32} + b_3S_{33} + \ldots + b_pS_{3p} \\ &- - - - - - - - - - - - - - - - \\ &- - - - - - - - - - - - - - - - \\ S_{p1} &= b_2S_{p2} + b_3S_{p3} + \ldots + b_pS_{pp} \end{aligned}\right\} \tag{3.9}$$

We denote $\frac{1}{n} \times S_{ij} = \frac{1}{n}\sum_\alpha (x_{i\alpha} - \bar{x}_i)(x_{j\alpha} - \bar{x}_j)$ by s_{ij}. (3.10)

Then $$s_{ij} = \begin{cases} Cov(x_i, x_j), \text{ if } i \neq j \\ Var(x_i) \text{ if } i = j \end{cases} \tag{3.11}$$

Dividing all equation of (3.9) by n and using (3.10) we obtain

$$\left.\begin{aligned} s_{21} &= b_2s_{22} + b_3s_{23} + \ldots + b_ps_{2p} \\ s_{31} &= b_2s_{32} + b_3s_{33} + \ldots + b_ps_{3p} \\ &- - - - - - - - - - - - - \\ &- - - - - - - - - - - - - \\ s_{p1} &= b_2s_{p2} + b_3s_{p3} + \ldots + b_ps_{pp} \end{aligned}\right\} \tag{3.12}$$

This system of equation can be written as

$$\begin{pmatrix} s_{21} \\ s_{22} \\ \vdots \\ s_{p1} \end{pmatrix} = \begin{pmatrix} s_{22} & s_{23} & \cdots & s_{2p} \\ s_{32} & s_{33} & \cdots & s_{3p} \\ \cdots & \cdots & \cdots & \cdots \\ s_{p2} & s_{p3} & \cdots & s_{pp} \end{pmatrix} \begin{pmatrix} b_2 \\ b_3 \\ \vdots \\ b_p \end{pmatrix} \tag{3.13}$$

We denote the matrix $\begin{pmatrix} s_{22} & s_{23} & \cdots & s_{2p} \\ s_{32} & s_{33} & \cdots & s_{3p} \\ \cdots & \cdots & \cdots & \cdots \\ s_{p2} & s_{p3} & \cdots & s_{pp} \end{pmatrix}$ by S. This matrix is called the *variance-covariance* or *dispersion* matrix of $x_1, x_2, \ldots, x_p$. If the matrix S is non-singular then the values of $b_2, b_3, \ldots, b_p$ can be determined from the equation (3.13) by Cramer's rule.

Therefore,

$$b_j = \frac{\begin{vmatrix} s_{22} & s_{23} & \cdots & s_{2(j-1)} & s_{21} & s_{2(j+1)} & \cdots & s_{2p} \\ s_{32} & s_{33} & \cdots & s_{3(j-1)} & s_{31} & s_{3(j+1)} & \cdots & s_{3p} \\ & & & & & & & \\ s_{p2} & s_{p3} & \cdots & s_{p(j-1)} & s_{p1} & s_{p(j+1)} & \cdots & s_{pp} \end{vmatrix}}{\begin{vmatrix} s_{22} & s_{23} \cdots\cdots\cdots\cdots\cdots\cdots s_{2p} \\ s_{32} & s_{33} \cdots\cdots\cdots\cdots\cdots\cdots s_{3p} \\ \cdots & \cdots\cdots\cdots\cdots\cdots\cdots \\ s_{p2} & s_{p3} \cdots\cdots\cdots\cdots\cdots\cdots s_{pp} \end{vmatrix}} \tag{3.14}$$

$j = 2, 3, \ldots, p.$

The correlation coefficient r_{ij} between x_i and x_j is $r_{ij} = \frac{s_{ij}}{s_i s_j}$, where s_i, s_j are the standard deviation of x_i and x_j. Then

$$b_j = \frac{\begin{vmatrix} r_{22}s_2s_2 & r_{23}s_2s_3 & \cdots & r_{2(j-1)}s_2s_{j-1} & r_{21}s_2s_1 & r_{2(j+1)}s_2s_{j+1} & \cdots & r_{2p}s_2s_p \\ r_{32}s_3s_2 & r_{33}s_3s_3 & \cdots & r_{3(j-1)}r_3s_{j-1} & r_{31}s_3s_1 & r_{3(j+1)}s_3s_{j+1} & \cdots & r_{3p}s_3s_p \\ \hline r_{p2}s_ps_2 & r_{p3}s_ps_3 & \cdots & r_{p(j-1)}r_ps_{j-1} & r_{p1}s_ps_1 & r_{p(j+1)}s_ps_{j+1} & \cdots & r_{pp}s_ps_p \end{vmatrix}}{\begin{vmatrix} r_{22}s_2s_2 & r_{23}s_2s_3 & \cdots & r_{2j}s_2s_j & \cdots & r_{2p}s_2s_p \\ r_{32}s_3s_2 & r_{33}s_3s_3 & \cdots & r_{3j}s_3s_j & \cdots & r_{3p}s_3s_p \\ \vdots & \vdots & & \vdots & & \vdots \\ r_{p2}s_ps_2 & r_{p3}s_ps_3 & \cdots & r_{pj}s_ps_j & \cdots & r_{pp}s_ps_p \end{vmatrix}}$$

$$= \frac{s_1}{s_2} \frac{\begin{vmatrix} r_{22} & r_{23} & \cdots & r_{2(j-1)} & r_{21} & r_{2(j+1)} & \cdots & r_{2p} \\ r_{32} & r_{33} & \cdots & r_{3(j-1)} & r_{31} & r_{3(j+1)} & \cdots & r_{3p} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ r_{p2} & r_{p3} & \cdots & r_{p(j-1)} & r_{p1} & r_{p(j+1)} & \cdots & r_{pp} \end{vmatrix}}{\begin{vmatrix} r_{22} & r_{23} & \cdots & r_{2j} & \cdots & r_{2p} \\ r_{32} & r_{33} & \cdots & r_{3j} & \cdots & r_{3p} \\ \vdots & \vdots & & \vdots & & \vdots \\ r_{p2} & r_{p3} & \cdots & r_{pj} & \cdots & r_{pp} \end{vmatrix}}$$

$$= (-1)^{j-2} \frac{s_1}{s_j} \frac{\begin{vmatrix} r_{21} & r_{22} & \cdots & r_{2(j-1)} & r_{2(j+1)} & \cdots & r_{2p} \\ r_{31} & r_{32} & \cdots & r_{3(j-1)} & r_{3(j+1)} & \cdots & r_{3p} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{p(j-1)} & r_{p(j+1)} & \cdots & r_{pp} \end{vmatrix}}{\begin{vmatrix} r_{22} & r_{23} & \cdots & r_{2p} \\ r_{32} & r_{33} & \cdots & r_{3p} \\ \vdots & \vdots & & \vdots \\ r_{p2} & r_{p3} & \cdots & r_{pp} \end{vmatrix}} \qquad (3.15)$$

We wreite R for the matrix

$$\begin{pmatrix} r_{11} & r_{12} \cdots\cdots r_{1p} \\ r_{21} & r_{22} \cdots\cdots r_{2p} \\ \cdots & \cdots\cdots\cdots \\ r_{p1} & r_{p2} \cdots\cdots r_{pp} \end{pmatrix}$$

which is the *correlation matrix* of $x_1, x_2, \ldots, x_p$; $|R|$ for the determinant of R and R_{ij} for the cofactor of r_{ij} in R. It may be noted that R is symmetric i.e. $r_{ij} = r_{ji}$ moreover $r_{ii} = 1$.

The numerator of (3.15) is the minor of r_{1j} in R and hence it is $(-1)^{1+j} \times$ cofactor of r_{1j}. Also, the determinant in the denominator is the minor (and also the cofactor) of r_{11} in R.

Hence $b_j = (-1)^{2j-1} \times \frac{s_1}{s_j} \times \frac{R_{1j}}{R_{11}}$.

$$= -\frac{R_{1j}}{R_{11}} \times \frac{s_1}{s_j}, \quad j = 2,3,\ldots,p. \tag{3.16}$$

From (3.6),

$$a = \bar{x}_1 + \sum_{j=2}^{p} \frac{R_{1j}}{R_{11}} \frac{s_1}{s_j} \bar{x}_j. \tag{3.17}$$

Thus the prediction equation called the multiple regression equation of x_1 on $x_2, x_3, \ldots, x_p$ becomes

$$X_1 = \bar{x}_1 - \frac{R_{12}}{R_{11}} \frac{s_1}{s_2}(x_2 - \bar{x}_1) - \frac{R_{13}}{R_{11}} \frac{s_1}{s_3}(x_3 - \bar{x}_3) - \cdots - \frac{R_{1p}}{R_{11}} \frac{s_1}{s_p}(x_p - \bar{x}_p). \tag{3.18}$$

The coefficient $b_j = -\frac{R_{1j}}{R_{11}} \frac{s_1}{s_j}$ is called the partial regression coefficient of x_1 on x_j for fixed $x_2, x_3, \ldots, x_{j-1}, x_{j+1}, \ldots, x_p$ and is often written in the form

$$b_{1j.23\ldots(j-1)(j+1)\ldots p}. \tag{3.19}$$

It gives the amount by which the predicted value X_1 increases when x_j is increased by a unit amount, the other independent variables being kept fixed.

3.1.1 Multiple regression for three variables

The multiple regression equation for the independent variables x_2 and x_3 on x_1 is

$$X_1 = \bar{x}_1 - \frac{R_{12}}{R_{11}}\frac{s_1}{s_2}(x_2 - \bar{x}_2) - \frac{R_{13}}{R_{11}}\frac{s_1}{s_3}(x_3 - \bar{x}_3)$$

$$= \bar{x}_1 + b_{12.3}(x_2 - \bar{x}_2) + b_{13.2}(x_3 - \bar{x}_3), \tag{3.20}$$

or, $X_1 = a + b_{12.3}x_2 + b_{13.2}x_3$, (3.21)

where $b_{12.3} = -\frac{R_{12}}{R_{11}}\frac{s_1}{s_2}$, $b_{13.2} = -\frac{R_{13}}{R_{11}}\frac{s_1}{s_3}$.

Now, $R_{11} = \begin{vmatrix} r_{22} & r_{23} \\ r_{32} & r_{33} \end{vmatrix} = r_{22}r_{33} - r_{23}r_{32}$

$= 1 - r_{23}^2$ as $r_{ii} = 1$ and $r_{ij} = r_{ji}$.

$$R_{12} = -\begin{vmatrix} r_{21} & r_{23} \\ r_{31} & r_{33} \end{vmatrix} = -r_{21}r_{33} + r_{23}r_{31} = r_{23}r_{13} - r_{12}$$

and $R_{13} = \begin{vmatrix} r_{21} & r_{22} \\ r_{31} & r_{32} \end{vmatrix} = r_{21}r_{32} + r_{22}r_{31} = r_{21}r_{32} - r_{13}$

Thus $b_{12.3} = \frac{r_{12} - r_{23}r_{13}}{1 - r_{23}^2}\frac{s_1}{s_2}$ and $b_{13.2} = \frac{r_{13} - r_{12}r_{23}}{1 - r_{23}^2}\frac{s_1}{s_3}$

Example 3.1.1 The following table shows, for each of 18 cinchona plants, the yield of dry bark (in *oz*), the height (in inches) and the girth (in inches) at a height of 6″ from the ground.

Plant No.	Yield of dry bark (*oz*)	Height (in.)	Girth at a height of 6″ (in.)
1	19	8	4
2	51	15	5
3	30	11	3
4	42	21	3
5	25	7	2
6	18	5	1
7	44	10	4
8	56	13	6
9	38	12	3
10	32	13	4

Plant No.	Yield of dry bark (*oz*)	Height (in.)	Girth at a height of 6″ (in.)
11	25	5	2
12	10	6	3
13	20	4	4
14	27	8	4
15	13	7	3
16	49	12	5
17	27	6	3
18	55	16	7

Solution. W denote these variables by x_1, x_2 and x_3 respectively. Here we find the dependence of x_1 on x_2 and x_3, i.e., the multiple regression equation of x_1 on x_2 and x_3.

	x_1	x_2	x_3	x_1^2	x_2^2	x_3^2	x_1x_2	x_1x_3	x_2x_3
	19	8	4	361	64	16	152	76	32
	51	15	5	2601	225	25	765	255	75
	30	11	3	900	121	9	330	90	33
	42	21	3	1764	441	9	882	126	63
	25	7	2	625	49	4	175	50	14
	18	5	1	324	25	1	50	18	5
	44	10	4	1936	100	16	440	176	40
	56	13	6	3136	169	36	728	336	78
	38	12	3	1444	144	9	456	114	36
	32	13	4	1024	169	16	416	128	52
	25	5	2	625	25	4	125	50	10
	10	6	3	100	36	9	60	30	18
	20	4	4	400	16	16	80	80	16
	27	8	4	729	64	16	216	108	32
	13	7	3	169	49	9	91	39	21
	49	12	5	2401	144	25	588	245	60
	27	6	3	729	36	9	162	81	18
	55	16	7	3025	256	49	880	385	112
Total	**581**	**179**	**66**	**22293**	**2133**	**278**	**6636**	**2387**	**715**

Now $\bar{x}_1 = \dfrac{\sum x_{1\alpha}}{n} = \dfrac{518}{18} = 32 \cdot 28 \ oz$

$\bar{x}_2 = \dfrac{\sum x_{2\alpha}}{n} = \dfrac{179}{18} = 9 \cdot 94 \ in.$

$\bar{x}_3 = \dfrac{\sum x_{3\alpha}}{n} = \dfrac{66}{18} = 3 \cdot 67 \ in.$

$s_1 = \dfrac{1}{n}\sqrt{n\sum x_{1\alpha}^2 - \left(\sum x_{1\alpha}\right)^2} = \dfrac{\sqrt{63713}}{18} = 14 \cdot 02 \ \ oz$

$s_2 = \dfrac{1}{n}\sqrt{n\sum x_{2\alpha}^2 - \left(\sum x_{2\alpha}\right)^2} = \dfrac{\sqrt{6353}}{18} = 4 \cdot 43 \ \ in.$

$s_3 = \dfrac{1}{n}\sqrt{n\sum x_{3\alpha}^2 - \left(\sum x_{3\alpha}\right)^2} = \dfrac{\sqrt{648}}{18} = 1 \cdot 41 \ \ in.$

$$r_{12} = \frac{n\sum x_{1\alpha}x_{2\alpha} - \left(\sum x_{1\alpha}\right)\left(\sum x_{2\alpha}\right)}{\sqrt{n\sum x_{1\alpha}^2 - \left(\sum x_{1\alpha}\right)^2}\sqrt{n\sum x_{2\alpha}^2 - \left(\sum x_{2\alpha}\right)^2}} = \frac{15449}{\sqrt{63713}\sqrt{6353}} = 0 \cdot 768$$

$$r_{13} = \frac{n\sum x_{1\alpha}x_{3\alpha} - \left(\sum x_{1\alpha}\right)\left(\sum x_{3\alpha}\right)}{\sqrt{n\sum x_{1\alpha}^2 - \left(\sum x_{1\alpha}\right)^2}\sqrt{n\sum x_{2\alpha}^2 - \left(\sum x_{2\alpha}\right)^2}} = \frac{4620}{\sqrt{63713}\sqrt{648}} = 0 \cdot 719$$

and

$$r_{23} = \frac{n\sum x_{2\alpha}x_{3\alpha} - \left(\sum x_{2\alpha}\right)\left(\sum x_{3\alpha}\right)}{\sqrt{n\sum x_{2\alpha}^2 - \left(\sum x_{2\alpha}\right)^2}\sqrt{n\sum x_{3\alpha}^2 - \left(\sum x_{3\alpha}\right)^2}} = \frac{1056}{\sqrt{6353}\sqrt{648}} = 0 \cdot 520$$

If the multiple regression equation is

$$X_1 = a + b_{12.3}x_2 + b_{13.2}x_3$$

the $b_{12.3} = \dfrac{r_{12} - r_{13}r_{23}}{1 - r_{23}^2} \times \dfrac{s_1}{s_2} = \dfrac{0.394}{0.730} \times \dfrac{14.02}{4.43} = 1.71$

and $b_{13.2} = \dfrac{r_{13} - r_{12}r_{23}}{1 - r_{23}^2} \times \dfrac{s_1}{s_3} = \dfrac{0.320}{0.730} \times \dfrac{14.02}{1.41} = 4.36$

and $a = \bar{x}_1 - b_{12.3}\bar{x}_2 - b_{13.2}\bar{x}_2 = \bar{x}_3 = -0.72.$

Hence the multiple regression equation of x_1 on x_2 and x_3 is

$$X_1 = -0.72 + 1.71x_2 + 4.36x_3$$

3.2 Multiple Correlation

In studying the dependence of x_1 on a set of independent variables, we may want to know to what extent x_1 is influenced by the independent variables. In the case of two variables, x and y, we have seen that r_{xy} serves as a measure of the strength of the interdependence of x and y or, if y may be looked upon as dependent on x, of the extent to which x influences y. Generalising this approach, we may take the simple correlation between x_1 and X_1, i.e. the value of x_1 given by the multiple regression equation of x_1 on $x_2,\ldots,x_p$, as a measure of the joint influence of $x_2, x_3 \ldots, x_p$ on x_1. It is called the *multiple correlation coefficient* of x_1 on $x_2, x_3 \ldots, x_p$ and is denoted by $r_{1.23\ldots p}$. Then

$$r_{1.23\ldots p} = \frac{Cov(x_1, X_1)}{\sqrt{Var(x_1)}\sqrt{Var(X_1)}}. \tag{3.22}$$

Again, the mean of the predicted value X_1 is

$$\begin{aligned}\bar{X}_1 &= \frac{1}{n}\sum_\alpha X_{1\alpha} \\ &= \bar{x}_1 - \frac{R_{12}}{R_{11}}\frac{s_1}{s_2}\frac{1}{n}\sum_\alpha (x_{2\alpha} - \bar{x}_2) - \frac{R_{13}}{R_{11}}\frac{s_1}{s_3}\frac{1}{n}\sum_\alpha (x_{3\alpha} - \bar{x}_3) \\ &\quad \ldots\ldots - \frac{R_{1p}}{R_{11}}\frac{s_1}{s_p}\frac{1}{n}\sum_\alpha (x_{p\alpha} - \bar{x}_p) \text{ [by (3.20)]} \\ &= \bar{x}_1. \end{aligned} \tag{3.23}$$

$$\left[\text{As } \frac{1}{n}\sum (x_{2\alpha} - \bar{x}_2) = \frac{1}{n}\sum x_{2\alpha} - \bar{x}_2 = \bar{x}_2 - \bar{x}_2 = 0, \text{ etc.}\right]$$

The error e_1 is $e_1 = x_1 - X_1$. Then $\bar{e}_1 = \bar{x}_1 - \bar{X}_1 = 0$.

Now, $Cov(x_1, X_1) = \frac{1}{n}\sum_\alpha (x_{1\alpha} - \bar{x}_1)(x_{1\alpha} - \bar{X}_1)$

$$= \frac{1}{n}\sum_\alpha (e_{1\alpha} + X_{1\alpha} - \bar{X}_1)(X_{1\alpha} - \bar{X}_1) = \frac{1}{n}\sum_\alpha e_{1\alpha}(X_{1\alpha} - \bar{X}_1) + \frac{1}{n}\sum_\alpha (X_{1\alpha} - \bar{X}_1)^2$$

$$= \frac{1}{n}\sum_{\alpha} e_{1\alpha}(X_{1\alpha} - \bar{X}_1) + Var(X_1) \tag{3.24}$$

Now, $\frac{1}{n}\sum_{\alpha} e_{1\alpha}(X_{1\alpha} - \bar{X}_1) = \frac{1}{n}\sum_{\alpha} e_{1\alpha}X_{1\alpha} - \bar{X}_1\frac{1}{n}\sum_{\alpha} e_{1\alpha}$

$$= \frac{1}{n}\sum_{\alpha} e_{1\alpha}\{\bar{x}_1 + b_2(x_{2\alpha} - \bar{x}_2) + b_3(x_{3\alpha} - \bar{x}_3) + \cdots + b_p(x_{p\alpha} - \bar{x}_p)\} - 0$$

$$\left[\because \frac{1}{n}\sum e_{1\alpha} = \bar{e}_1 = 0\right]$$

$= 0$. [Using normal equations].

Therefore, $Cov(x_1, X_1) = Var(X_1)$. (3.25)

Now, $Cov(x_1, X_1) = \frac{1}{n}\sum_{\alpha}(x_{1\alpha} - \bar{x}_1)(X_1 - \bar{x}_1)$

$$= \frac{1}{n}\sum_{\alpha}(x_{1\alpha} - \bar{x}_1)\left\{-\frac{R_{12}}{R_{11}}\frac{s_1}{s_2}(x_{2\alpha} - \bar{x}_2) - \frac{R_{13}}{R_{11}}\frac{s_1}{s_3}(x_{3\alpha} - \bar{x}_3) - \cdots - \frac{R_{1p}}{R_{11}}\frac{s_1}{s_p}(x_{p\alpha} - \bar{x}_p)\right\}$$

$$= -\frac{R_{12}}{R_{11}}\frac{s_1}{s_2}s_{12} - \frac{R_{13}}{R_{11}}\frac{s_1}{s_3}s_{13} - \cdots - \frac{R_{1p}}{R_{11}}\frac{s_1}{s_p}s_{1p}$$

$$= -\frac{s_1^2}{R_{11}}(r_{12}R_{12} + r_{13}R_{13} + \cdots + r_{1p}R_{1p})$$

$$= -\frac{s_1^2}{R_{11}}(|R| - r_{11}R_{11}) = \left(1 - \frac{|R|}{R_{11}}\right)s_1^2. \tag{3.26}$$

$$\therefore Var(X_1) = Cov(x_1, X_1) = \left(1 - \frac{|R|}{R_{11}}\right)s_1^2$$

Hence $r_{1.23\ldots p} = \dfrac{\left(1 - \frac{|R|}{R_{11}}\right)s_1^2}{\sqrt{s_1^2\left(1 - \frac{|R|}{R_{11}}\right)s_1^2}} = \left(1 - \frac{|R|}{R_{11}}\right)^{1/2}$ (3.27)

The multiple correlation coefficient basically a simple correlation coefficient and so must lie between –1 and 1. But, $Cov(x_1, X_1) = Var(X_1) > 0$. Thus

$$r_{1.23\cdots p} = \frac{Cov(x_1, X_1)}{\sqrt{Var(x_1)}\sqrt{Var(X_1)}} > 0 \text{ and hence}$$

$$0 \le r_{1.23\cdots p} \le 1. \tag{3.28}$$

3.2.1 Some results on multiple regression and multiple correlation

(i) $\bar{X}_1 = \bar{x}_1$ and $\bar{e}_1 = 0.$ (3.29)

(ii) $Var(X_1) = \left(1 - \frac{|R|}{R_{11}}\right)^{1/2} s_1^2 = r^2_{1.23\cdots p}.s_1^2.$ (3.30)

(iii) Using normal equations it can be shown that

$$Cov(x_i, e_1) = 0, \quad i = 2,3,\cdots,p \tag{3.31}$$

(iv) Since $x_1 = X_1 + e_1$ and $Cov(X_1, e_1) = 0$, it can be shown that

$$Var(x_1) = Var(X_1) + Var(e_1). \tag{3.32}$$

Hence $Var(e_1) = s_1^2 - Var(X_1) = \frac{|R|}{R_{11}} s_1^2.$ (3.33)

The term $Var(e_1)$ being the standard error of estimate and we have

$$Var(e_1) = \left(1 - r^2_{1.23\cdots p}\right)s_1^2. \tag{3.34}$$

Using (3.30) and (3.34) we may write

$$r^2_{1.23\cdots p} = \frac{Var(X_1)}{Var(x_1)} = 1 - \frac{Var(e_1)}{Var(x_1)}. \tag{3.35}$$

Example 3.2.1 For the data of example 3.1.1, the multiple correlation coefficient of weight of dry bark (x_1) on height (x_2) and girth at a height of $6''$ (x_3) may be computed. We have

$$r_{12} = 0.768,\ r_{13} = 0.719 \text{ and } r_{23} = 0.520.$$

The multiple correlation coefficient is

$$r_{1.23} = \sqrt{\frac{r_{12}^2 + r_{13}^2 - 2r_{12}r_{13}r_{23}}{1 - r_{23}^2}}. \tag{3.35a}$$

$$= \sqrt{\frac{0.5325}{0.7296}} = 0.854$$

It indicates that x_2 and x_3 have considerable influence on x_1. It indicates that the multiple regression equation obtained in example 3.1.1 serves as an excellent formula for predicting x_1 from given values of x_2 and x_3.

3.3 Partial Correlation

Sometimes the correlation between two variables say x_1 and x_2, may be partly (or wholly) due to the influence of a group of variables, say $x_3, x_4, \cdots, x_p$ on both x_1 and x_2. In such a situation one may want to know what the correlation between x_1 and x_2 would be if the effects of $x_3, x_4, \cdots, x_p$ on each of them were eliminated. This correlation is called the partial correlation or net correlation between x_1 and x_2, eliminating the effects of $x_3, x_4, \cdots, x_p$, as opposed to their simple or total correlation.

Consider the multiple regression equations of x_1 on $x_3, x_4, \cdots, x_p$ and of x_2 on $x_3, x_4, \cdots, x_p$. Then we write

$$x_1 = X_1' + e_1' \text{ and } x_2 = X_2' + e_2',$$

where X_1' and X_2' are the predicted values of x_1 and x_2, e_1' and e_2' being the errors of estimation. Since e_1' and e_2' are uncorrelated with $x_3, x_4, \cdots, x_p$ these may be looked upon as the parts of x_1 and x_2 respectively, which are unaffected by this group of variables. Hence the simple correlation coefficient between e_1' and e_2' may be used to measure the partial correlation of x_1 and x_2, eliminating the effects of $x_3, x_4, \cdots, x_p$, in so far as this can be done with the help of linear regression equations. This is known as a partial correlation coefficient and is denoted by $r_{12.34\cdots p}$.

Thus, assuming $Var(e_1') > 0$ and $Var(e_2') > 0$, so that R_{11} and R_{22} are both positive, we have

$$r_{12.34\cdots p} = \frac{Cov(e_1', e_2')}{\sqrt{Var(e_1').Var(e_2')}} \tag{3.36}$$

Now,

$$e_1' = x_1 - X_1' = (x_1 - \bar{x}_1) + \frac{R_{13}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_3}(x_3 - \bar{x}_3) + \frac{R_{14}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_4}(x_4 - \bar{x}_4) + \cdots + \frac{R_{1p}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_p}(x_p - \bar{x}_p)$$

where $R_{14}^{(2)}$ is the cofactor of r_{ij} in $R^{(2)}$, the determinant obtained from R by deleting the second row and the second column. Now, putting

$$u_i = x_i - \bar{x}_i \tag{3.37}$$

and $$e_{u_1} = e_1' - \bar{e}_1' = e_1'. \tag{3.38}$$

Therefore,

$$e_{u_1} = u_1 + \frac{R_{13}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_3}u_3 + \frac{R_{14}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_4}u_4 + \cdots + \frac{R_{1p}^{(2)}}{R_{11}^{(2)}}\frac{s_1}{s_p}u_p.$$

Similarly, putting

$$e_{u_2} = e_2' - \bar{e}_2' = e_2' \tag{3.39}$$

We have

$$e_{u_2} = u_2 + \frac{R_{23}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_3}u_3 + \frac{R_{24}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_4}u_4 + \cdots + \frac{R_{2p}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_p}u_p.$$

In the similar way of (3.33), we have

$$Var(e_1') = \frac{R^{(2)}}{R_{11}^{(2)}}s_1^2 \tag{3.40}$$

and $$Var(e_2') = \frac{R^{(1)}}{R_{22}^{(1)}}s_2^2. \tag{3.41}$$

Analogous to (3.26)

$$nCov(e_1', e_2') = \sum_\alpha u_{1\alpha}u_{2\alpha} + \frac{R_{23}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_3}\sum_\alpha u_{1\alpha}u_{2\alpha}$$

$$+ \frac{R_{24}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_4}\sum_\alpha u_{1\alpha}u_{4\alpha} + \cdots + \frac{R_{2p}^{(1)}}{R_{22}^{(1)}}\frac{s_2}{s_p}\sum_\alpha u_{1\alpha}u_{p\alpha}$$

or, $$Cov(e_1', e_2') = s_1 s_2\left(r_{12} + r_{13}\frac{R_{23}^{(1)}}{R_{22}^{(1)}} + r_{14}\frac{R_{24}^{(1)}}{R_{22}^{(1)}} + \cdots + r_{1p}\frac{R_{2p}^{(1)}}{R_{22}^{(1)}}\right) \tag{3.42}$$

$$\left[\text{Since } \sum_\alpha u_{i\alpha}u_{j\alpha} = nCov(x_i, x_j) = nr_{ij}s_i s_j\right]$$

Now, $r_{12}R_{22}^{(1)} + r_{13}R_{23}^{(1)} + r_{14}R_{24}^{(1)} + \cdots + r_{1p}R_{2p}^{(1)}$

= determinant obtained from $R^{(1)}$ by replacing its first row $(r_{22}r_{23}\ldots r_{2p})$ with $(r_{12}r_{13}\ldots r_{1p})$

$$= \begin{vmatrix} r_{12} & r_{13} & \cdots & r_{1p} \\ r_{32} & r_{33} & \cdots & r_{3p} \\ \vdots & \vdots & & \vdots \\ r_{p2} & r_{p3} & \cdots & r_{pp} \end{vmatrix}$$

= minor of r_{21} in R = minor of r_{12} in R

$= -R_{12}$.

Therefore, from (3.42) we have

$$Cov(e_1', e_2') = -\frac{R_{12}}{R_{22}^{(1)}} s_1 s_2 . \tag{3.43}$$

Thus, in terms of the simple or total correlation coefficients r_{ij},

$$r_{12.34...p} = \frac{-\frac{R_{12}}{R_{22}^{(1)}} s_1 s_2}{\left[\frac{R^{(2)}}{R_{11}^{(2)}}\right]^{1/2}\left[\frac{R^{(1)}}{R_{22}^{(1)}}\right]^{1/2} s_1 s_2} = -\frac{R_{12}}{\sqrt{R_{11} R_{22}}} \tag{3.44}$$

since $R^{(1)} = R_{11}$, $R^{(2)} = R_{22}$ and $R_{11}^{(2)} = R_{22}^{(1)}$.

3.3.1 Case of three variables

For the case of three variables x_1, x_2, x_3

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Then $-R_{12} = \begin{vmatrix} r_{21} & r_{23} \\ r_{31} & r_{33} \end{vmatrix} = r_{12} - r_{13} r_{23}$

$$R_{11} = \begin{vmatrix} r_{22} & r_{23} \\ r_{32} & r_{33} \end{vmatrix} = 1 - r_{23}^2$$

and $R_{22} = \begin{vmatrix} r_{11} & r_{13} \\ r_{31} & r_{33} \end{vmatrix} = 1 - r_{13}^2$.

Thus $r_{12.3} = \dfrac{r_{12} - r_{13} r_{23}}{\sqrt{1 - r_{13}^2}\sqrt{1 - r_{23}^2}}$. (3.45)

The value of partial correlation coefficient $r_{12.34...p}$ lies between -1 and 1 i.e.,

$-1 \le r_{12.34...p} \le 1$. (3.46)

Example 3.3.1. Let us consider the data of example 3.1.1. The partial correlation coefficient of x_1 (yield of dry bark) and x_2 (height of plant), the effect of x_3 (girth at a height of 6″) being accounted for, is

$$r_{12.3} = \frac{r_{12} - r_{13}r_{23}}{\sqrt{1-r_{13}^2}\sqrt{1-r_{23}^2}} = \frac{0.394}{\sqrt{0.4830}\sqrt{0.7296}} = 0.663$$

The partial correlation coefficient of x_1 and x_3, eliminating the effect of x_2, is

$$r_{13.2} = \frac{r_{13} - r_{12}r_{23}}{\sqrt{1-r_{12}^2}\sqrt{1-r_{23}^2}} = \frac{0.320}{\sqrt{0.4102}\sqrt{0.7296}} = 0.585.$$

These values may be considered together with the total correlation coefficients

$$r_{12} = 0.768 \text{ and } r_{13} = 0.719$$

Since r_{12} is quite large, one will naturally take x_2 as an independent variable for predicting x_1. The partial correlation $r_{13.2}$, being equal to 0.585, indicates that the inclusion of x_3 as an independent variable, in addition to x_2 would be worth while as it would considerably increase the accuracy of prediction.

3.3.1 Some results

Property 3.4.1 $1-r_{1.23}^2 = (1-r_{12}^2)(1-r_{13.2}^2)$.

Proof:

$$1-r_{13.2}^2 = 1-\left(\frac{r_{13} - r_{12}r_{23}}{\sqrt{1-r_{12}^2}\sqrt{1-r_{23}^2}}\right)^2$$

$$= 1-\frac{(r_{13} - r_{12}r_{23})^2}{(1-r_{12}^2)(1-r_{23}^2)}$$

$$= \frac{1-r_{12}^2-r_{23}^2+r_{12}^2r_{23}^2-r_{13}^2-r_{12}^2r_{23}^2+2r_{13}r_{12}r_{23}}{(1-r_{12}^2)(1-r_{23}^2)}$$

$$= \frac{1-r_{13}^2-r_{23}^2-r_{12}^2+2r_{12}r_{13}r_{23}}{(1-r_{12}^2)(1-r_{23}^2)}$$

or, $$(1-r_{12}^2)(1-r_{13.2}^2) = \frac{1-r_{13}^2-r_{23}^2-r_{12}^2+2r_{12}r_{13}r_{23}}{1-r_{23}^2}$$

$$= 1 - \frac{r_{12}^2 + r_{13}^2 - 2r_{12}r_{13}r_{23}}{1 - r_{23}^2}$$

$$= 1 - r_{1.23}^2$$

Hence $1 - r_{1.23}^2 = (1 - r_{12}^2)(1 - r_{13.2}^2)$.

Property 3.4.2 The correlation coeffieicients r_{12}, r_{13} and r_{23} must satisfy the inequality

$$r_{12}^2 + r_{13}^2 + r_{23}^2 - 2r_{12}r_{13}r_{23} \leq 1.$$

Proof. We know $r_{1.23}^2 \leq 1$. That is,

$$\frac{r_{12}^2 + r_{13}^2 - 2r_{12}r_{13}r_{23}}{1 - r_{23}^2} \leq 1$$

or, $r_{12}^2 + r_{13}^2 - 2r_{12}r_{13}r_{23} \leq 1 - r_{23}^2$

or, $r_{12}^2 + r_{13}^2 + r_{23}^2 - 2r_{12}r_{13}r_{23} \leq 1$.

3.4 Linear Estimation

Here we concerned with point estimation under a special set-up. This will be based on linear models for the expectations and finiteness of first and of second order moments of the observations in the sample. The estimators with which we shall be concerned here are, linear functions of the observations and they are known as linear estimators. Further, we shall consider only unbiased linear estimators of linear functions of parameters.

We know that the sample mean $\bar{Y}$ is an unbiased estimator for the population mean μ, and $\bar{Y} = \frac{1}{n}\sum_i Y_i$ is a linear estimator. Also, we have seen that $S^2 = \frac{1}{n-1}\sum_i (Y_i - \bar{Y})$ is an unbiased estimator of the population variance σ^2, but it is not a linear estimator; S^2 is a quadratic estimator of σ^2. Here we consider only unbiased linear estimators of estimable linear functions of the parameters occurring in the expressions for the expectations of the random variables. We may have more than one unbiassed linear estimator e.g., $\bar{Y}$ and $\sum_i a_i Y_i$ with $\sum_i a_i = 1$, are both unbiassed linear estimators of the population mean. So we have the problem of selection the one that may be taken to be the best in some suitable sense. One widely used principle to make this selection is to choose that linear estimator from amongst all unbiased linear estimators which has the smallest variance. The sampling distribution of that estimator will have the maximum concentration around the unknown true parametric function. Such an

estimator is known as a minimum-variance unbiassed linear estimator or best linear unbiassed estimator (BLUE). These minimum-variance unbiassed linear estimators have variances and covariances which, again themselves unbiassed estimation.

3.4.1 Gauss-Markov linear model

Consider a set of n independent random variables $Y_1, Y_2', \ldots Y_n$ with a common variance σ^2, whose expectations are linear functions with known coefficients $(a_{ij}'s)$ of p unknown parameter $\beta_1, \beta_2, \ldots, \beta_p$ (with $p < n$). Thus

$$\left.\begin{array}{r} E(Y_i) = a_{i1}\beta_1 + a_{i2}\beta_2 + \cdots + a_{ip}\beta_p \\ \text{and} \quad Var(Y_i) = \sigma^2, \text{ for } i = 1,2,\cdots,n, \\ Con(Y_i, Y_j) = 0 \text{ for } i \neq j \end{array}\right\} \tag{3.47}$$

The above system of equations is called the Gauss-Markov linear model. With the help of the following column vectors of the random variables and parameters

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}$$

and the matrix of the known coefficients:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{pmatrix},$$

the equation (3.47) may be written completely as

$$E(Y) = A\beta, D(Y) = \sigma^2 I, \tag{3.48}$$

where $D(Y)$ is called dispersion matrix and I is the identity matrix of order n.

An alternative representation of (3.48), using the column vector e of independent errors $e_1, e_2, \ldots, e_n$, is

$$\left.\begin{aligned} &Y = A\beta + e \\ &E(e) = 0 \text{ and } D(e) = \sigma^2 I \end{aligned}\right\} \tag{3.49}$$

where O is the null vector.

The unknown parameters B_j's in the model are called effects. In linear estimation the effects are all fixed quantities (parameters) and such a model where all effects are unknown parameters is also called a fixed-effect model or Model I. Sometimes one of the β_j's is a constant with $a_{ij} = 1$ for that j and all $i = 1,2,\ldots,n$. Such an effect is called a general effect or an additive constant.

In linear estimation, we find unbiased linear estimators of estimable linear parametric functions starting from a model of the form (3.47). Among unbiased linear estimators, again, we find one having the minimum possible variance. This estimator is considered to be the best, in the sense of having the maximum concentration of the distribution of the estimator around the unknown true value of the parametric function. The value of the estimator in a particular sampling situation will be the corresponding estimate. Thus we obtain the estimate if we put the observed values $y_1, y_2, \ldots, y_n$ for the random variables $Y_1, Y_2, \ldots, Y_n$ in the estimator.

3.4.2 Least-square estimators and normal equations

Let $b_1, b_2, \ldots, b_n$ denote any set of p known quantities which may be used as estimates of $\beta_1, \beta_2, \ldots, \beta_p$. Then for such a $b' = (b_1, b_2, \ldots, b_p)$ we may form the sum of squares $(Y - Ab)'(Y - Ab)$, which will provide us with a measure of how well the estimate b for β fits the model (3.48). The smaller the above measure the better the corresponding estimate. And, from this argument, we get the following set of least-square estimators.

A set of measurable functions of Y, say $\hat{\beta}_1 = \hat{\beta}_1(Y), \hat{\beta}_2 = \hat{\beta}_2(Y), \cdots, \hat{\beta}_p = \hat{\beta}_p(Y)$, such that the values $\hat{\beta}_1, \hat{\beta}_2, \cdots, \hat{\beta}_p$ minimize the sum of squares of the deviations of $Y_1, Y_2, \cdots, Y_n$ from their expectations,

i.e., $S = (Y - A\beta)'(Y - A\beta)$.

is called a set of least-square (LS) estimators of the unknown parameters $\hat{\beta}_1, \hat{\beta}_2, \cdots, \hat{\beta}_p$ of the linear model (3.48).

We next show that the minimum value of S is attained when $\hat{\beta}$ is a solution of a set of equations which are called the normal equations.

We have

$$\hat{\beta}A'Y = Y'AB = \sum_{\alpha,j} a_{\alpha j}\beta_j Y_\alpha$$

where $j = 1,2,\cdots,p$ and $\alpha = 1,2,\cdots,n$.

Hence

$$\frac{d}{d\beta_j}(\beta'B'A'Y) = \frac{d}{d\beta_j}(Y'A\beta) = \sum_{\alpha} a_{\alpha j}Y_\alpha = d_j'Y$$

where $d_1, d_2, \ldots, d_p$ are the column vectors of A.

Let $A'A = C = (C_{ij})$.

Clearly, C is a symmetric matrix of order p.

Now,

$$\beta'A'A\beta = \beta'C\beta = \sum_{i,j} C_{ij}\beta_i\beta_j,$$

with $i, j = 1,2,\ldots,p$

Hence

$$\frac{d}{d\beta_j}(\beta'A'A\beta) = \frac{d}{d\beta_j}(\beta'C\beta) = 2\sum_i C_{ij}\beta_i = 2C_j'\beta,$$

where $C_j' = (C_{1j}, C_{2j}, \cdots C_{pj})$.

Differentiating $S = (Y - A\beta)'(Y - A\beta)$ w.r.t. $\beta_1, \beta_2, \cdots \beta_p$ and equating the derivatives to zero, we have then

$$\frac{1}{2}\frac{d}{d\beta_j}S = -d_j'Y + C_j'\beta = 0 \text{ for } j = 1,2,\cdots,p. \tag{3.50}$$

The equations (3.50) are called the normal equations and equivalent to

$$\text{or,} \quad \left.\begin{array}{l} A'A\beta = A'Y \\ C\beta = A'Y \end{array}\right\} \tag{3.51}$$

where $C = A'A$

The normal equations always admit a solution since $A'Y$ lies in the vector space generated by the column of C. Let $\hat{\beta}$ be a solution of these equations.

Every solution of the normal equations is a set of LS estimators and every set of LS estimators satisfies the

normal equations.

The solution of the normal equations $\beta = \hat{\beta}$ gives an extreme value of S. To show that this extreme value is the minimum value of S, we produce as follows:

$$\begin{aligned}(Y - A\beta)'(Y - A\beta) &= \left[Y - A\hat{\beta} + A\left(\hat{\beta} - \beta\right)\right]'\left[Y - A\hat{\beta} + A\left(\hat{\beta} - \beta\right)\right] \\ &= \left(Y - A\hat{\beta}\right)'\left(Y - A\hat{\beta}\right) + \left(\hat{\beta} - \beta\right)' A' A\left(\hat{\beta} - \beta\right) \\ &\geq \left(Y - A\hat{\beta}\right)'\left(Y - A\hat{\beta}\right) \qquad (4.52)\end{aligned}$$

since the quadratic form $\left[A\left(\hat{\beta} - \beta\right)\right]'\left[A\left(\hat{\beta} - \beta\right)\right]$ cannot be negative. The equality holds only when $\beta = \hat{\beta}$. Thus $\beta = \hat{\beta}$ minimizes S.

Further, if $\hat{\beta}$ and $\hat{\beta}$ are any two solutions of (3.51), then

$$\left(Y - A\hat{\beta}\right)'\left(Y - A\hat{\beta}\right) = \left(Y - A\hat{\hat{\beta}}\right)'\left(Y - A\hat{\hat{\beta}}\right).$$

This along with (4.52) shows that every solution of the normal equations is a set of LS estimators.

3.5 Unit Summary

In this unit the concept of multiple regression and multiple correlation along with partial correlation are introduced. The multiple regression equation for $(p-1)$ independent variables is deduced. An example is also given. The expression for multiple and partial correlation coefficients are deduced. Some relations among simple correlation coefficients, multiple correlation coefficients and partial correlation coefficients are presented. An exercise is also given at the end of this unit.

3.6 Self Assessment Questions

3.1 Let (x_i, y_i, z_i), $i = 1,2,\ldots,n$ be a sample of size n drawn from a population. Find the regression equation of z on x and y.

3.2 Let (2,5,3), (8,3,9), (5,3,6), (5,0,1), (3,-1,2) be a sample of the variables x_1, x_2, x_3 drawn from a random population. Find the regression line of x_2 on x_1 and x_3.

3.3 Obtain the multiple regression equation of x_i on $x_2, x_3, \cdots, x_p$ in terms of the means, the standard deviations and the intercorrelations of the variables.

3.4 Define multiple correlation and partial correlation, and indicate how they differ from simple correlation. Deduce the formulae for a multiple and a partial correlation coefficient in terms of total correlation coefficients.

3.5 Prove that

$$1 - r_{1.23}^2 = (1 - r_{12}^2)(1 - r_{13.2}^2).$$

Use this relation to show that the multiple correlation coefficient is numerically greater than any of the total or partial correlation coefficients of x_1 with the other variables.

3.6 Show that r_{12}, r_{13} and r_{23} must satisfy the inequality

$$r_{12}^2 + r_{13}^2 + r_{23}^2 - 2r_{12}r_{13}r_{23} \leq 1.$$

3.7 The following constants are obtained from measurements on length in mm. (x_1), volume in c.c. (x_2) and weight in gm. (x_3) of 300 eggs:

$\bar{x}_1 = 55.95$	$s_1 = 2.26$	$r_{12} = 0.578$
$\bar{x}_2 = 51.48$	$s_2 = 4.39$	$r_{13} = 0.581$
$\bar{x}_3 = 56.03$	$s_3 = 4.41$	$r_{23} = 0.974$

(a) Obtain the linear regression equation of egg-weight on egg-length and egg-volume. Hence estimate the weight of an egg whose length is 58.0 mm. and volume is 52.5 cc.

(b) Compute the partial correlation coefficient of weight and volume, estimating the effect of length.

3.7 Suggested Further Reading

1. A.M. Goon, M.K. Gupta and B. Dasgupta, Fundamentals of Statistics, Vol.I, The World Press Private Ltd.
2. A.M. Goon, M.K. Gupta and B. Dasgupta, An outline of Statistical Theory, Vol. II., The World Press Private Ltd.
3. B.C. Montgomery, E.A. Peck and G.G. Vining, Introduction to Linear Regression Analysis, 3ed, John Wiley and Sons, Inc.

M.Sc. Course in Applied Mathematics with Oceanology and Computer Programming

PART-I

Paper-III Group-E

Module No.- 28

Numerical Analysis

(Symbolic Operators & Interpolation)

Module Structure:

28.1 Introduction

You have learnt in B.Sc., a very few operators used in numerical methods. Also you have learnt Lagrange's, Newton forward and backward interpolation methods. In this module you will learn some more new operators and other efficient interpolation methods.

Let us consider a function $y = f(x)$ defined on $[a, b]$. The variables x and y are called independent and dependent variables respectively. The points $x_0, x_1, \ldots, x_n$ are taken as equidistance, i.e., $x_i = x_0 + ih$, $i = 0, 1, 2, \ldots, n$. Then the value of y, when $x = x_i$, is denoted by y_i, where $y_i = f(x_i)$. The values of x are called **arguments** and that of y are called **entries**. The interval h is called the difference interval. In this chapter, some important difference operators, viz., forward difference (Δ), backward difference (∇), central difference (δ), shift (E) and mean (μ) are introduced.

28.2 Objectives

Gone through this module the students will learn the following:

- Difference operators
- Gauss forward and backward interpolation
- Stirling interpolation
- Bessel interpolation
- Everett interpolation
- Inverse interpolation

- Cubic spline interpolation.

28.3 Keywords

Operators: E, $\Delta, \nabla, D, \mu, \delta$, Stirling interpolation, Bessel interpolation, Everett interpolation, inverse interpolation, cubic spline interpolation.

28.4 Finite Difference Operators

28.4.1 Forward differences

The **forward difference** or simply **difference** operator is denoted by Δ and is defined by

$$\Delta f(x) = f(x+h) - f(x). \tag{28.1}$$

In terms of y, at $x = x_i$ the above equation gives

$$\Delta f(x_i) = f(x_i + h) - f(x_i), \text{ i.e., } \Delta y_i = y_{i+1} - y_i, i = 0, 1, 2, \ldots, n-1. \tag{28.2}$$

Explicitly, $\Delta y_0 = y_1 - y_0, \Delta y_1 = y_2 - y_1, \ldots, \Delta y_{n-1} = y_n - y_{n-1}$.

The differences of the first differences are called second differences and they are denoted by $\Delta^2 y_0, \Delta^2 y_1, \ldots$. Similarly, one can define third differences, fourth differences, etc.

Thus,

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0 = (y_2 - y_1) - (y_1 - y_0) = y_2 - 2y_1 + y_0$$
$$\Delta^2 y_1 = \Delta y_2 - \Delta y_1 = (y_3 - y_2) - (y_2 - y_1) = y_3 - 2y_2 + y_1$$
$$\Delta^3 y_0 = \Delta^2 y_1 - \Delta^2 y_0 = (y_3 - 2y_2 + y_1) - (y_2 - 2y_1 + y_0) = y_3 - 3y_2 + 3y_1 - y_0$$
$$\Delta^3 y_1 = y_4 - 3y_3 + 3y_2 - y_1$$

and so on.

In general,

$$\Delta^{n+1} f(x) = \Delta[\Delta^n f(x)], \text{ i.e., } \Delta^{n+1} y_i = \Delta[\Delta^n y_i], n = 0, 1, 2, \ldots. \tag{28.3}$$

Also, $\Delta^{n+1} f(x) = \Delta^n [f(x+h) - f(x)] = \Delta^n f(x+h) - \Delta^n f(x)$

and

$$\Delta^{n+1} y_i = \Delta^n y_{i+1} - \Delta^n y_i, n = 0, 1, 2, \ldots, \tag{28.4}$$

where $\Delta^0 \equiv$ identity operator, i.e., $\Delta^0 f(x) = f(x)$ and $\Delta^1 \equiv \Delta$.

28.4.2 Backward differences

The backward difference operator is denoted by ∇ and it is defined as

$$\nabla f(x) = f(x) - f(x-h). \tag{28.5}$$

In terms of y, the above relation transforms to

$$\nabla y_i = y_i - y_{i-1}, \qquad i = n, n-1, \ldots, 1. \tag{28.6}$$

That is,

$$\nabla y_1 = y_1 - y_0, \nabla y_2 = y_2 - y_1, \ldots, \nabla y_n = y_n - y_{n-1}. \tag{28.7}$$

These differences are called first differences. The second differences are denoted by $\nabla^2 y_2, \nabla^2 y_3, \ldots, \nabla^2 y_n$. That is,

$$\nabla^2 y_2 = \nabla(\nabla y_2) = \nabla(y_2 - y_1) = \nabla y_2 - \nabla y_1 = (y_2 - y_1) - (y_1 - y_0) = y_2 - 2y_1 + y_0.$$

Similarly, $\nabla^2 y_3 = y_3 - 2y_2 + y_1, \nabla^2 y_4 = y_4 - 2y_3 + y_2$, and so on.

In general,

$$\nabla^k y_i = \nabla^{k-1} y_i - \nabla^{k-1} y_{i-1}, \qquad i = n, n-1, \ldots, k, \tag{28.8}$$

where $\nabla^0 y_i = y_i, \nabla^1 y_i = \nabla y_i$.

These backward differences can be written in a tabular form and this table is known as **backward difference** or **horizontal** table.

28.4.3 Central differences

The **central difference** operator is denoted by δ and is defined by

$$\delta f(x) = f(x + h/2) - f(x - h/2). \tag{28.9}$$

In terms of y, the first central difference is

$$\delta y_i = y_{i+1/2} - y_{i-1/2} \tag{28.10}$$

where $y_{i+1/2} = f(x_i + h/2)$ and $y_{i-1/2} = f(x_i - h/2)$.

Thus $\delta y_{1/2} = y_1 - y_0, \delta y_{3/2} = y_2 - y_1, \ldots, \delta y_{n-1/2} = y_n - y_{n-1}$.

The second central differences are

$\delta^2 y_i = \delta y_{i+1/2} - \delta y_{i-1/2} = (y_{i+1} - y_i) - (y_i - y_{i-1}) = y_{i+1} - 2y_i + y_{i-1}$.

Table 28.1: Central difference table.

x	y	δ	δ^2	δ^3	δ^4
x_0	y_0				
		$\delta y_{1/2}$			
x_1	y_1		$\delta^2 y_1$		
		$\delta y_{3/2}$		$\delta^3 y_{3/2}$	
x_2	y_2		$\delta^2 y_2$		$\delta^4 y_2$
		$\delta y_{5/2}$		$\delta^3 y_{5/2}$	
x_3	y_3		$\delta^2 y_3$		
		$\delta y_{7/2}$			
x_4	y_4				

In general,

$$\delta^n y_i = \delta^{n-1} y_{i+1/2} - \delta^{n-1} y_{i-1/2}. \tag{28.11}$$

The central difference table for the five arguments $x_0, x_1, \ldots, x_4$ is shown in Table 28.1.

It is observed that all odd differences have fraction suffices and all the even differences are with integral suffices.

28.4.4 Shift, Average and Differential operators

Shift operator, E:

The shift operator is defined by

$$Ef(x) = f(x+h). \tag{28.12}$$

This gives,

$$Ey_i = y_{i+1}. \tag{28.13}$$

That is, shift operator shifts the function value y_i to the next higher value y_{i+1}.

The second shift operator gives

$$E^2 f(x) = E[Ef(x)] = E[f(x+h)] = f(x+2h). \tag{28.14}$$

In general,

$$E^n f(x) = f(x+nh) \text{ or } E^n y_i = y_{i+nh}. \tag{28.15}$$

The inverse shift operator E^{-1} is defined as

$$E^{-1}f(x) = f(x-h). \tag{28.16}$$

Similarly, second and higher inverse operators are

$$E^{-2}f(x) = f(x-2h) \qquad \text{and} \qquad E^{-n}f(x) = f(x-nh). \tag{28.17}$$

More general form of E operator is

$$E^r f(x) = f(x+rh), \tag{28.18}$$

where r is positive as well as negative rationals.

Average operator, μ:

The average operator μ is defined as

$$\mu f(x) = \frac{1}{2}[f(x+h/2) + f(x-h/2)]$$

$$\text{i.e.,} \qquad \mu y_i = \frac{1}{2}[y_{i+1/2} + y_{i-1/2}]. \tag{28.19}$$

Differential operator, D:

The differential operator is usually denoted by D, where

$$Df(x) = \frac{d}{dx}f(x) = f'(x)$$

$$D^2 f(x) = \frac{d^2}{dx^2}f(x) = f''(x). \tag{28.20}$$

28.5 Properties of Forward Differences

Property 28.5.1

$$\begin{aligned}\Delta[f(x)g(x)] &= f(x+h)g(x+h) - f(x)g(x)\\ &= f(x+h)g(x+h) - f(x+h)g(x) + f(x+h)g(x) - f(x)g(x)\\ &= f(x+h)[g(x+h) - g(x)] + g(x)[f(x+h) - f(x)]\\ &= f(x+h)\Delta g(x) + g(x)\Delta f(x).\end{aligned}$$

Also, it can be shown that

$$\begin{aligned}\Delta[f(x)g(x)] &= f(x)\Delta g(x) + g(x+h)\Delta f(x)\\ &= f(x)\Delta g(x) + g(x)\Delta f(x) + \Delta f(x)\Delta g(x).\end{aligned}$$

Property 28.5.2 $\Delta\left[\frac{f(x)}{g(x)}\right] = \frac{g(x)\Delta f(x) - f(x)\Delta g(x)}{g(x+h)g(x)}, g(x) \neq 0.$

Proof.

$$\begin{aligned}\Delta\left[\frac{f(x)}{g(x)}\right] &= \frac{f(x+h)}{g(x+h)} - \frac{f(x)}{g(x)} \\ &= \frac{f(x+h)g(x) - g(x+h)f(x)}{g(x+h)g(x)} \\ &= \frac{g(x)[f(x+h) - f(x)] - f(x)[g(x+h) - g(x)]}{g(x+h)g(x)} \\ &= \frac{g(x)\Delta f(x) - f(x)\Delta g(x)}{g(x+h)g(x)}.\end{aligned}$$

Property 28.5.3 *In particular, when the numerator is 1, then*

$$\Delta\left[\frac{1}{f(x)}\right] = -\frac{\Delta f(x)}{f(x+h)f(x)}.$$

28.5.1 Properties of shift operators

Property 28.5.4 $Ec = c$, *where* c *is a constant.*

Property 28.5.5 $E\{cf(x)\} = cEf(x)$.

Property 28.5.6 $E\{c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)]$
$= c_1 Ef_1(x) + c_2 Ef_2(x) + \cdots + c_n Ef_n(x)$.

Property 28.5.7 $E^m E^n f(x) = E^n E^m f(x) = E^{m+n} f(x)$.

Property 28.5.8 $E^n E^{-n} f(x) = f(x)$.
In particular, $EE^{-1} \equiv I, I$ *is the identity operator and it is some times denoted by 1.*

Property 28.5.9 $(E^n)^m f(x) = E^{mn} f(x)$.

Property 28.5.10 $E\left\{\frac{f(x)}{g(x)}\right\} = \frac{Ef(x)}{Eg(x)}$.

Property 28.5.11 $E\{f(x)\, g(x)\} = Ef(x)\, Eg(x)$.

Property 28.5.12 $E\Delta f(x) = \Delta E f(x)$.

Property 28.5.13 $\Delta^m f(x) = \nabla^m E^m f(x) = E^m \nabla^m f(x)$
and $\nabla^m f(x) = \Delta^m E^{-m} f(x) = E^{-m} \Delta^m f(x)$.

28.6 Relations Among Operators

It is clear from the forward, backward and central difference tables that in a definite numerical case, the same values occur in the same positions, practically there are no differences among the values of the tables, but, different symbols have been used for the theoretical importance.

Thus

$$\Delta y_i = y_{i+1} - y_i = \nabla y_{i+1} = \delta y_{i+1/2}$$
$$\Delta^2 y_i = y_{i+2} - 2y_{i+1} + y_i = \nabla^2 y_{i+2} = \delta^2 y_{i+1}$$

etc.

In general,

$$\Delta^n y_i = \nabla^n y_{i+n}, \qquad i = 0, 1, 2, \ldots. \tag{28.21}$$

Again,

$$\Delta f(x) = f(x+h) - f(x) = Ef(x) - f(x) = (E-1)f(x).$$

This relation indicates that the effect of the operator Δ on $f(x)$ is the same as that of the operator $E-1$ on $f(x)$. Thus

$$\Delta \equiv E - 1 \qquad \text{or} \qquad E \equiv \Delta + 1. \tag{28.22}$$

Also,

$$\nabla f(x) = f(x) - f(x-h) = f(x) - E^{-1}f(x) = (1 - E^{-1})f(x).$$

That is,

$$\nabla \equiv 1 - E^{-1}. \tag{28.23}$$

The higher order forward difference can be expressed in terms of the given function values in the following way:

$$\Delta^3 y_i = (E-1)^3 y_i = (E^3 - 3E^2 + 3E - 1)y_i = y_3 - 3y_2 + 3y_1 - y_0.$$

There is a relation among the central difference, δ, and the shift operator E, as

$$\delta f(x) = f(x + h/2) - f(x - h/2) = E^{1/2}f(x) - E^{-1/2}f(x) = (E^{1/2} - E^{-1/2})f(x).$$

That is,

$$\delta \equiv E^{1/2} - E^{-1/2}. \tag{28.24}$$

Again,

$$\mu f(x) = \frac{1}{2}[f(x + h/2) + f(x - h/2)]$$
$$= \frac{1}{2}[E^{1/2}f(x) + E^{-1/2}f(x)] = \frac{1}{2}(E^{1/2} + E^{-1/2})f(x).$$

Thus,

$$\mu \equiv \frac{1}{2}\left[E^{1/2} + E^{-1/2}\right]. \tag{28.25}$$

The average operator μ can also be expressed in terms of the central difference operator.

$$\begin{aligned}\mu^2 f(x) &= \frac{1}{4}\left[E^{1/2} + E^{-1/2}\right]^2 f(x) \\ &= \frac{1}{4}\left[(E^{1/2} - E^{-1/2})^2 + 4\right] f(x) = \frac{1}{4}\left[\delta^2 + 4\right] f(x).\end{aligned}$$

Hence,

$$\mu \equiv \sqrt{1 + \frac{1}{4}\delta^2}. \tag{28.26}$$

Some more relations among the operators Δ, ∇, E and δ are deduced in the following.

$$\nabla E f(x) = \nabla f(x+h) = f(x+h) - f(x) = \Delta f(x).$$

Also,

$$\delta E^{1/2} f(x) = \delta f(x + h/2) = f(x+h) - f(x) = \Delta f(x).$$

Thus,

$$\Delta \equiv \nabla E \equiv \delta E^{1/2}. \tag{28.27}$$

From the definition of E,

$$\begin{aligned}Ef(x) = f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \cdots \\ &\quad \text{[by Taylor's series]} \\ &= f(x) + hDf(x) + \frac{h^2}{2!}D^2 f(x) + \frac{h^3}{3!}D^3 f(x) + \cdots \\ &= \left[1 + hD + \frac{h^2}{2!}D^2 + \frac{h^3}{3!}D^3 + \cdots\right] f(x) \\ &= e^{hD} f(x).\end{aligned}$$

Hence,

$$E \equiv e^{hD}. \tag{28.28}$$

Also,

$$hD \equiv \log E. \tag{28.29}$$

This relation is used to separate the effect of E into that of the powers of Δ and this method of separation is called the **method of separation of symbols.**

The operators μ and δ can be expressed in terms of D, as shown below

$$\mu f(x) = \frac{1}{2}[E^{1/2} + E^{-1/2}]f(x) = \frac{1}{2}[e^{hD/2} + e^{-hD/2}]f(x)$$
$$= \cosh\left(\frac{hD}{2}\right)f(x)$$
$$\text{and} \quad \delta f(x) = [E^{1/2} - E^{-1/2}]f(x) = [e^{hD/2} - e^{-hD/2}]f(x)$$
$$= 2\sinh\left(\frac{hD}{2}\right)f(x).$$

Thus,

$$\mu \equiv \cosh\left(\frac{hD}{2}\right) \text{ and } \delta \equiv 2\sinh\left(\frac{hD}{2}\right). \tag{28.30}$$

Again,

$$\mu\delta \equiv 2\cosh\left(\frac{hD}{2}\right)\sinh\left(\frac{hD}{2}\right) = \sinh(hD). \tag{28.31}$$

The inverse relation

$$hD \equiv \sinh^{-1}(\mu\delta) \tag{28.32}$$

is also useful.

Since $E \equiv 1 + \Delta$ and $E^{-1} \equiv 1 - \nabla$, [from (28.22) and (28.23)] from (28.29), it is obtained that

$$hD \equiv \log E \equiv \log(1 + \Delta) \equiv -\log(1 - \nabla) \equiv \sinh^{-1}(\mu\delta). \tag{28.33}$$

The operators μ and E are commutative, as

$$\mu E f(x) = \mu f(x + h) = \frac{1}{2}[f(x + 3h/2) + f(x + h/2)],$$

while

$$E\mu f(x) = E\left[\frac{1}{2}\{f(x + h/2) + f(x - h/2)\}\right] = \frac{1}{2}[f(x + 3h/2) + f(x + h/2)].$$

Hence,

$$\mu E \equiv E\mu. \tag{28.34}$$

Example 28.6.1 Prove the following relations.

(i) $1 + \delta^2\mu^2 \equiv \left(1 + \frac{\delta^2}{2}\right)^2$, (ii) $E^{1/2} \equiv \mu + \frac{\delta}{2}$, (iii) $\Delta \equiv \frac{\delta^2}{2} + \delta\sqrt{1 + \frac{\delta^2}{4}}$,

(iv) $(1 + \Delta)(1 - \nabla) \equiv 1$, (v) $\mu\delta \equiv \frac{\Delta E^{-1}}{2} + \frac{\Delta}{2}$, (vi) $\mu\delta \equiv \frac{\Delta + \nabla}{2}$,

(vii) $\Delta\nabla \equiv \nabla\Delta \equiv \delta^2$.

Solution. (i) $\delta\mu f(x) = \frac{1}{2}(E^{1/2} + E^{-1/2})(E^{1/2} - E^{-1/2})f(x) = \frac{1}{2}[E - E^{-1}]f(x)$.

Therefore,

$$(1 + \delta^2\mu^2)f(x) = \left[1 + \frac{1}{4}(E - E^{-1})^2\right]f(x)$$
$$= \left[1 + \frac{1}{4}(E^2 - 2 + E^{-2})\right]f(x) = \frac{1}{4}(E + E^{-1})^2 f(x)$$
$$= \left[1 + \frac{1}{2}(E^{1/2} - E^{-1/2})^2\right]^2 f(x) = \left[1 + \frac{\delta^2}{2}\right]^2 f(x).$$

Hence

$$1 + \delta^2\mu^2 \equiv \left(1 + \frac{\delta^2}{2}\right)^2. \tag{28.35}$$

(ii) $\left(\mu + \frac{\delta}{2}\right)f(x) = \left\{\frac{1}{2}[E^{1/2} + E^{-1/2}] + \frac{1}{2}[E^{1/2} - E^{-1/2}]\right\}f(x) = E^{1/2}f(x)$.

Thus

$$E^{1/2} \equiv \mu + \frac{\delta}{2}. \tag{28.36}$$

(iii)

$$\left[\frac{\delta^2}{2} + \delta\sqrt{1 + \frac{\delta^2}{4}}\right]f(x)$$
$$= \frac{1}{2}(E^{1/2} - E^{-1/2})^2 f(x) + \left[(E^{1/2} - E^{-1/2})\sqrt{1 + \frac{1}{4}(E^{1/2} - E^{-1/2})^2}\right]f(x)$$
$$= \frac{1}{2}[E + E^{-1} - 2]f(x) + \frac{1}{2}(E^{1/2} - E^{-1/2})(E^{1/2} + E^{-1/2})f(x)$$
$$= \frac{1}{2}[E + E^{-1} - 2]f(x) + \frac{1}{2}(E - E^{-1})f(x)$$
$$= (E - 1)f(x).$$

Hence,

$$\frac{\delta^2}{2} + \delta\sqrt{1 + \frac{\delta^2}{4}} \equiv E - 1 \equiv \Delta. \tag{28.37}$$

(iv) $(1 + \Delta)(1 - \nabla)f(x) = (1 + \Delta)[f(x) - f(x) + f(x - h)]$
$= (1 + \Delta)f(x - h) = f(x - h) + f(x) - f(x - h)$
$= f(x)$.

Therefore,

$$(1 + \Delta)(1 - \nabla) \equiv 1. \tag{28.38}$$

(v)

$$\left[\frac{\Delta E^{-1}}{2}+\frac{\Delta}{2}\right]f(x)=\frac{1}{2}[\Delta f(x-h)+\Delta f(x)]$$
$$=\frac{1}{2}[f(x)-f(x-h)+f(x+h)-f(x)]$$
$$=\frac{1}{2}[f(x+h)-f(x-h)]=\frac{1}{2}[E-E^{-1}]f(x)$$
$$=\frac{1}{2}(E^{1/2}+E^{-1/2})(E^{1/2}-E^{-1/2})f(x)$$
$$=\mu\delta f(x).$$

Hence

$$\frac{\Delta E^{-1}}{2}+\frac{\Delta}{2}\equiv\mu\delta. \tag{28.39}$$

(vi)

$$\left[\frac{\Delta+\nabla}{2}\right]f(x)=\frac{1}{2}[\Delta f(x)+\nabla f(x)]$$
$$=\frac{1}{2}[f(x+h)-f(x)+f(x)-f(x-h)]$$
$$=\frac{1}{2}[f(x+h)-f(x-h)]=\frac{1}{2}[E-E^{-1}]f(x)$$
$$=\mu\delta f(x)\qquad\text{(as in previous case).}$$

Thus,

$$\mu\delta\equiv\frac{\Delta+\nabla}{2}. \tag{28.40}$$

(vii) $\Delta\nabla f(x)=\Delta[f(x)-f(x-h)]=f(x+h)-2f(x)+f(x-h)$.
Again,

$$\nabla\Delta f(x)=f(x+h)-2f(x)+f(x-h)=(E-2+E^{-1})f(x)$$
$$=(E^{1/2}-E^{-1/2})^2f(x)=\delta^2f(x).$$

Hence,

$$\Delta\nabla\equiv\nabla\Delta\equiv(E^{1/2}-E^{-1/2})^2\equiv\delta^2. \tag{28.41}$$

The relations among the various operators are shown in Table 28.2.
From definition of derivative, we have

$$f'(x)=\lim_{h\to0}\frac{f(x+h)-f(x)}{h}=\lim_{h\to0}\frac{\Delta f(x)}{h}.$$

Thus one can write,

$$\Delta f(x)\simeq hf'(x).$$

Table 28.2: Relationship between the operators.

	E	Δ	∇	δ	hD
E	E	$\Delta+1$	$(1-\nabla)^{-1}$	$1+\frac{\delta^2}{2}+\delta\sqrt{1+\frac{\delta^2}{4}}$	e^{hD}
Δ	$E-1$	Δ	$(1-\nabla)^{-1}-1$	$\frac{\delta^2}{2}+\delta\sqrt{1+\frac{\delta^2}{4}}$	$e^{hD}-1$
∇	$1-E^{-1}$	$1-(1+\Delta)^{-1}$	∇	$-\frac{\delta^2}{2}+\delta\sqrt{1+\frac{\delta^2}{4}}$	$1-e^{-hD}$
δ	$E^{1/2}-E^{-1/2}$	$\Delta(1+\Delta)^{-1/2}$	$\nabla(1-\nabla)^{-1/2}$	δ	$2\sinh(hD/2)$
μ	$\frac{E^{1/2}+E^{-1/2}}{2}$	$(1+\Delta/2)\times(1+\Delta)^{-1/2}$	$(1-\nabla/2)(1-\nabla)^{-1/2}$	$1+\frac{\delta^2}{4}$	$\cosh(hD/2)$
hD	$\log E$	$\log(1+\Delta)$	$-\log(1-\nabla)$	$2\sinh^{-1}(\delta/2)$	hD

Again,

$$f''(x) = \lim_{h\to 0}\frac{f'(x+h)-f'(x)}{h}$$
$$\simeq \lim_{h\to 0}\frac{\frac{\Delta f(x+h)}{h}-\frac{\Delta f(x)}{h}}{h}$$
$$= \lim_{h\to 0}\frac{\Delta f(x+h)-\Delta f(x)}{h^2} = \lim_{h\to 0}\frac{\Delta^2 f(x)}{h^2}.$$

Therefore, $h^2 f''(x) \simeq \Delta^2 f(x)$.

In general, $\Delta^n f(x) \simeq h^n f^n(x)$. That is, for small h, the operators Δ and hD are almost equal.

28.7 Worked out Examples

Example 28.7.1 Find the missing term in the following table:

x	:	1	2	3	4	5
$f(x)$	:	-2	3	8	-	21

Solution. Here four values of $f(x)$ are given. So, we consider $f(x)$ be a polynomial of degree 3. Thus the fourth differences of $f(x)$ vanish, i.e.,

$\Delta^4 f(x) = 0$ or, $(E-1)^4 f(x) = 0$

or, $(E^4 - 4E^3 + 6E^2 - 4E + 1)f(x) = 0$

or, $E^4 f(x) - 4E^3 f(x) + 6E^2 f(x) - 4Ef(x) + f(x) = 0$

or, $f(x+4) - 4f(x+3) + 6f(x+2) - 4f(x+1) + f(x) = 0$.

Here, $h = 1$ as the values are in spacing of 1 unit.

For $x = 1$ the above equation becomes

$f(5) - 4f(4) + 6f(3) - 4f(2) + f(1) = 0$ or, $21 - 4f(4) + 6 \times 8 - 4 \times 3 - 2 = 0$

or, $f(4) = 13.75$.

Example 28.7.2 Use finite difference method to find the values of a and b in the following table.

x	:	0	2	4	6	8	10
$f(x)$	:	-5	a	8	b	20	32

Solution. Here, four values of $f(x)$ are known, so we can assume that $f(x)$ is a polynomial of degree 3.

Then, $\Delta^4 f(x) = 0$.

or, $(E-1)^4 f(x) = 0$

or, $E^4 f(x) - 4E^3 f(x) + 6E^2 f(x) - 4Ef(x) + f(x) = 0$

or, $f(x+8) - 4f(x+6) + 6f(x+4) - 4f(x+2) + f(x) = 0$

[Here $h = 2$, because the values of x are given in 2 unit interval]

In this problem, two unknowns a and b are to be determined and needs two equations. Therefore, the following equations are obtained by substituting $x = 2$ and $x = 0$ to the above equation.

$f(10) - 4f(8) + 6f(6) - 4f(4) + f(2) = 0$ and

$f(8) - 4f(6) + 6f(4) - 4f(2) + f(0) = 0$.

These equations are simplifies to

$32 - 4 \times 20 + 6b - 4 \times 8 + a = 0$ and $20 - 4b + 6 \times 8 - 4a - 5 = 0$.

That is, $6b + a - 80 = 0$ and $-4b - 4a + 63 = 0$. Solution of these equations is $a = 2.9, b = 12.85$.

Example 28.7.3 Find the value of $\left(\frac{\Delta^2}{E}\right)x^2$.

Solution.

$$\left(\frac{\Delta^2}{E}\right)x^2 = \left[\frac{(E-1)^2}{E}\right]x^2$$

$$= \left[\frac{E^2 - 2E + 1}{E}\right]x^2$$

$$= (E - 2 + E^{-1})x^2 = Ex^2 - 2x^2 + E^{-1}x^2$$

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

$$= 2.$$

Example 28.7.4 Show that $\nabla \log f(x) = -\log\left[1 - \frac{\nabla f(x)}{f(x)}\right]$.

Solution.

$$\begin{aligned}\nabla \log f(x) &= \log f(x) - \log f(x-h) = \log f(x) - \log E^{-1} f(x) \\ &= \log \frac{f(x)}{E^{-1} f(x)} = -\log \frac{E^{-1} f(x)}{f(x)} = -\log \frac{(1-\nabla) f(x)}{f(x)} \\ &= -\log \frac{f(x) - \nabla f(x)}{f(x)} = -\log \left[1 - \frac{\nabla f(x)}{f(x)}\right].\end{aligned}$$

28.8 Interpolation

In undergraduates courses you have learn about the Lagrange's, Newton forward and backward interpolation formulae. Here will shall discuss some central difference interpolation formulae, viz., Stirling and Bessel, and cubic spline interpolation.

Central Difference Interpolation Formulae

28.9 Gaussian Interpolation Formulae

Newton's forward and Newton's backward formulae does not give accurate value of $f(x)$ when x is in the middle of the table. To get more accurate result another formula may be used. There are several methods available to solve this type of problem. Among them Gaussian forward and backward, Stirling's and Bessel's interpolation formulae are widely used.

28.9.1 Gauss's forward difference formula

Case I: For $2n+1$ (odd) arguments

Suppose the values of the function $y = f(x)$ are known at $2n+1$ equally spaced points $x_{-n}, x_{-(n-1)}, \ldots, x_{-1}, x_0, x_1, \ldots, x_{n-1}, x_n$, i.e., $y_i = f(x_i)$, $i = 0, \pm 1, \pm 2, \ldots, \pm n$.

The problem is to construct a polynomial $\phi(x)$ of degree at most $2n$ such that

$$\phi(x_i) = y_i, i = 0, \pm 1, \pm 2, \ldots, \pm n, \tag{28.42}$$

where $x_i = x_0 + ih$, h is the spacing.

Let us consider $\phi(x)$ as

$$\begin{aligned}\phi(x) = a_0 &+ a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_3(x - x_{-1})(x - x_0)(x - x_1) \\ &+ a_4(x - x_{-1})(x - x_0)(x - x_1)(x - x_2) + \cdots \\ &+ a_{2n-1}(x - x_{-n+1})(x - x_{-n+2}) \cdots (x - x_{-1})(x - x_0) \cdots (x - x_{n-1}) \\ &+ a_{2n}(x - x_{-n+1})(x - x_{-n+2}) \cdots (x - x_{-1})(x - x_0) \cdots (x - x_n),\end{aligned} \tag{28.43}$$

where a_i's are unknown constants and their values are to be determined by substituting $x = x_0, x_1, x_{-1}, x_2, x_{-2}, \ldots, x_n, x_{-n}$.

Therefore,

$$y_0 = a_0$$

$$y_1 = a_0 + a_1(x_1 - x_0) \text{ i.e., } y_1 = y_0 + a_1 h,$$

$$\text{i.e., } a_1 = \frac{y_1 - y_0}{x_1 - x_0} = \frac{\Delta y_0}{h}.$$

$$y_{-1} = y_0 + a_1(-h) + a_2(-h)(-2h)$$

$$= y_0 - h\frac{\Delta y_0}{h} + a_2 h^2 \cdot 2!$$

$$\text{i.e., } a_2 = \frac{y_{-1} - 2y_0 + y_1}{2!\, h^2} = \frac{\Delta^2 y_{-1}}{2!\, h^2}.$$

$$y_2 = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

$$+a_3(x_2 - x_{-1})(x_2 - x_0)(x_2 - x_1)$$

$$= y_0 + \frac{y_1 - y_0}{h}(2h) + \frac{y_{-1} - 2y_0 + y_1}{2!h^2}(2h)(h) + a_3(3h)(2h)(h)$$

$$\text{or, } a_3 = \frac{y_2 - 3y_1 + 3y_0 - y_{-1}}{3!h^3} = \frac{\Delta^3 y_{-1}}{3!h^3}.$$

In this manner, the remaining values are obtained as

$$a_4 = \frac{\Delta^4 y_{-2}}{4!h^4}, a_5 = \frac{\Delta^5 y_{-2}}{5!h^5}, \ldots, a_{2n-1} = \frac{\Delta^{2n-1} y_{-(n-1)}}{(2n-1)!h^{2n-1}}, a_{2n} = \frac{\Delta^{2n} y_{-n}}{(2n)!h^{2n}}.$$

Thus the Gauss's forward difference formula is

$$\phi(x) = y_0 + (x - x_0)\frac{\Delta y_0}{h} + (x - x_0)(x - x_1)\frac{\Delta^2 y_{-1}}{2!h^2}$$

$$+(x - x_{-1})(x - x_0)(x - x_1)\frac{\Delta^3 y_{-1}}{3!h^3} + \cdots$$

$$+(x - x_{-(n+1)}) \cdots (x - x_{n-1})\frac{\Delta^{2n-1} y_{-(n-1)}}{(2n-1)!h^{2n-1}}$$

$$+(x - x_{-(n+1)}) \cdots (x - x_{n-1})(x - x_n)\frac{\Delta^{2n} y_{-n}}{(2n)!h^{2n}}. \quad (28.44)$$

To simplify the above equation, a new variable s is introduced, where $s = \frac{x - x_0}{h}$ i.e, $x = x_0 + sh$.

Also, $x_{\pm i} = x_0 \pm ih, i = 0, 1, 2, \ldots, n$.

Therefore, $x - x_{\pm i} = (s \pm i)h$.

Then $x - x_0 = sh, x - x_1 = (s-1)h, x - x_{-1} = (s+1)h$,

$x - x_2 = (s-2)h, x - x_{-2} = (s+2)h$ and so on.

Making use of these results, (28.44) becomes

$$\phi(x) = y_0 + sh\frac{\Delta y_0}{h} + sh(s-1)h\frac{\Delta^2 y_{-1}}{2!h^2} + (s+1)hsh(s-1)h\frac{\Delta^3 y_{-1}}{3!h^3} + \cdots$$
$$+(s+\overline{n-1})h\cdots sh(s-1)h\cdots(s-\overline{n-1})h\frac{\Delta^{2n-1}y_{-(n-1)}}{(2n-1)!h^{2n-1}}$$
$$+(s+\overline{n-1})h\cdots sh(s-1)h\cdots(s-\overline{n-1})h(s-n)h\frac{\Delta^{2n}y_{-n}}{(2n)!h^{2n}}$$

$$= y_0 + s\Delta y_0 + s(s-1)\frac{\Delta^2 y_{-1}}{2!} + (s+1)s(s-1)\frac{\Delta^3 y_{-1}}{3!} + \cdots$$
$$+(s+\overline{n-1})\cdots s(s-1)\cdots(s-\overline{n-1})\frac{\Delta^{2n-1}y_{-(n-1)}}{(2n-1)!}$$
$$+(s+\overline{n-1})\cdots s(s-1)\cdots(s-\overline{n-1})(s-n)\frac{\Delta^{2n}y_{-n}}{(2n)!}$$
$$= y_0 + s\Delta y_0 + s(s-1)\frac{\Delta^2 y_{-1}}{2!} + s(s^2-1^2)\frac{\Delta^3 y_{-1}}{3!}$$
$$+s(s^2-1^2)(s-2)\frac{\Delta^4 y_{-2}}{4!} + \cdots$$
$$+s(s^2-\overline{n-1}^2)(s^2-\overline{n-2}^2)\cdots(s^2-1^2)\frac{\Delta^{2n-1}y_{-(n-1)}}{(2n-1)!}$$
$$+s(s^2-\overline{n-1}^2)(s^2-\overline{n-2}^2)\cdots(s^2-1^2)(s-n)\frac{\Delta^{2n}y_{-n}}{(2n)!}. \qquad (28.45)$$

The formula (28.44) or (28.45) is known as **Gauss's forward central difference formula** or **the first interpolation formula of Gauss.**

Case II: For $2n$ (even) arguments

In this case the arguments are $x_0, x_{\pm 1}, \ldots, x_{\pm(n-1)}$ and x_n.

For these points the Gauss's forward interpolation takes the following form.

$$\begin{aligned}\phi(x) &= y_0 + s\Delta y_0 + s(s-1)\frac{\Delta^2 y_{-1}}{2!} + (s+1)s(s-1)\frac{\Delta^3 y_{-1}}{3!} \\ &\quad +(s+1)s(s-1)(s-2)\frac{\Delta^4 y_{-2}}{4!} \\ &\quad +(s+2)(s+1)s(s-1)(s-2)\frac{\Delta^5 y_{-2}}{5!} + \cdots \\ &\quad +(s+\overline{n-1})\cdots s\cdots(s-\overline{n-1})\frac{\Delta^{2n-1} y_{-(n-1)}}{(2n-1)!} \\ &= y_0 + s\Delta y_0 + s(s-1)\frac{\Delta^2 y_{-1}}{2!} + s(s^2-1^2)\frac{\Delta^3 y_{-1}}{3!} \\ &\quad +s(s^2-1^2)(s-2)\frac{\Delta^4 y_{-2}}{4!} + (s^2-2^2)(s^2-1^2)s\frac{\Delta^5 y_{-2}}{5!} + \cdots \\ &\quad +(s^2-\overline{n-1}^2)\cdots(s^2-1^2)s\frac{\Delta^{2n-1} y_{-(n-1)}}{(2n-1)!}. \end{aligned} \tag{28.46}$$

28.9.2 Remainder in Gauss's forward central difference formula

The remainder of Gauss's forward central difference interpolation for $2n+1$ arguments is

$$\begin{aligned}E(x) &= (x-x_{-n})(x-x_{-(n-1)})\cdots(x-x_{-1})(x-x_0)\cdots(x-x_n)\frac{f^{2n+1}(\xi)}{(2n+1)!} \\ &= (s+n)(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)(s-n) \\ &\qquad \times h^{2n+1}\frac{f^{2n+1}(\xi)}{(2n+1)!} \\ &= s(s^2-1^2)\cdots(s^2-n^2).h^{2n+1}\frac{f^{2n+1}(\xi)}{(2n+1)!} \end{aligned} \tag{28.47}$$

where $s = \dfrac{x-x_0}{h}$ and ξ lies between $\min\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\}$ and $\max\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\}$.

In case of $2n$ arguments, the error is

$$\begin{aligned}E(x) &= (s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)(s-n)h^{2n}\frac{f^{2n}(\xi)}{(2n)!} \\ &= s(s^2-1^2)\cdots(s^2-\overline{n-1}^2)(s-n).h^{2n}\frac{f^{2n}(\xi)}{(2n)!}, \end{aligned} \tag{28.48}$$

where, $\min\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\} < \xi$
$< \max\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\}$.

28.9.3 Gauss's backward difference formula

Case I: For $2n+1$ (odd) number of arguments

Let a function $y = f(x)$ is known for $2n+1$ equispaced arguments $x_{\pm i}, i = 0, 1, 2, \ldots, n$, such that

$$x_{\pm i} = x_0 \pm ih, i = 0, 1, 2, \ldots, n.$$

Let $y_{\pm i} = f(x_{\pm i}), i = 0, 1, 2, \ldots, n.$

Our aim is to determine a polynomial $\phi(x)$ of degree not more than $2n$ which satisfies the conditions

$$\phi(x_{\pm i}) = y_{\pm i}, i = 0, 1, \ldots, n. \tag{28.49}$$

The polynomial $\phi(x)$ is considered in the following form.

$$\begin{aligned}\phi(x) = a_0 &+ a_1(x - x_0) + a_2(x - x_{-1})(x - x_0) + a_3(x - x_{-1})(x - x_0)(x - x_1) \\ &+ a_4(x - x_{-2})(x - x_{-1})(x - x_0)(x - x_1) \\ &+ a_5(x - x_{-2})(x - x_{-1})(x - x_0)(x - x_1)(x - x_2) + \cdots \\ &+ a_{2n-1}(x - x_{-(n-1)}) \cdots (x - x_{-1})(x - x_0) \cdots (x - x_{n-1}) \\ &+ a_{2n}(x - x_{-n})(x - x_{-(n-1)}) \cdots (x - x_{-1})(x - x_0) \cdots (x - x_{n-1}).\end{aligned} \tag{28.50}$$

The coefficients a_i's are unknown constants. These values are determined by using the relations (28.49). Substituting $x = x_0, x_{-1}, x_1, x_{-2}, x_2, \ldots, x_{-n}, x_n$ to (28.50) in succession. Note that $x_i - x_{-j} = (i+j)h$ and $(x_{-i} - x_j) = -(i+j)h$. Then it is found that

$$y_0 = a_0$$

$$\phi(x_{-1}) = a_0 + a_1(x_{-1} - x_0)$$

$$\text{i.e., } y_{-1} = y_0 + a_1(-h),$$

$$\text{i.e., } a_1 = \frac{y_0 - y_{-1}}{h} = \frac{\Delta y_{-1}}{h}$$

$$\phi(x_1) = a_0 + a_1(x_1 - x_0) + a_2(x_1 - x_{-1})(x_1 - x_0)$$

$$y_1 = y_0 + h.\frac{\Delta y_{-1}}{h} + a_2(2h)(h)$$

$$\text{i.e., } a_2 = \frac{y_1 - y_0 - (y_0 - y_{-1})}{2!h^2} = \frac{\Delta^2 y_{-1}}{2!h^2}$$

$$\phi(x_{-2}) = a_0 + a_1(x_{-2} - x_0) + a_2(x_{-2} - x_{-1})(x_{-2} - x_0)$$
$$+a_3(x_{-2} - x_{-1})(x_{-2} - x_0)(x_{-2} - x_1)$$
$$\text{i.e.,}\quad y_{-2} = y_0 + \frac{\Delta y_{-1}}{h}(-2h) + \frac{\Delta^2 y_{-1}}{2!h^2}(-h)(-2h) + a_3(-h)(-2h)(-3h)$$
$$= y_0 - 2(y_0 - y_{-1}) + (y_1 - 2y_0 + y_{-1}) + a_3(-1)^3(3!)h^3$$
$$\text{or,}\quad a_3 = \frac{y_1 - 3y_0 + 3y_{-1} - y_{-2}}{3!h^3} = \frac{\Delta^3 y_{-2}}{3!h^3}.$$

In this manner, the other values are obtained as

$$a_4 = \frac{\Delta^4 y_{-2}}{4!h^4}, a_5 = \frac{\Delta^5 y_{-3}}{5!h^5}, \ldots, a_{2n-1} = \frac{\Delta^{2n-1} y_{-n}}{(2n-1)!h^{2n-1}}, a_{2n} = \frac{\Delta^{2n} y_{-n}}{(2n)!h^{2n}}.$$

Making use of a_i's, equation (28.50) becomes

$$\phi(x) = y_0 + (x - x_0)\frac{\Delta y_{-1}}{1!h} + (x - x_{-1})(x - x_0)\frac{\Delta^2 y_{-1}}{2!h^2}$$
$$+(x - x_{-1})(x - x_0)(x - x_1)\frac{\Delta^3 y_{-2}}{3!h^3}$$
$$+(x - x_{-2})(x - x_{-1})(x - x_0)(x - x_1)\frac{\Delta^4 y_{-2}}{4!h^4} + \cdots$$
$$+(x - x_{-(n-1)})\cdots(x - x_{-1})(x - x_0)\cdots(x - x_{n-1})\frac{\Delta^{2n-1} y_{-n}}{(2n-1)!h^{2n-1}}$$
$$+(x - x_{-n})(x - x_{-1})(x - x_0)(x - x_1)\cdots(x - x_{n-1})\frac{\Delta^{2n} y_{-n}}{(2n)!h^{2n}}. \qquad (28.51)$$

As in previous case, a new unit less variable s is introduced to reduce the above formula into a simple form, where $s = \frac{x - x_0}{h}$ i.e, $x = x_0 + sh$.

Then

$$\frac{x - x_i}{h} = \frac{x - x_0 - ih}{h} = s - i \text{ and}$$
$$\frac{x - x_{-i}}{h} = \frac{x - x_0 + ih}{h} = s + i,\quad i = 0, 1, 2, \ldots, n.$$

Then the above formula is transferred to

$$\phi(x) = y_0 + s\Delta y_{-1} + \frac{(s+1)s}{2!}\Delta^2 y_{-1} + \frac{(s+1)s(s-1)}{3!}\Delta^3 y_{-2}$$
$$+\frac{(s+2)(s+1)s(s-1)}{4!}\Delta^4 y_{-2} + \cdots$$
$$+\frac{(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-n}$$
$$+\frac{(s+n)(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n)!}\Delta^{2n} y_{-n}. \qquad (28.52)$$

The above formula (28.52) is known as **Gauss's backward interpolation formula** or **second interpolation formula of Gauss.**

Case II: For $2n$ (even) number of arguments

In this case the arguments are taken as $x_0, x_{\pm 1}, \ldots, x_{\pm(n-1)}$ and x_{-n}, where $x_{\pm i} = x_0 \pm ih$, $i = 0, 1, \ldots, n-1$ and $x_{-n} = x_0 - nh$.

For these equispaced points the Gauss's backward interpolation formula is

$$\begin{aligned}\phi(x) = y_0 + s\Delta y_{-1} + \frac{(s+1)s}{2!}\Delta^2 y_{-1} + \frac{(s+1)s(s-1)}{3!}\Delta^3 y_{-2} \\ + \frac{(s+2)(s+1)s(s-1)}{4!}\Delta^4 y_{-2} + \cdots \\ + \frac{(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-n},\end{aligned} \tag{28.53}$$

where $s = \dfrac{x - x_0}{h}$.

28.9.4 Remainder of Gauss's backward central difference formula

The remainder for $(2n+1)$ equispaced points is

$$\begin{aligned}E(x) &= (x - x_{-n})(x - x_{-(n-1)})\cdots(x - x_{-1})(x - x_0)\cdots(x - x_n)\frac{f^{2n+1}(\xi)}{(2n+1)!} \\ &= (s+n)(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)(s-n) \\ &\qquad \times h^{2n+1}\frac{f^{2n+1}(\xi)}{(2n+1)!},\end{aligned}$$

where $\min\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\} < \xi$
$< \max\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_n\}$.

The remainder for the case of $2n$ equispaced points is

$$\begin{aligned}E(x) &= (x - x_{-n})(x - x_{-(n-1)})\cdots(x - x_{-1})(x - x_0)\cdots(x - x_{n-1})\frac{f^{2n}(\xi)}{(2n)!} \\ &= (s+n)(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)h^{2n}\frac{f^{2n}(\xi)}{(2n)!},\end{aligned}$$

$\min\{x_{-n}, x_{-n}, \ldots, x_0, x_1, \ldots, x_{n-1}, x_{n-1}\} < \xi$
$< \max\{x_{-n}, x_{-(n-1)}, \ldots, x_0, x_1, \ldots, x_{n-1}\}$.

28.10 Stirling's Interpolation Formula

Stirling's interpolation formula is used for odd number of equispaced arguments.

This formula is obtained by taking the arithmetic mean of the Gauss's forward and backward difference formulae given by (28.45) and (28.52).

Therefore Stirling's formula is

$$\begin{aligned}\phi(x) &= \frac{\phi(x)_{\text{forward}} + \phi(x)_{\text{backward}}}{2} \\ &= y_0 + \frac{s}{1!}\frac{\Delta y_{-1} + \Delta y_0}{2} + \frac{s^2}{2!}\Delta^2 y_{-1} + \frac{s(s^2-1^2)}{3!}\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2} \\ &\quad + \frac{s^2(s^2-1^2)}{4!}\Delta^4 y_{-2} + \frac{s(s^2-1^2)(s^2-2^2)}{5!}\frac{\Delta^5 y_{-3} + \Delta^5 y_{-2}}{2} + \cdots \\ &\quad + \frac{s^2(s^2-1^2)(s^2-2^2)\cdots(s^2-\overline{n-1}^2)}{(2n)!}\Delta^{2n} y_{-n}. \end{aligned} \tag{28.54}$$

The remainder of this formula is

$$E(x) = \frac{s(s^2-1^2)(s^2-2^2)\cdots(s^2-n^2)}{(2n+1)}h^{2n+1}f^{2n+1}(\xi), \tag{28.55}$$

where $\min\{x_{-n},\ldots,x_0,\ldots,x_n\} < \xi < \max\{x_{-n},\ldots,x_0,\ldots,x_n\}$.

The formula (28.54) is known as **Stirling's central difference interpolation formula.**

Note 28.10.1 (a) The Stirling's interpolation formula (28.54) gives the best approximate result when $-0.25 < s < 0.25$. So we choose x_0 in such a way that $s = \dfrac{x - x_0}{h}$ satisfy this condition.

(b) The Stirling's interpolation formula is used when the point x, for which $f(x)$ to be determined, is at the centre of the table and the number of points at which the values of $f(x)$ known is *odd.*

28.11 Bessel's Interpolation Formula

This central difference formula is also obtained by taking the arithmetic mean of Gauss's forward and backward interpolation formulae. But, one difference is that the backward formula taken after one modification.

Let us consider $2n$ equispaced points $x_{-(n-1)},\ldots,x_{-1},x_0,x_1,\ldots,x_{n-1},x_n$ as arguments, where $x_{\pm i} = x_0 \pm ih$, h is the spacing.

If x_0, y_0 be the initial values of x and y respectively, then the Gauss's backward difference interpolation formula (28.53) is

$$\begin{aligned}\phi(x) &= y_0 + s\Delta y_{-1} + \frac{s(s+1)}{2!}\Delta^2 y_{-1} + \frac{(s+1)s(s-1)}{3!}\Delta^3 y_{-2} \\ &\quad + \frac{(s+2)(s+1)s(s-1)}{4!}\Delta^4 y_{-2} + \cdots \\ &\quad + \frac{(s+n-1)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-n}. \end{aligned} \tag{28.56}$$

Suppose x_1, y_1 be the initial values of x and y. Then

$$\frac{x - x_1}{h} = \frac{x - (x_0 + h)}{h} = \frac{x - x_0}{h} - 1 = s - 1.$$

Also, the indices of all the differences of (28.56) will increase by 1. Now, replacing s by $s-1$ and increasing the indices of (28.56) by 1, then the above equation becomes

$$\begin{aligned}\phi_1(x) = y_1 &+ (s-1)\Delta y_0 + \frac{s(s-1)}{2!}\Delta^2 y_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 y_{-1} \\ &+ \frac{(s+1)s(s-1)(s-2)}{4!}\Delta^4 y_{-1} \\ &+ \frac{(s+1)s(s-1)(s-2)(s-3)}{5!}\Delta^5 y_{-2} + \cdots \\ &+ \frac{(s+n-2)\cdots(s+1)s(s-1)(s-2)\cdots(s-n)}{(2n-1)!}\Delta^{2n-1} y_{-n+1}. \end{aligned} \tag{28.57}$$

Taking arithmetic mean of (28.57) and Gauss's forward interpolation formula given by (28.46),

$$\begin{aligned}\phi(x) &= \frac{\phi_1(x) + \phi(x)_{\text{forward}}}{2} \\ &= \frac{y_0+y_1}{2} + \left(s-\frac{1}{2}\right)\Delta y_0 + \frac{s(s-1)}{2!}\frac{\Delta^2 y_0 + \Delta^2 y_{-1}}{2} \\ &+ \frac{(s-\frac{1}{2})s(s-1)}{3!}\Delta^3 y_{-1} + \frac{s(s-1)(s+1)(s-2)}{4!}\frac{\Delta^4 y_{-2} + \Delta^4 y_{-1}}{2} \\ &+ \frac{(s-\frac{1}{2})s(s-1)(s+1)(s-2)}{5!}\Delta^5 y_{-2} + \cdots \\ &+ \frac{(s-\frac{1}{2})s(s-1)(s+1)\cdots(s+n-2)(s-\overline{n-1})}{(2n-1)!}\Delta^{2n-1} y_{-(n-1)}, \end{aligned} \tag{28.58}$$

where $s = \dfrac{x-x_0}{h}$.

Introducing $u = s - \dfrac{1}{2} = \dfrac{x-x_0}{h} - \dfrac{1}{2}$ and then the above formula reduces to

$$\begin{aligned}\phi(x) = \frac{y_0+y_1}{2} &+ u\Delta y_0 + \frac{u^2-\frac{1}{4}}{2!}\cdot\frac{\Delta^2 y_{-1} + \Delta^2 y_0}{2} + \frac{u(u^2-\frac{1}{4})}{3!}\Delta^3 y_{-1} \\ &+ \frac{(u^2-\frac{1}{4})(u^2-\frac{9}{4})\cdots(u^2-\frac{(2n-3)^2}{4})}{(2n-1)!}\Delta^{2n-1} y_{-(n-1)}. \end{aligned} \tag{28.59}$$

The formulae given by (28.58) and (28.59) are known as Bessel's central difference interpolation formula.

Note 28.11.1 (a) The Bessel's formula gives the best result when the starting point x_0 be so chosen such that $-0.25 < u < 0.25$ i.e., $-0.25 < s - 0.5 < 0.25$ or, $0.25 < s < 0.75$.

(b) This central difference formula is used when the interpolating point is near the middle of the table and the number of arguments is even.

Example 28.11.1 Use the central difference interpolation formula of Stirling or Bessel to find the values of y at (i) $x = 1.40$ and (ii) $x = 1.60$ from the following table

x :	1.0	1.25	1.50	1.75	2.00
y :	1.0000	1.0772	1.1447	1.2051	2.2599

Solution. The central difference table is

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
−2	1.00	1.0000			
			0.0772		
−1	1.25	1.0772		−0.0097	
			0.0675		0.0026
0	1.50	1.1447		−0.0071	
			0.0604		0.0015
1	1.75	1.2051		−0.0056	
			0.0548		
2	2.00	1.2599			

(i) For $x = 1.40$, we take $x_0 = 1.50$, then $s = (1.40 - 1.50)/0.25 = -0.4$.

The Bessel's formula gives

$$\begin{aligned} y(1.40) &= \frac{y_0 + y_1}{2} + \left(s - \frac{1}{2}\right)\Delta y_0 + \frac{s(s-1)}{2!}\frac{\Delta^2 y_0 + \Delta^2 y_{-1}}{2} \\ &\quad + \frac{1}{3!}\left(s - \frac{1}{2}\right)s(s-1)\Delta^3 y_{-1} \\ &= \frac{1.1447 + 1.2051}{2} + (-0.4 - 0.5) \times 0.0604 \\ &\quad + \frac{-0.4(-0.4-1)}{2!}\frac{-0.0071 - 0.0056}{2} \\ &\quad + \frac{1}{6}(-0.4 - 0.5)(-0.4)(-0.4-1) \times 0.0015 \\ &= 1.118636. \end{aligned}$$

(ii) For $x = 1.60$, we take $x_0 = 1.50$, then $s = (1.60 - 1.50)/0.25 = 0.4$.

Using Stirling's formula

$$\begin{aligned} y(1.60) &= y_0 + s\frac{\Delta y_{-1} + \Delta y_0}{2} + \frac{s^2}{2!}\Delta^2 y_{-1} + \frac{s(s^2 - 1^2)}{3!}\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2} \\ &= 1.1447 + 0.4\frac{0.0675 + 0.0604}{2} + \frac{(0.4)^2}{2} \times (-0.0071) \\ &\quad + \frac{0.4(0.16 - 1)}{6}\frac{0.0026 + 0.0015}{2} \\ &= 1.1447 + 0.02558 - 0.000568 - 0.0001148 = 1.1695972. \end{aligned}$$

28.12 Everett's Interpolation Formula

In this interpolation formula, $2n$ (even) equispaced arguments $x_{-(n-1)}, x_{-(n-2)}, \ldots, x_{-1}, x_0, x_1, \ldots, x_{n-1}$ and x_n are considered. The Everett's interpolation formula is obtained from Gauss's forward interpolation formula (28.45) by replacing the odd order differences using the lower order even differences. That is, by the substitution

$$\Delta^{2k+1} y_{-k} = \Delta^{2k} y_{-(k-1)} - \Delta^{2k} y_{-k}.$$

That is,

$$\begin{aligned}
\Delta y_0 &= y_1 - y_0; \\
\Delta^3 y_{-1} &= \Delta^2 y_0 - \Delta^2 y_{-1}; \\
\Delta^5 y_{-2} &= \Delta^4 y_{-1} - \Delta^4 y_{-2}, \\
&\vdots \quad \vdots \quad \vdots \\
\Delta^{2n-1} y_{-(n-1)} &= \Delta^{2n-2} y_{-(n-2)} - \Delta^{2n-2} y_{-(n-1)}.
\end{aligned}$$

On substitution of these relations, equation (28.45) yields

$$\begin{aligned}
\phi(x) = y_0 &+ s(y_1 - y_0) + \frac{s(s-1)}{2!}\Delta^2 y_{-1} + \frac{(s+1)s(s-1)}{3!}(\Delta^2 y_0 - \Delta^2 y_{-1}) \\
&+ \frac{(s+1)s(s-1)(s-2)}{4!}\Delta^4 y_{-2} \\
&+ \frac{(s+2)(s+1)s(s-1)(s-2)}{5!}(\Delta^4 y_{-1} - \Delta^4 y_{-2}) + \cdots \\
&+ \frac{(s+n-1)(s+n-2)\cdots(s+1)s(s-1)(s-n+1)}{(2n-1)!} \\
&\qquad \times (\Delta^{2n-2} y_{-(n-2)} - \Delta^{2n-2} y_{-(n-1)}) \\
= \Big[&s y_1 + \frac{(s+1)s(s-1)}{3!}\Delta^2 y_0 + \frac{(s+2)(s+1)s(s-1)(s-2)}{5!}\Delta^4 y_{-1} + \cdots
\end{aligned}$$

$$+\frac{(s+n-1)(s+n-2)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n-1)!}\Delta^{2n-2}y_{-(n-2)}\Big]$$
$$+\Big[(1-s)y_0+\Big\{\frac{s(s-1)}{2!}-\frac{(s+1)s(s-1)}{3!}\Big\}\Delta^2y_{-1}$$
$$+\Big\{\frac{(s+1)s(s-1)(s-2)}{4!}-\frac{(s+2)(s+1)s(s-1)(s-2)}{5!}\Big\}\Delta^4y_{-2}+\cdots$$
$$+\frac{(s+n-1)(s+n-2)\cdots(s+1)s(s-1)\cdots(s-n+1)}{(2n-1)!}\Delta^{2n-2}y_{-(n-1)}\Big]$$
$$=\Big[sy_1+\frac{s(s^2-1^2)}{3!}\Delta^2y_0+\frac{s(s^2-1^1)(s^2-2^2)}{5!}\Delta^4y_{-1}+\cdots$$
$$+\frac{s(s^2-1^2)(s^2-2^2)\cdots(s^2-(n-1)^2)}{(2n-1)!}\Delta^{2n-2}y_{-(n-2)}\Big]$$
$$+\Big[uy_0+\frac{u(u^2-1^1)}{3!}\Delta^2y_{-1}+\frac{u(u^2-1^2)(u^2-2^2)}{5!}\Delta^4y_{-2}+\cdots$$
$$+\frac{u(u^2-1^2)(u^2-2^2)\cdots(u^2-(n-1)^2)}{(2n-1)!}\Delta^{2n-2}y_{-(n-1)}\Big], \tag{28.60}$$

where $s=\dfrac{x-x_0}{h}$ and $u=1-s$.

Example 28.12.1 Use Everett's interpolation formula to find the value of y when $x=1.60$ from the following table.

x :	1.0	1.25	1.50	1.75	2.00	2.25
y :	1.0000	1.1180	1.2247	1.3229	1.4142	1.5000

Solution. The difference table is

i	x_i	y_i	Δy_i	Δ^2y_i	Δ^3y_i	Δ^4y_i
−2	1.00	1.0000				
			0.1180			
−1	1.25	1.1180		−0.0113		
			0.1067		0.0028	
0	1.50	1.2247		−0.0085		−0.0012
			0.0982		0.0016	
1	1.75	1.3229		−0.0069		−0.0002
			0.0913		0.0014	
2	2.00	1.4142		−0.0055		
			0.0858			
3	2.25	1.5000				

We take $x_0=1.50$. Here $h=0.25$.

Then $s = \dfrac{x - x_0}{h} = \dfrac{1.60 - 1.50}{0.25} = 0.4$ and $u = 1 - s = 0.6$.
Using Everett's formula, the value of $y(1.60)$ is given by

$$\begin{aligned}
y(1.60) &= \left[sy_1 + \frac{s(s^2 - 1^2)}{3!}\Delta^2 y_0 + \frac{s(s^2 - 1^2)(s^2 - 2^2)}{5!}\Delta^4 y_{-1}\right] \\
&\quad + \left[uy_0 + \frac{u(u^2 - 1^2)}{3!}\Delta^2 y_{-1} + \frac{u(u^2 - 1^2)(u^2 - 2^2)}{5!}\Delta^4 y_{-2}\right] \\
&= \left[0.4 \times 1.3229 + \frac{0.4(0.16 - 1)}{6}(-0.0069)\right. \\
&\quad \left. + \frac{0.4(0.16 - 1)(0.16 - 4)}{120}(-0.0002)\right] + \left[0.6 \times 1.2247\right. \\
&\quad \left. + \frac{0.6(0.36 - 1)}{6}(-0.0085) + \frac{0.6(0.36 - 1)(0.36 - 4)}{120}(-0.0012)\right] \\
&= 0.5292 + 0.0004 - 0.0000025 + 0.7348 + 0.0005 - 0.00001 \\
&= 1.2649.
\end{aligned}$$

Central difference table

The difference Table 28.1 shows how the different order of differences are used in different interpolation formulae.

Note 28.12.1 In Newton's forward and backward interpolation formulae the first or the last interpolating point is taken as initial point x_0. But, in central difference interpolation formulae, a middle point is taken as the initial point x_0.

28.13 Inverse Interpolation

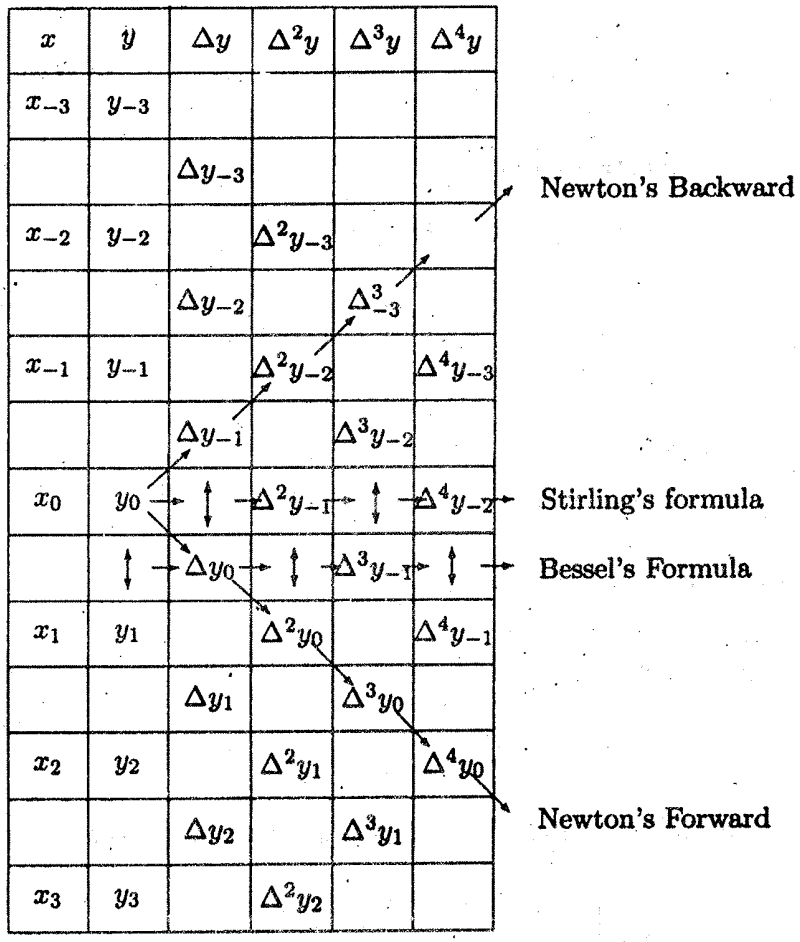
In interpolation, for a given set of values of x and y, the value of y is determined for a given value of x. But the inverse interpolation is the process which finds the value of x for a given y. Commonly used inverse interpolation formulae are based on successive iteration.

In the following, three inverse interpolation formulae based on Lagrange, Newton forward and Newton backward interpolation formulae are described. The inverse interpolation based on Lagrange's formula is a direct method while the formulae based on Newton's interpolation formulae are iterative.

28.13.1 Inverse interpolation based on Lagrange's formula

The Lagrange's interpolation formula of y on x is

$$y = \sum_{i=0}^{n} \frac{w(x)\, y_i}{(x - x_i)w'(x_i)}.$$



x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
x_{-3}	y_{-3}				
		Δy_{-3}			
x_{-2}	y_{-2}		$\Delta^2 y_{-3}$		
		Δy_{-2}		Δ^3_{-3}	
x_{-1}	y_{-1}		$\Delta^2 y_{-2}$		$\Delta^4 y_{-3}$
		Δy_{-1}		$\Delta^3 y_{-2}$	
x_0	y_0		$\Delta^2 y_{-1}$		$\Delta^4 y_{-2}$
		Δy_0		$\Delta^3 y_{-1}$	
x_1	y_1		$\Delta^2 y_0$		$\Delta^4 y_{-1}$
		Δy_1		$\Delta^3 y_0$	
x_2	y_2		$\Delta^2 y_1$		$\Delta^4 y_0$
		Δy_2		$\Delta^3 y_1$	
x_3	y_3		$\Delta^2 y_2$		

Figure 28.1: Central difference table.

When x and y are interchanged then the above relation changes to

$$x = \sum_{i=0}^{n} \frac{w(y)x_i}{(y - y_i)w'(y_i)} = \sum_{i=0}^{n} L_i(y)x_i,$$

where

$$L_i(y) = \frac{w(y)}{(y-y_i)w'(y_i)} = \frac{(y-y_0)(y-y_1)\cdots(y-y_{i-1})(y-y_{i+1})\cdots(y-y_n)}{(y_i-y_0)(y_i-y_1)\cdots(y_i-y_{i-1})(y_i-y_{i+1})\cdots(y_i-y_n)}.$$

This formula gives the value of x for given value of y and the formula is known as **Lagrange's inverse interpolation** formula.

28.13.2 Method of successive approximations

Based on Newton's forward difference interpolation formula

The Newton's forward difference interpolation formula is

$$y = y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \cdots + \frac{u(u-1)(u-2)\cdots(u-\overline{n-1})}{n!}\Delta^n y_0,$$

where $u = \dfrac{x - x_0}{h}$.

The above formula can be written as

$$u = \frac{1}{\Delta y_0}\left[y - y_0 - \frac{u(u-1)}{2!}\Delta^2 y_0 - \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 - \cdots - \frac{u(u-1)(u-2)\cdots(u-\overline{n-1})}{n!}\Delta^n y_0\right].$$

Let the first approximation of u be denoted by $u^{(1)}$ and it is obtained by neglecting the second and higher differences as

$$u^{(1)} = \frac{1}{\Delta y_0}(y - y_0).$$

Next, the second approximation, $u^{(2)}$, is obtained by neglecting third and higher order differences as follows:

$$u^{(2)} = \frac{1}{\Delta y_0}\left[y - y_0 - \frac{u^{(1)}(u^{(1)}-1)}{2!}\Delta^2 y_0\right].$$

Similarly, the third approximation $u^{(3)}$ is given by

$$u^{(3)} = \frac{1}{\Delta y_0}\left[y - y_0 - \frac{u^{(2)}(u^{(2)}-1)}{2!}\Delta^2 y_0 - \frac{u^{(2)}(u^{(2)}-1)(u^{(2)}-2)}{3!}\Delta^3 y_0\right].$$

In general,

$$u^{(k+1)} = \frac{1}{\Delta y_0}\left[y - y_0 - \frac{u^{(k)}(u^{(k)}-1)}{2!}\Delta^2 y_0 - \frac{u^{(k)}(u^{(k)}-1)(u^{(k)}-2)}{3!}\Delta^3 y_0 - \cdots - \frac{u^{(k)}(u^{(k)}-1)\cdots(u^{(k)}-k)}{(k+1)!}\Delta^{k+1} y_0\right],$$

$k = 0, 1, 2, \ldots$.

This process of approximation should be continued till two successive approximations $u^{(k+1)}$ and $u^{(k)}$ be equal up to desired number of decimal places. Then the value of x is obtained from the relation $x = x_0 + u^{(k+1)}h$.

Example 28.13.1 From the table of values

x	:	1.8	2.0	2.2	2.4	2.6
y	:	3.9422	4.6269	5.4571	6.4662	7.6947

find x when $y = 5.0$ using the method of successive approximations.

Solution. The difference table is

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
1.8	3.9422			
		0.6847		
2.0	**4.6269**		0.1455	
		0.8302		0.0334
2.2	5.4571		**0.1789**	
		1.0091		**0.0405**
2.4	6.4662		0.2194	
		1.2285		
2.6	7.6947			

Let $x_0 = 2.0, h = 0.2$. The value of u is determined by successive approximation.
The first approximation is

$$u^{(1)} = \frac{1}{\Delta y_0}(y - y_0) = \frac{1}{0.8302}(5.0 - 4.6269) = 0.4494.$$

$$u^{(2)} = \frac{1}{\Delta y_0}\left[y - y_0 - \frac{u^{(1)}(u^{(1)} - 1)}{2!}\Delta^2 y_0\right] = u^{(1)} - \frac{u^{(1)}(u^{(1)} - 1)}{2!}\frac{\Delta^2 y_0}{\Delta y_0}$$

$$= 0.4494 - \frac{0.4494(0.4494 - 1)}{2}\frac{0.1789}{0.8302} = 0.4761.$$

$$u^{(3)} = u^{(1)} - \frac{u^{(2)}(u^{(2)} - 1)}{2}\frac{\Delta^2 y_0}{\Delta y_0} - \frac{u^{(2)}(u^{(2)} - 1)(u^{(2)} - 2)}{3!}\frac{\Delta^3 y_0}{\Delta y_0}$$

$$= 0.4494 - \frac{0.4761(0.4761 - 1)}{2} \cdot \frac{0.1789}{0.8302}$$

$$- \frac{0.4761(0.4761 - 1)(0.4761 - 2)}{6} \cdot \frac{0.0405}{0.8302}$$

$$= 0.4494 + 0.0269 - 0.0031 = 0.4732.$$

Thus the value of x is given by $x = x_0 + u^{(3)}h = 2.0 + 0.4732 \times 0.2 = 2.0946$.

28.14 Spline Interpolation

Spline interpolation is very powerful and widely used method and has many applications in numerical differentiation, integration, solution of boundary value problems, two and three - dimensional graph plotting

etc. Spline interpolation method, interpolates a function between a given set of points by means of piecewise smooth polynomials. In this interpolation, the curve passes through the given set of points and also its slope and its curvature are continuous at each point. The splines with different degree are found in literature, among them cubic splines are widely used.

28.14.1 Cubic spline

A function $y(x)$ is called **cubic spline** in $[x_0, x_n]$ if there exist cubic polynomials $p_0(x), p_1(x), \ldots, p_{n-1}(x)$ such that

$$y(x) = p_i(x) \text{ on } [x_i, x_{i+1}], i = 0, 1, 2, \ldots, n-1. \tag{28.61}$$

$$p'_{i-1}(x_i) = p'_i(x_i), i = 1, 2, \ldots, n-1 \text{ (equal slope).} \tag{28.62}$$

$$p''_{i-1}(x_i) = p''_i(x_i), i = 1, 2, \ldots, n-1 \text{ (equal curvature).} \tag{28.63}$$

$$\text{and } p_i(x_i) = y_i, \quad p_i(x_{i+1}) = y_{i+1}, i = 0, 1, \ldots, n-1. \tag{28.64}$$

It may be noted that, at the endpoints x_0 and x_n, no continuity on slope and curvature are assigned. The conditions at these points are assigned, generally, depending on the applications.

Let the interval $[x_i, x_{i+1}]$, $i = 0, 1, \ldots, n-1$ be denoted by ith interval.

Let $h_i = x_i - x_{i-1}, i = 1, 2, \ldots, n$ and $M_i = y''(x_i), i = 0, 1, 2, \ldots, n$.

Let the cubic spline for the ith interval be

$$y(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i, \quad \text{in } [x_i, x_{i+1}]. \tag{28.65}$$

Since it passes through the end points x_i and x_{i+1}, therefore,

$$y_i = d_i \tag{28.66}$$

and

$$\begin{aligned} y_{i+1} &= a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i \\ &= a_i h_{i+1}^3 + b_i h_{i+1}^2 + c_i h_{i+1} + d_i. \end{aligned} \tag{28.67}$$

Equation (28.65) is differentiated twice and obtained the following equations.

$$y'(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i. \tag{28.68}$$

$$\text{and } y''(x) = 6a_i(x - x_i) + 2b_i. \tag{28.69}$$

From (28.69), $y''_i = 2b_i$ and $y''_{i+1} = 6a_i h_{i+1} + 2b_i$,
that is, $M_i = 2b_i$, $M_{i+1} = 6a_i h_{i+1} + 2b_i$.

Therefore,

$$b_i = \frac{M_i}{2}, \tag{28.70}$$

$$a_i = \frac{M_{i+1} - M_i}{6h_{i+1}}. \tag{28.71}$$

Using (28.66), (28.70) and (28.71), equation (28.67) becomes

$$y_{i+1} = \frac{M_{i+1} - M_i}{6h_{i+1}} h_{i+1}^3 + \frac{M_i}{2} h_{i+1}^2 + c_i h_{i+1} + y_i$$

$$i.e., \quad c_i = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{2h_{i+1}M_i + h_{i+1}M_{i+1}}{6}. \tag{28.72}$$

Thus, the coefficients a_i, b_i, c_i and d_i of (28.65) are determined in terms of $n+1$ unknowns $M_0, M_1, \ldots, M_n$. These unknowns are determined in the following way.

The condition of equation (28.62) state that the slopes of the two cubics p_{i+1} and p_i are same at x_i.

Now, for the ith interval

$$y_i' = 3a_i(x_i - x_i)^2 + 2b_i(x_i - x_i) + c_i = c_i, \tag{28.73}$$

and for the $(i-1)$th interval

$$y_i' = 3a_{i-1}(x_i - x_{i-1})^2 + 2b_{i-1}(x_i - x_{i-1}) + c_{i-1}. \tag{28.74}$$

Equating (28.73) and (28.74), we obtain

$$c_i = 3a_{i-1}h_i^2 + 2b_{i-1}h_i + c_{i-1}.$$

The values of $a_{i-1}, b_{i-1}, c_{i-1}$ and c_i are substituted to the above equation and obtained the following equation.

$$\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{2h_{i+1}M_i + h_{i+1}M_{i+1}}{6}$$

$$= 3\left(\frac{M_i - M_{i-1}}{6h_i}\right)h_i^2 + \left(\frac{M_i}{2}\right)h_i + \frac{y_i - y_{i-1}}{h_i} - \frac{2h_iM_{i-1} + h_iM_i}{6}.$$

After simplification the above equation reduces to

$$h_iM_{i-1} + 2(h_i + h_{i+1})M_i + h_{i+1}M_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right). \tag{28.75}$$

This relation is true for $i = 1, 2, \ldots, n-1$. Thus $n-1$ equations are available for the $n+1$ unknown quantities $M_0, M_1, \ldots, M_n$. Now, two more conditions are required to solve these equations uniquely. These conditions can be assumed to take one of the following forms:

(i) $M_0 = M_n = 0$. If this conditions are satisfied then the corresponding spline is called **natural spline.**

(ii) $M_0 = M_n, M_1 = M_{n+1}, y_0 = y_n, y_1 = y_{n+1}, h_1 = h_{n+1}$. The corresponding spline satisfying these conditions is called **periodic spline.**

(iii) $y'(x_0) = y'_0, y'(x_n) = y'_n$, i.e.,

$$2M_0 + M_1 = \frac{6}{h_1}\left(\frac{y_1 - y_0}{h_1} - y'_0\right)$$

$$\text{and} \quad M_{n-1} + 2M_n = \frac{6}{h_n}\left(y'_n - \frac{y_n - y_{n-1}}{h_n}\right). \tag{28.76}$$

The corresponding spline satisfying the above conditions is called **non-periodic spline** or **clamped cubic spline.**

(iv) If $M_0 = M_1 - \dfrac{h_1(M_2 - M_1)}{h_2}$ and $M_n = M_{n-1} + \dfrac{h_n(M_{n-1} - M_{n-2})}{h_{n-1}}$. The corresponding spline is called **extrapolated spline.**

(v) If $M_0 = y''_0$ and $M_n = y''_n$ are specified. If a spline satisfy these conditions then it is called **endpoint curvature-adjusted spline.**

Let $A_i = h_i, B_i = 2(h_i + h_{i+1}), C_i = h_{i+1}$ and $D_i = 6\left(\dfrac{y_{i+1} - y_i}{h_{i+1}} - \dfrac{y_i - y_{i-1}}{h_i}\right)$.

Then (28.75) becomes

$$A_i M_{i-1} + B_i M_i + C_i M_{i+1} = D_i, i = 1, 2, \ldots, n-1. \tag{28.77}$$

The system of equations (28.77) is basically a tri-diagonal system. Depending on different conditions, the tri-diagonal systems are different and they are stated below.

For natural spline, the tri-diagonal system for $M_1, M_2, \ldots, M_{n-1}$ is

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & A_{n-1} & B_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_{n-1} \end{bmatrix} \tag{28.78}$$

and $M_0 = M_n = 0$.

Imposing the conditions for non-periodic spline, we find

$$2M_0 + M_1 = D_0 \tag{28.79}$$

$$\text{and } M_{n-1} + 2M_n = D_n, \tag{28.80}$$

$$\text{where } D_0 = \frac{6}{h_1}\left(\frac{y_1 - y_0}{h_1} - y'_0\right)$$

$$\text{and } D_n = \frac{6}{h_n}\left(y'_n - \frac{y_n - y_{n-1}}{h_n}\right). \tag{28.81}$$

Then equations (28.77), (28.79), (28.80) and (28.81) result the following tri-diagonal system for the unknowns $M_0, M_1, \ldots, M_n$.

$$\begin{bmatrix} 2 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & A_2 & B_2 & C_2 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \vdots \\ D_{n-1} \\ D_n \end{bmatrix}. \tag{28.82}$$

For the extrapolated spline the values of M_0 and M_n are given by the relations

$$M_0 = M_1 - \frac{h_1(M_2 - M_1)}{h_2} \text{ and } M_n = M_{n-1} + \frac{h_n(M_{n-1} - M_{n-2})}{h_{n-1}}. \tag{28.83}$$

The first expression is rewritten as

$$M_1\left[A_1 + B_1 + \frac{A_1 h_1}{h_2}\right] + M_2\left[C_1 - \frac{A_1 h_1}{h_2}\right] = D_1 \text{ or, } M_1 B_1' + M_2 C_1' = D_1$$

where $B_1' = A_1 + B_1 + \dfrac{A_1 h_1}{h_2}$ and $C_1' = C_1 - \dfrac{A_1 h_1}{h_2}$.

Similarly, the second expression is transferred to $M_{n-2}A_{n-1}' + M_{n-1}B_{n-1}' = D_{n-1}$ where

$$A_{n-1}' = A_{n-1} - \frac{C_{n-1}h_n}{h_{n-1}} \text{ and } B_{n-1}' = B_{n-1} + C_{n-1} + \frac{h_n C_{n-1}}{h_{n-1}}.$$

For this case, the tri-diagonal system of equations for $M_1, M_2, \ldots, M_{n-1}$ is

$$\begin{bmatrix} B_1' & C_1' & 0 & 0 & \cdots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & A_{n-1}' & B_{n-1}' \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_{n-1} \end{bmatrix}. \tag{28.84}$$

The values of M_0 and M_n are obtained from the equation (28.83).

For the endpoint curvature-adjusted spline, the values of M_0 and M_n are respectively y_0'' and y_n'', where y_0'' and y_n'' are specified. The values of $M_1, M_2, \ldots, M_{n-1}$ are given by solving the following tri-diagonal system of equations

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \cdots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & A_{n-1} & B_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} D_1' \\ D_2 \\ \vdots \\ D_{n-2} \\ D_{n-1}' \end{bmatrix}, \tag{28.85}$$

where $D_1' = D_1 - A_1 y_0''$, $\quad D_{n-1}' = D_{n-1} - C_{n-1} y_n''$.

Example 28.14.1 Fit a cubic spline curve that passes through (0, 0.0), (1, 0.5), (2, 2.0) and (3, 1.5) with the natural end boundary conditions, $y''(0) = y''(3) = 0$.

Solution. Here the intervals are (0, 1) (1, 2) and (2, 3), i.e., three intervals of x, in each of which we can construct a cubic spline. These piecewise cubic spline polynomials together gives the cubic spline curve $y(x)$ in the entire interval (0, 3).

Here $h_1 = h_2 = h_3 = 1$.

Then equation (28.75) becomes

$$M_{i-1} + 4M_i + M_{i+1} = 6(y_{i+1} - 2y_i + y_{i-1}), \quad i = 1, 2, 3.$$

That is,

$$M_0 + 4M_1 + M_2 = 6(y_2 - 2y_1 + y_0) = 6 \times (2.0 - 2 \times 0.5 + 0.0) = 6$$
$$M_1 + 4M_2 + M_3 = 6(y_3 - 2y_2 + y_1) = 6 \times (1.5 - 2 \times 2.0 + 0.5) = -12.$$

Imposing the conditions $M_0 = y''(0) = 0$ and $M_3 = y''(3) = 0$ to the above equations, and they simplify as

$$4M_1 + M_2 = 6, \qquad M_1 + 4M_2 = -12.$$

These equations give $M_1 = \dfrac{12}{5}$ and $M_2 = -\dfrac{18}{5}$.

Let the natural cubic spline be given by

$$p_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

where the coefficients a_i, b_i, c_i and d_i are given by the relations

$$a_i = \frac{M_{i+1} - M_i}{6h_{i+1}}, \qquad b_i = \frac{M_i}{2},$$
$$c_i = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{2h_{i+1}M_i + h_{i+1}M_{i+1}}{6}, \qquad d_i = y_i,$$

for $i = 0, 1, 2$.

Therefore,

$$a_0 = \frac{M_1 - M_0}{6} = 0.4, \qquad b_0 = \frac{M_0}{2} = 0,$$
$$c_0 = \frac{y_1 - y_0}{1} - \frac{2M_0 + M_1}{6} = 0.1, \qquad d_0 = y_0 = 0.$$
$$a_1 = \frac{M_2 - M_1}{6} = -1, \qquad b_1 = \frac{M_1}{2} = \frac{6}{5},$$

$$c_1 = \frac{y_2 - y_1}{1} - \frac{2M_1 + M_2}{6} = 1.3, \qquad d_1 = y_1 = 0.5.$$
$$a_2 = \frac{M_3 - M_2}{6} = \frac{3}{5}, \qquad b_2 = \frac{M_2}{2} = -\frac{9}{5},$$
$$c_2 = \frac{y_3 - y_2}{1} - \frac{2M_2 + M_3}{6} = 0.7, \qquad d_2 = y_2 = 2.0.$$

Hence the required piecewise cubic splines in each interval are given by

$$p_0(x) = 0.4x^3 + 0.1x, \qquad 0 \le x \le 1$$
$$p_1(x) = -(x-1)^3 + 1.2(x-1)^2 + 1.3(x-1) + 0.5, \qquad 1 \le x \le 2$$
$$p_2(x) = 0.6(x-2)^3 - 1.8(x-2)^2 + 0.7(x-2) + 2.0, \qquad 2 \le x \le 3.$$

Example 28.14.2 Fit a cubic spline curve for the following data with end conditions $y'(0) = 0.2$ and $y'(3) = -1$.

x :	0	1	2	3
y :	0	0.5	3.5	5

Solution. Here, the three intervals (0, 1) (1, 2) and (2, 3) are given in each of which the cubic splines are to be constructed. These cubic spline functions are denoted by y_0, y_1 and y_2. In this example, $h_1 = h_2 = h_3 = 1$.

For the boundary conditions, equation (28.75) is used. That is,

$$M_0 + 4M_1 + M_2 = 6(y_2 - 2y_1 + y_0)$$
$$M_1 + 4M_2 + M_3 = 6(y_3 - 2y_2 + y_1).$$

Also, from the equations (28.76)

$$2M_0 + M_1 = 6(y_1 - y_0 - y_0') \text{ and } M_2 + 2M_3 = 6(y_3' - y_3 + y_2)$$

i.e.,

$$M_0 + 4M_1 + M_2 = 15$$
$$M_1 + 4M_2 + M_3 = -9$$
$$2M_0 + M_1 = 1.8$$
$$M_2 + 2M_3 = 6(-1 - 5 + 3.5) = -15.$$

These equations give $M_0 = -1.36, M_1 = 4.52, M_2 = -1.72$ and $M_3 = -6.64$.
Let the cubic spline in each interval be given by

$$y_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i.$$

The coefficients are computed as

$$a_i = \frac{M_{i+1} - M_i}{6h_{i+1}}, \qquad b_i = \frac{M_i}{2},$$
$$c_i = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{2h_{i+1}M_i + h_{i+1}M_{i+1}}{6}, \qquad d_i = y_i, \text{ for } i = 0, 1, 2.$$

Therefore,

$$a_0 = \frac{M_1 - M_0}{6} = 0.98, \quad b_0 = -0.68, \quad c_0 = 0.2, \quad d_0 = 0.$$
$$a_1 = \frac{M_2 - M_1}{6} = -1.04, \quad b_1 = 2.26, \quad c_1 = 1.78, \quad d_1 = 0.5.$$
$$a_2 = \frac{M_3 - M_2}{6} = -0.82, \quad b_2 = -0.86, \quad c_2 = 3.18, \quad d_2 = 3.5.$$

Hence, the required piecewise cubic spline polynomials in each interval are given by

$$y_0(x) = 0.98x^3 - 0.68x^2 + 0.2x, \qquad 0 \le x \le 1$$
$$y_1(x) = -1.04(x-1)^3 + 2.26(x-1)^2 + 1.78(x-1) + 0.5, \qquad 1 \le x \le 2$$
$$y_2(x) = -0.82(x-2)^3 - 0.86(x-2)^2 + 3.18(x-2) + 3.5, \qquad 2 \le x \le 3.$$

Example 28.14.3 Consider the function

$$f(x) = \begin{cases} -\frac{11}{2}x^3 + 26x^2 - \frac{75}{2}x + 18, & 1 \le x \le 2, \\ \frac{11}{2}x^3 - 40x^2 + \frac{189}{2}x - 70, & 2 \le x \le 3. \end{cases}$$

Show that $f(x)$ is a cubic spline.

Solution. Let

$$p_0(x) = -\frac{11}{2}x^3 + 26x^2 - \frac{75}{2}x + 18, \qquad 1 \le x \le 2,$$
$$\text{and } p_1(x) = \frac{11}{2}x^3 - 40x^2 + \frac{189}{2}x - 70, \qquad 2 \le x \le 3.$$

Here $x_0 = 1, x_1 = 2$ and $x_2 = 3$. The function $f(x)$ will be a cubic spline if

(a) $p_i(x_i) = f(x_i), \qquad p_i(x_{i+1}) = f(x_{i+1}), \qquad i = 0, 1$ and

(b) $p'_{i-1}(x_i) = p'_i(x_i), \qquad p''_{i-1}(x_i) = p''_i(x_i), \qquad i = 1.$

But, here the values of $f(x_0), f(x_1)$ and $f(x_2)$ are not supplied, so only the conditions of (b) are to be checked.

Now,

$$p_0'(x) = -\frac{33}{2}x^2 + 52x - \frac{75}{2}, \qquad p_1'(x) = \frac{33}{2}x^2 - 80x + \frac{189}{2}$$
$$p_0''(x) = -33x + 52, \qquad p_1''(x) = 33x - 80.$$

$p_0'(x_1) = p_0'(2) = 0.5$, $p_1'(x_1) = p_1'(2) = 0.5$, i.e., $p_0'(x_1) = p_1'(x_1)$.

$p_0''(x_1) = p_0''(2) = -14$ and $p_1''(x_1) = p_1''(2) = -14$. Thus $p_0''(x_1) = p_1''(x_1)$.

Hence $f(x)$ is a spline.

28.15 Worked out Examples

Example 28.15.1 Use Stirling's formula to find u_{32} from the following table

$$u_{20} = 14.035, \quad u_{25} = 13.674, \quad u_{30} = 13.257,$$
$$u_{35} = 12.734, \quad u_{40} = 12.089, \quad u_{45} = 11.309.$$

Solution. The finite difference table is shown below.

i	x_i	u_{x_i}	Δu_{x_i}	$\Delta^2 u_{x_i}$	$\Delta^3 u_{x_i}$
−2	20	14.035			
			−0.361		
−1	25	13.674		−0.056	
			−0.417		−0.050
0	30	13.257		−0.106	
			−0.523		−0.016
1	35	12.734		−0.122	
			−0.645		−0.013
2	40	12.089		−0.135	
			−0.780		
3	45	11.309			

Here $x = 32, x_0 = 30, h = 5, s = (x - x_0)/h = 0.4$.

Therefore,

$$u_{32} = y_0 + s\frac{\Delta y_{-1} + \Delta y_0}{2} + \frac{s^2}{2!}\Delta^2 y_{-1} + \frac{s(s^2-1)}{3!}\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2}$$
$$= 13.257 + 0.4\frac{-0.417 - 0.523}{2} + \frac{0.4^2}{2}(-0.106)$$
$$+ \frac{0.4(0.4^2 - 1)}{6}\frac{-0.050 - 0.016}{2}$$
$$= 13.059.$$

Example 28.15.2 The function $y = \sqrt[3]{x}$ is tabulated below.

x :	5600	5700	5800	5900	6000
y :	17.75808	17.86316	17.96702	18.06969	18.17121

Compute $\sqrt[3]{5860}$ by (i) Bessel's formula, and (ii) Stirling's formula.

Solution. The finite difference table is

i	x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
−2	5600	17.75808				
			0.10508			
−1	5700	17.86316		−0.00122		
			0.10386		0.00003	
0	5800	17.96702		−0.00119		0.00001
			0.10267		0.00004	
1	5900	18.06969		−0.00115		
			0.10152			
2	6000	18.17121				

(i) For $x = 5860$, let us take $x_0 = 5800$, then $s = (5860 - 5800)/100 = 0.6$.
By Bessel's formula

$$y(5860) = \frac{y_0 + y_1}{2} + (s - 0.5)\Delta y_0 + \frac{s(s-1)}{2!}\frac{\Delta^2 y_0 + \Delta^2 y_{-1}}{2} + \frac{1}{3!}(s - 0.5)s(s-1)\Delta^3 y_{-1}$$

$$= \frac{17.96702 + 18.06969}{2} + (0.6 - 0.5) \times 0.10267 + \frac{0.6(0.6-1)}{2!}\frac{-0.00115 - 0.00119}{2} + \frac{1}{6}(0.6 - 0.5)(0.6)(0.6-1) \times 0.00004$$

$$= 18.02877.$$

(ii) By Stirling's formula

$$y(5860) = y_0 + s\frac{\Delta y_{-1} + \Delta y_0}{2} + \frac{s^2}{2!}\Delta^2 y_{-1} + \frac{s(s^2-1)}{3!}\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2}$$

$$= 17.96702 + 0.6\frac{0.10386 + 0.10267}{2} + \frac{0.6^2}{2}(-0.00119) + \frac{0.6(0.6^2 - 1)}{6}\frac{0.00003 + 0.00004}{2}$$

$$= 18.02877.$$

Thus $\sqrt[3]{5860} = 18.02877$.

Example 28.15.3 Given $y = f(x)$ in the following table,

x	:	10	15	17
y	:	3	7	11

Find the values of x when $y = 10$ and $y = 5$.

Solution. Here, inverse Lagrange's interpolation formula is used in the following form

$$\phi(y) = \sum_{i=0}^{n} L_i(y) x_i.$$

The Lagrangian functions

$$L_0(y) = \frac{(y-y_1)(y-y_2)}{(y_0-y_1)(y_0-y_2)} = \frac{(y-7)(y-11)}{(3-7)(3-11)} = \frac{y^2-18y+77}{32},$$

$$L_1(y) = \frac{(y-y_0)(y-y_2)}{(y_1-y_0)(y_1-y_2)} = \frac{(y-3)(y-11)}{(7-3)(7-11)} = \frac{y^2-14y+33}{-16},$$

$$\text{and } L_2(y) = \frac{(y-y_0)(y-y_1)}{(y_2-y_0)(y_2-y_1)} = \frac{(y-3)(y-7)}{(11-3)(11-7)} = \frac{y^2-10y+21}{32}.$$

Then

$$\phi(y) = \frac{y^2-18y+77}{32} \times 10 - \frac{y^2-14y+33}{16} \times 15 + \frac{y^2-10y+21}{32} \times 17$$
$$= \frac{1}{32}(137 + 70y - 3y^2).$$

Hence,

$$x(10) \simeq \phi(10) = \frac{1}{32}(137 + 700 - 300) = 16.78125$$
$$\text{and } x(5) \simeq \phi(5) = \frac{1}{32}(137 + 350 - 75) = 12.87500.$$

Example 28.15.4 Use inverse Lagrange's interpolation to find a root of the equation $y \equiv x^3 - 3x + 1 = 0$.

Solution. Here $y(0) = 1 > 0$ and $y(1) = -1 < 0$. One root lies between 0 and 1.

Now, x and y are tabulated, considering five points of x as $0, 0.25, 0.50, 0.75$ and 1.

x	:	0	0.25	0.50	0.75	1.00
y	:	1	0.26563	−0.37500	−0.82813	−1.0000

Solution. Here $y = 0$. Then

$$L_0(y) = \frac{(y-y_1)(y-y_2)(y-y_3)(y-y_4)}{(y_0-y_1)(y_0-y_2)(y_0-y_3)(y_0-y_4)}$$
$$= \frac{y_1y_2y_3y_4}{(y_0-y_1)(y_0-y_2)(y_0-y_3)(y_0-y_4)} = \frac{-0.08249}{3.69194} = -0.02234.$$
$$L_1(y) = \frac{y_0y_2y_3y_4}{(y_1-y_0)(y_1-y_2)(y_1-y_3)(y_1-y_4)} = \frac{-0.31054}{-0.65125} = 0.47684.$$
$$L_2(y) = \frac{y_0y_1y_3y_4}{(y_2-y_0)(y_2-y_1)(y_2-y_3)(y_2-y_4)} = \frac{0.21997}{0.24947} = 0.88176.$$
$$L_3(y) = \frac{y_0y_1y_2y_4}{(y_3-y_0)(y_3-y_1)(y_3-y_2)(y_3-y_4)} = \frac{0.09961}{-0.15577} = -0.63966.$$
$$L_4(y) = \frac{y_0y_1y_2y_3}{(y_4-y_0)(y_4-y_1)(y_4-y_2)(y_4-y_3)} = \frac{0.08249}{0.27190} = 0.30338.$$

Therefore,

$$x \simeq \phi(0) = \sum_{i=0}^{4} L_i(y)x_i = -0.02234 \times 0 + 0.47684 \times 0.25$$
$$+0.88176 \times 0.50 + 0.06014 \times 0.75 + 0.30338 \times 1$$
$$= 0.38373.$$

Hence, the approximate root is 0.38373.

28.16 Module Summary

In this module, some important operators used in numerical method are introduced and finds several number of relations among them. The central difference interpolation formulae – Gauss forward and backward are derived. Based on these formulae Stirling and Bessel formulae are derived. Another interpolation formula called Everett's formula is presented here. The inverse and cubic spline interpolation methods are also discussed in this modulo.

28.17 Self Assessment Questions

1. Define the operators: forward difference (Δ), backward difference (∇), shift (E), central difference (δ) and average (μ).

2. Prove the following relations among the operators
   (i) $1 + \delta^2\mu^2 \equiv \left(1 + \frac{\delta^2}{2}\right)^2$, (ii) $E^{1/2} \equiv \mu + \frac{\delta}{2}$
   (iii) $\mu\delta \equiv \frac{\Delta + \nabla}{2}$, (iv) $\mu\delta \equiv \frac{\Delta E^{-1}}{2} + \frac{\Delta}{2}$,
   (v) $\Delta \equiv \frac{\delta^2}{2} + \delta\sqrt{1 + (\delta^2/4)}$, (vi) $hD \equiv \sinh^{-1}(\mu\delta)$,

(vii) $hD \equiv \log(1+\Delta) \equiv -\log(1-\nabla) \equiv \sinh^{-1}(\mu\delta)$, (viii) $E \equiv e^{hD}$,
(ix) $\mu \equiv \left(1+\frac{\Delta}{2}\right)(1+\Delta)^{-1/2}$, (x) $\mu \equiv \cosh(hD/2)$,
(xi) $\nabla \equiv -\frac{\delta^2}{2} + \delta\sqrt{1+\frac{\delta^2}{4}}$, (xii) $E \equiv 1 + \frac{\delta^2}{2} + \delta\sqrt{1+\frac{\delta^2}{4}}$,
(xiii) $E\nabla \equiv \Delta \equiv \delta E^{1/2}$, (xiv) $\nabla \equiv E^{-1}\Delta$.

3. Prove that
(i) $e^x = \left(\frac{\Delta^2}{E}\right)e^x \frac{Ee^x}{\Delta^2 e^x}$
(ii) $\Delta \log f(x) = \log\left[1 + \frac{\Delta f(x)}{f(x)}\right]$
(iii) $\left(\frac{\Delta^2}{E}\right)x^3 = 6x$.

4. Show that the operators δ, μ, E, Δ and ∇ are commute with one another.

5. Compute the missing term in the following table.

x	:	0	2	4	6	8
$f(x)$	:	12	6	0	?	−25

6. Using Gauss's forward formula, find the value of $f(32)$ given that
$f(25) = 0.2707, f(30) = 0.3027, f(35) = 0.3386, f(40) = 0.3794$.

7. Using Gauss's backward formula, find the value of $\sqrt{518}$ given that
$\sqrt{500} = 22.360680$, $\sqrt{510} = 22.583100$,
$\sqrt{520} = 22.803509$, $\sqrt{530} = 23.021729$.

8. Use a suitable central difference formula of either Stirling's or Bessel's to find the values of $f(x)$ from the following tabulated function at $x = 1.35$ and at $x = 1.42$.

x	:	1.0	1.2	1.4	1.6	1.8
$f(x)$	:	1.17520	1.50946	1.90430	2.37557	2.94217

9. From Bessel's formula, derive the following formula for midway interpolation

$$y_{1/2} = \frac{1}{2}(y_0 + y_1) - \frac{1}{16}(\Delta^2 y_{-1} + \Delta^2 y_0) + \frac{3}{256}(\Delta^4 y_{-2} + \Delta^4 y_{-1}) - \cdots$$

Also deduce this formula from Everett's formula.

10. Using Everett's formula, evaluate $f(20)$ from the following table.

x	:	14	18	22	26
$f(x)$	:	2877	3162	3566	3990

11. Use the technique of inverse interpolation to find x for which $\sinh x = 5.5$ from the following table.

x	:	2.2	2.4	2.6	2.8
$\sinh x$	:	4.457	5.466	6.095	8.198

12. Given the following table of $f(x)$ between $x = 1.1$ and $x = 1.5$, find the zero of $f(x)$.

x	:	1.1	1.2	1.3	1.4	1.5
$f(x)$	:	1.769	1.472	1.103	−1.344	−1.875

13. Use the technique of inverse interpolation to find a real root of the equation $x^3 - 2x - 4 = 0$.

14. The following values of x and y are calculated from the relation $y = x^3 + 10$

x	:	1	2	3	4	5
y	:	11	18	37	74	135

Determine the cubic spline $p(x)$ for the interval [2, 3] given that
(a) $p'(1) = y'(1)$ and $p'(5) = y'(5)$, (b) $p''(1) = y''(1)$ and $p''(5) = y''(5)$.

15. Fit a cubic spline to the function defined by the set of points given in the following table.

x	:	0.10	0.15	0.20	0.25	0.30
$y = e^x$	:	1.1052	1.1618	1.2214	1.2840	1.3499

Use the end conditions

(a) $M_0 = M_N = 0$
(b) $p'(0.10) = y'(0.10)$ and $p'(0.30) = y'(0.30)$ and
(c) $p''(0.10) = y''(0.10)$ and $p''(0.30) = y''(0.30)$.

Interpolate in each case for $x = 0.12$ and state which of the end conditions gives the best fit.

16. The distance d_i that a car has travelled at time t_i is given below.

time t_i	:	0	2	4	6	8
distance d_i	:	0	40	160	300	480

Use the values $p'(0)$ and $p'(8) = 98$, and find the clamped spline for the points.

17. Fit a cubic spline for the points (0,1), (1,0), (2,0), (3,1), (4,2), (5,2) and (6,1) and $p'(0) = -0.6, p'(6) = -1.8$ and $p''(0) = 1$ and $p''(6) = -1$.

18. A function $f(x)$ is defined as follows:

$$f(x) = \begin{cases} 1+x, & 0 \le x \le 3 \\ 1+x+(x-3)^2, & 3 \le x \le 4. \end{cases}$$

Show that $f(x)$ is a cubic spline in [0, 4].

28.18 References

1. M.Pal, Numerical Analysis for Scientists and Engineers: Theory and C Programs, Narosa, 2007.

2. M.K. Jain, S.R.K. Iyengar and R.K. Jain, Numerical Methods for Scientific and Engineering Computation, New Age International (P) Limited, New Delhi, 1984.

3. J.H. Mathews, Numerical Methods for Mathematics, Science, and Engineering, 2nd ed., Prentice-Hall, Inc., N.J., U.S.A., 1992.

4. S.S.Sastry, Introductory Method of Numerical Analysis, PHI, 2006.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III Group-B

Module No.- 29

Numerical Analysis

(Integration and Least Square Approximation)

Module Structure:

29.1 Introduction

In this module you will learn about differentiation and integration by numerical method. Also, you will learn about approximation of a function by least square method.

You have learnt few methods to find derivative and integration of a function by numerical method in undergraduate class. Here, you will learn some more efficient methods for the same types of problems.

Approximation of a function to a polynomial of any other types of function is very important problem in applied mathematics. We shall discuss about approximation of a function by least square method in terms of orthogonal polynomials.

29.2 Objectives

Gone through this module the students will learn the following:

- Numerical differentiation
- Two-point, three-point formulae
- Simpson 3/8 rule
- Romberge integration
- Gauss-Legendre quadrature
- Gauss-Chebyshev quadrature
- Orthogonal polynomial
- Chebyshev polynomial
- Properties of Chebyshev polynomial
- Least square method
- Function approximation
- Economization of power series.

29.3 Keywords

Numerical differentiation, numerical integration, approximation, least square method, orthogonal functions.

29.4 Differentiation

Numerical differentiation is a method to find the derivatives of a function at some values of independent variable x, when the function $f(x)$ is not known explicitly, but is known only for a set of arguments.

Like interpolation, a number of formulae for differentiation are derived. The choice of formula depends on the point at which the derivative is to be determined. So, to find the derivative at a point at the beginning of the table, the formula based on Newton's forward interpolation is used, but, at a point which is at the end of the table, the formula based on Newton's backward interpolation is used. If the given values of x_i are not equispaced then the formula based on Lagrange's interpolation is appropriate.

29.4.1 Error in Numerical Differentiation

The error in polynomial interpolation is

$$E(x) = (x - x_0)(x - x_1)\cdots(x - x_n)\frac{f^{n+1}(\xi)}{(n+1)!} = w(x)\frac{f^{n+1}(\xi)}{(n+1)!}$$

where $\min\{x, x_0, \ldots, x_n\} < \xi < \max\{x, x_0, \ldots, x_n\}$ and $w(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$. Obviously, $\xi = \xi(x)$ is an unknown function of x.

Therefore,

$$E'(x) = w'(x)\frac{f^{n+1}(\xi)}{(n+1)!} + w(x)\frac{f^{n+2}(\xi)}{(n+1)!}\xi'(x). \tag{29.1}$$

The bound of the second term is unknown due to the presence of the unknown quantity $\xi'(x)$.

But, at $x = x_i$, $w(x) = 0$. Thus,

$$E'(x_i) = w'(x_i)\frac{f^{n+1}(\xi_i)}{(n+1)!}, \tag{29.2}$$

where $\min\{x, x_0, \ldots, x_n\} < \xi_i < \max\{x, x_0, \ldots, x_n\}$. The error can also be expressed in terms of divided difference.

Let $E(x) = w(x)f[x, x_0, x_1, \ldots, x_n]$ where $f[x, x_0, x_1, \ldots, x_n] = \dfrac{f^{n+1}(\xi)}{(n+1)!}$.

Then $E'(x) = w'(x)f[x, x_0, x_1, \ldots, x_n] + w(x)f[x, x, x_0, x_1, \ldots, x_n]$.

Now, this expression is differentiated $(k-1)$ times by Leibnitz's theorem.

$$\begin{aligned} E^k(x) &= \sum_{i=0}^{k} {}^kC_r\, w^{(i)}(x) \frac{d^{k-1}}{dx^{k-i}}(f[x, x_0, \ldots, x_n]) \\ &= \sum_{i=0}^{k} {}^kC_r\, w^{(i)}(x)(k-i)! f[\overbrace{x, x, \ldots, x}^{k-i+1}, x_0, \ldots, x_n] \\ &= \sum_{i=0}^{k} \frac{k!}{i!} w^{(i)}(x)(k-i)! f[\overbrace{x, x, \ldots, x}^{k-i+1}, x_0, \ldots, x_n], \end{aligned} \tag{29.3}$$

where $w^{(i)}(x)$ denotes the ith derivative of $w(x)$.

Note 29.4.1 If a function $f(x)$ is well-approximated by a polynomial $\phi(x)$ of degree at most n, the slope $f'(x)$ can also be approximated by the slope $\phi'(x)$. But, the error committed in $\phi'(x)$ is more than the error committed in $\phi(x)$.

29.5 Differentiation Based on Newton's Forward Interpolation Polynomial

Suppose the function $y = f(x)$ is known at $(n+1)$ equispaced points $x_0, x_1, \ldots, x_n$ and they are $y_0, y_1, \ldots, y_n$ respectively, i.e., $y_i = f(x_i), i = 0, 1, \ldots, n$. Let $x_i = x_0 + ih$ and $u = \dfrac{x - x_0}{h}$, h is the spacing.

The Newton's forward interpolation formula is

$$\begin{aligned} \phi(x) &= y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \cdots + \frac{u(u-1)\cdots(u-\overline{n-1})}{n!}\Delta^n y_0 \\ &= y_0 + u\Delta y_0 + \frac{u^2-u}{2!}\Delta^2 y_0 + \frac{u^3-3u^2+2u}{3!}\Delta^3 y_0 + \frac{u^4-6u^3+11u^2-6u}{4!}\Delta^4 y_0 \\ &\quad + \frac{u^5-10u^4+35u^3-50u^2+24u}{5!}\Delta^5 y_0 + \cdots \end{aligned} \tag{29.4}$$

with error

$$E(x) = \frac{u(u-1)\cdots(u-n)}{(n+1)!} h^{n+1} f^{(n+1)}(\xi),$$

where $\min\{x, x_0, \cdots, x_n\} < \xi < \max\{x, x_0, \ldots, x_n\}$.

Differentiating (29.4) successively with respect to x, we obtain

$$\begin{aligned} \phi'(x) = \frac{1}{h}\Bigg[&\Delta y_0 + \frac{2u-1}{2!}\Delta^2 y_0 + \frac{3u^2-6u+2}{3!}\Delta^3 y_0 + \frac{4u^3-18u^2+22u-6}{4!}\Delta^4 y_0 \\ &+ \frac{5u^4-40u^3+105u^2-100u+24}{5!}\Delta^5 y_0 + \cdots \Bigg] \end{aligned} \tag{29.5}$$

$\left(\text{as } \frac{du}{dx} = \frac{1}{h}\right)$

$$\phi''(x) = \frac{1}{h^2}\left[\Delta^2 y_0 + \frac{6u-6}{3!}\Delta^3 y_0 + \frac{12u^2-36u+22}{4!}\Delta^4 y_0 + \frac{20u^3-120u^2+210u-100}{5!}\Delta^5 y_0 + \cdots\right] \tag{29.6}$$

$$\phi'''(x) = \frac{1}{h^3}\left[\Delta^3 y_0 + \frac{24u-36}{4!}\Delta^4 y_0 + \frac{60u^2-240u+210}{5!}\Delta^5 y_0 + \cdots\right] \tag{29.7}$$

and so on.

It may be noted that Δy_0, $\Delta^2 y_0$, $\Delta^3 y_0, \cdots$ are constants.

The above equations give the approximate derivative of $f(x)$ at arbitrary point x $(= x_0 + uh)$.

When $x = x_0$, $u = 0$, the above formulae become

$$\phi'(x_0) = \frac{1}{h}\left[\Delta y_0 - \frac{1}{2}\Delta^2 y_0 + \frac{1}{3}\Delta^3 y_0 - \frac{1}{4}\Delta^4 y_0 + \frac{1}{5}\Delta^5 y_0 - \cdots\right] \tag{29.8}$$

$$\phi''(x_0) = \frac{1}{h^2}\left[\Delta^2 y_0 - \Delta^3 y_0 + \frac{11}{12}\Delta^4 y_0 - \frac{5}{6}\Delta^5 y_0 + \cdots\right] \tag{29.9}$$

$$\phi'''(x_0) = \frac{1}{h^3}\left[\Delta^3 y_0 - \frac{3}{2}\Delta^4 y_0 + \frac{7}{4}\Delta^5 y_0 - \cdots\right] \tag{29.10}$$

and so on.

Error in differentiation formula based on Newton's forward interpolation polynomial

The error in Newton's forward interpolation formula is

$$E(x) = u(u-1)\cdots(u-n)h^{n+1}\frac{f^{n+1}(\xi)}{(n+1)!}.$$

Then

$$E'(x) = h^{n+1}\frac{f^{n+1}(\xi)}{(n+1)!}\frac{d}{du}[u(u-1)\cdots(u-n)]\frac{1}{h} + \frac{u(u-1)\cdots(u-n)}{(n+1)!}h^{n+1}\frac{d}{dx}[f^{n+1}(\xi)]$$

$$= h^n\frac{f^{n+1}(\xi)}{(n+1)!}\frac{d}{du}[u(u-1)\cdots(u-n)] + \frac{u(u-1)\cdots(u-n)}{(n+1)!}h^{n+1}f^{n+2}(\xi_1), \tag{29.11}$$

where $\min\{x, x_0, x_1, \ldots, x_n\} < \xi, \xi_1 < \max\{x, x_0, x_1, \ldots, x_n\}$.

Error at the point $x = x_0$, i.e., $u = 0$ is

$$E'(x_0) = h^n\frac{f^{n+1}(\xi)}{(n+1)!}\frac{d}{du}[u(u-1)\cdots(u-n)]_{u=0} = \frac{h^n(-1)^n\, n!\, f^{n+1}(\xi)}{(n+1)!}$$

$$\left[\text{as } \frac{d}{du}[u(u-1)\cdots(u-n)]_{u=0} = (-1)^n n!\right]$$

$$= \frac{(-1)^n h^n f^{n+1}(\xi)}{n+1}, \tag{29.12}$$

where $\min\{x, x_0, x_1, \ldots, x_n\} < \xi < \max\{x, x_0, x_1, \ldots, x_n\}$.

Example 29.5.1 From the following table find the value of $\frac{dy}{dx}$ and $\frac{d^2y}{dx^2}$ at the point $x = 1.5$.

x :	1.5	2.0	2.5	3.0	3.5	4.0
y :	3.375	7.000	13.625	24.000	38.875	59.000

Solution. The forward difference table is

x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
1.5	3.375			
		3.625		
2.0	7.000		3.000	
		6.625		0.750
2.5	13.625		3.750	
		10.375		0.750
3.0	24.000		4.500	
		14.875		0.750
3.5	38.875		5.250	
		20.125		
4.0	59.000			

Here $x_0 = 1.5$ and $h = 0.5$. Then $u = 0$ and hence

$$y'(1.5) = \frac{1}{h}\left(\Delta y_0 - \frac{1}{2}\Delta^2 y_0 + \frac{1}{3}\Delta^3 y_0 - \cdots\right)$$
$$= \frac{1}{0.5}\left(3.625 - \frac{1}{2} \times 3.000 + \frac{1}{3} \times 0.750\right) = 4.750.$$
$$y''(1.5) = \frac{1}{h^2}(\Delta^2 y_0 - \Delta^3 y_0 + \cdots) = \frac{1}{(0.5)^2}(3.000 - 0.750) = 9.000.$$

Table of derivatives

The summary of the formulae of derivatives based on finite differences.

$$f'(x_0) \simeq \frac{1}{h}\left(\Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 - \frac{1}{4}\Delta^4 + \frac{1}{5}\Delta^5 - \frac{1}{6}\Delta^6 + \cdots\right)y_0 \tag{29.13}$$

$$f''(x_0) \simeq \frac{1}{h^2}\left(\Delta^2 - \Delta^3 + \frac{11}{12}\Delta^4 - \frac{5}{6}\Delta^5 + \frac{137}{180}\Delta^6 - \frac{7}{10}\Delta^7 + \frac{363}{560}\Delta^8 + \cdots\right)y_0 \tag{29.14}$$

$$f'(x_n) \simeq \frac{1}{h}\left(\nabla + \frac{1}{2}\nabla^2 + \frac{1}{3}\nabla^3 + \frac{1}{4}\nabla^4 + \frac{1}{5}\nabla^5 + \frac{1}{6}\nabla^6 + \cdots\right)y_n \tag{29.15}$$

$$f''(x_n) \simeq \frac{1}{h^2}\left(\nabla^2 + \nabla^3 + \frac{11}{12}\nabla^4 + \frac{5}{6}\nabla^5 + \frac{137}{180}\nabla^6 + \frac{7}{10}\nabla^7 + \frac{363}{560}\nabla^8 + \cdots\right)y_n \tag{29.16}$$

$$f'(x_0) \simeq \frac{1}{h}\left(\frac{\Delta y_{-1}+\Delta y_0}{2} - \frac{\Delta^3 y_{-2}+\Delta^3 y_{-1}}{12} + \frac{\Delta^5 y_{-3}+\Delta^5 y_{-2}}{60} + \cdots\right) y_0 \tag{29.17}$$

$$f''(x_0) \simeq \frac{1}{h^2}\left(\Delta^2 y_{-1} - \frac{1}{12}\Delta^4 y_{-2} + \frac{1}{90}\Delta^6 y_{-3} - \cdots\right) y_0 \tag{29.18}$$

29.6 Two-point and Three-point Formulae

Only the first term of (29.13), gives a simple formula for the first order derivative

$$f'(x_i) \simeq \frac{\Delta y_i}{h} = \frac{y_{i+1} - y_i}{h} = \frac{y(x_i + h) - y(x_i)}{h}. \tag{29.19}$$

Similarly, the equation (29.15), gives

$$f'(x_i) \simeq \frac{\nabla y_i}{h} = \frac{y_i - y_{i-1}}{h} = \frac{y(x_i) - y(x_i - h)}{h}. \tag{29.20}$$

Adding equations (29.19) and (29.20), we obtain the central difference formula for first order derivative, as

$$f'(x_i) \simeq \frac{y(x_i + h) - y(x_i - h)}{2h}. \tag{29.21}$$

Equations (29.19)-(29.21) give two-point formulae to find first order derivative at $x = x_i$.

Similarly, from equation (29.14)

$$f''(x_i) \simeq \frac{\Delta^2 y_i}{h^2} = \frac{y_{i+2} - 2y_{i+1} + y_i}{h^2} = \frac{y(x_i+2h) - 2y(x_i + h)+y(x_i)}{h^2}. \tag{29.22}$$

From equation (29.16)

$$f''(x_i) \simeq \frac{\nabla^2 y_i}{h^2} = \frac{y_i - 2y_{i-1}+y_{i-2}}{h^2} = \frac{y(x_i)-2y(x_i - h)+y(x_i - 2h)}{h^2}. \tag{29.23}$$

Equation (29.18) gives

$$f''(x_0) \simeq \frac{\Delta^2 y_{-1}}{h^2} = \frac{y_1 - 2y_0 + y_{-1}}{h^2} = \frac{y(x_0 + h) - 2y(x_0) + y(x_0 - h)}{h^2}.$$

In general,

$$f''(x_i) \simeq \frac{y(x_i + h) - 2y(x_i) + y(x_i - h)}{h^2}. \tag{29.24}$$

Equations (29.22)-(29.24) give the three-point formulae for second order derivative.

29.7 Integration

It is well known that, if a function $f(x)$ is known completely, even then it is not always possible to evaluate the definite integral of it using analytic method. Again, in many real life problems, we are required to integrate a function between two given limits, but the function is not known explicitly, but, it is known in a tabular form (equally or unequally spaced). Then a method, known as **numerical integration or quadrature** can be used to solve all such problems.

The problem of numerical integration is stated below:

Given a set of data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ *of a function* $y = f(x)$, *it is required to find the value of the definite integral* $\int_a^b f(x)\,dx$. *The function* $f(x)$ *is replaced by a suitable interpolating polynomial* $\phi(x)$.

Then the approximate value of the definite integral is calculated using the following formula

$$\int_a^b f(x)\,dx \simeq \int_a^b \phi(x)\,dx. \tag{29.25}$$

Thus, different integration formulae can be derived depending on the type of the interpolation formulae used.

A numerical integration formula is said to be of **closed type**, if the limits of integration a and b are taken as interpolating points. If a and b are not taken as interpolating points then the formula is known as **open type** formula.

29.8 General Quadrature Formula Based on Newton's Forward Interpolation

The Newton's forward interpolation formula for the equispaced points $x_i, i = 0, 1, \ldots, n$, $x_i = x_0 + ih$ is

$$\phi(x) = y_0 + u\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \cdots, \tag{29.26}$$

where $u = \dfrac{x - x_0}{h}$, h is the spacing.

Let the interval $[a, b]$ be divided into n equal subintervals such that $a = x_0 < x_1 < x_2 < \cdots < x_n = b$. Then

$$\begin{aligned} I = \int_a^b f(x)\,dx &\simeq \int_{x_0}^{x_n} \phi(x)\,dx \\ &= \int_{x_0}^{x_n} \left[y_0 + u\Delta y_0 + \frac{u^2 - u}{2!}\Delta^2 y_0 + \frac{u^3 - 3u^2 + 2u}{3!}\Delta^3 y_0 + \cdots\right] dx. \end{aligned}$$

Since $x = x_0 + uh$, $dx = h\,du$, when $x = x_0$ then $u = 0$ and when $x = x_n$ then $u = n$.

Thus,

$$I = \int_0^n \left[y_0 + u\Delta y_0 + \frac{u^3-u}{2!}\Delta^2 y_0 + \frac{u^3-3u^2+2u}{3!}\Delta^3 y_0 + \cdots\right] hdu$$
$$= h\left[y_0[u]_0^n + \Delta y_0\left[\frac{u^2}{2}\right]_0^n + \frac{\Delta^2 y_0}{2!}\left[\frac{u^3}{3} - \frac{u^2}{2}\right]_0^n + \frac{\Delta^3 y_0}{3!}\left[\frac{u^4}{4} - u^3 + u^2\right]_0^n + \cdots\right]$$
$$= nh\left[y_0 + \frac{n}{2}\Delta y_0 + \frac{2n^2-3n}{12}\Delta^2 y_0 + \frac{n^3-4n^2+4n}{24}\Delta^3 y_0 + \cdots\right]. \quad (29.27)$$

From this formula, one can generate different integration formulae by substituting $n = 1, 2, 3, \ldots$.

You have learn Trapezoidal ($n = 1$), Simpson's 1/3 ($n = 2$) and Weddle's ($n = 6$) rules in B.Sc. to find the integration of a function numerically. Here we shall discuss the case $n = 3$.

29.8.1 Simpson's 3/8 rule

Simpson's 3/8 rule can be obtained by substituting $n = 3$ in (29.27). Note that the differences higher than the third order do not exist here.

$$\int_a^b f(x)dx = \int_{x_0}^{x_3} f(x)dx = 3h\left[y_0 + \frac{3}{2}\Delta y_0 + \frac{3}{4}\Delta^2 y_0 + \frac{1}{8}\Delta^3 y_0\right]$$
$$= 3h\left[y_0 + \frac{3}{2}(y_1 - y_0) + \frac{3}{4}(y_2 - 2y_1 + y_0) + \frac{1}{8}(y_3 - 3y_2 + 3y_1 - y_0)\right]$$
$$= \frac{3h}{8}[y_0 + 3y_1 + 3y_2 + y_3]. \quad (29.28)$$

This formula is known as **Simpson's 3/8 rule.**

Now, the interval $[a, b]$ is divided into n (divisible by 3) equal subintervals by the points $x_0, x_1, \ldots, x_n$ and the formula is applied to each of the intervals.

Then

$$\int_{x_0}^{x_n} f(x)dx = \int_{x_0}^{x_3} f(x)dx + \int_{x_3}^{x_6} f(x)dx + \cdots + \int_{x_{n-3}}^{x_n} f(x)dx$$
$$= \frac{3h}{8}[(y_0 + 3y_1 + 3y_2 + y_3) + (y_3 + 3y_4 + 3y_5 + y_6)$$
$$+ \cdots + (y_{n-3} + 3y_{n-2} + 3y_{n-1} + y_n)]$$
$$= \frac{3h}{8}[y_0 + 3(y_1 + y_2 + y_4 + y_5 + y_7 + y_8 + \cdots + y_{n-2} + y_{n-1})$$
$$+2(y_3 + y_6 + y_9 + \cdots + y_{n-3}) + y_n]. \quad (29.29)$$

This formula is known as **Simpson's 3/8 composite rule.**

Note 29.8.1 This method is not so accurate as Simpson's 1/3 rule. The error in this formula is

$$-\frac{3}{80}h^5 f^{iv}(\xi), x_0 < \xi < x_3.$$

29.9 Romberg's Integration

The value of the integration obtained from trapezoidal or Simpson's rules or other rules may be improved by repeated use of Richardson's extrapolation procedure. This method is known as **Romberg's integration** method as he was the first one to describe the algorithm in recursive form.

We assume that $f \in C^n[a,b]$ for all n, then the error term for the trapezoidal rule can be represented in a series involving only even powers of h, i.e.,

$$I = \int_a^b f(x)dx = I_0(h) + c_1h^2 + c_2h^4 + c_3h^6 + \cdots . \tag{29.30}$$

The Richardson's extrapolation method is used successively to eliminate c_1, c_2, c_3 and so on. This process generates integration formulae whose error terms are of the even orders $O(h^4), O(h^6), O(h^8)$, etc.

The step length h is replaced by $h/2$ in (29.30) then

$$I = I_0(h/2) + c_1\frac{h^2}{4} + c_2\frac{h^4}{16} + c_3\frac{h^6}{64} + \cdots \tag{29.31}$$

To eliminate c_1, equation (29.31) is multiplied by 4 and this equation is subtracted from (29.30). Thus

$$3I = 4I_0(h/2) - I_0(h) - \frac{3}{4}c_2h^4 - \frac{15}{16}c_4h^6 - \cdots$$

This equation is divided by 3 and the coefficients of h^4, h^6 etc. are denoted by d_1, d_2 etc. Thus

$$I = \frac{4I_0(h/2) - I_0(h)}{3} + d_1h^4 + d_2h^6 + \cdots \tag{29.32}$$

Denoting $\dfrac{4I_0(h/2) - I_0(h)}{3}$ by $I_1(h/2)$ and this is the first Romberg's improvement. Thus by this notation the equation (29.32) can be written as

$$I = I_1(h/2) + d_1h^4 + d_2h^6 + \cdots . \tag{29.33}$$

Again, h is replaced by $h/2$ in (29.33), therefore,

$$I = I_1(h/2^2) + d_1\frac{h^4}{16} + d_2\frac{h^6}{64} + \cdots . \tag{29.34}$$

Now, eliminating d_1, we obtain

$$15I = 16I_1(h/2^2) - I_1(h/2) - \frac{3}{4}d_2h^6 + \cdots$$

That is,

$$\begin{aligned} I &= \frac{16I_1(h/2^2) - I_1(h/2)}{15} + e_1h^6 + \cdots \\ &= I_2(h/2^2) + e_1h^6 + \cdots , \end{aligned} \tag{29.35}$$

where

$$I_2(h/2^2) = \frac{16I_1(h/2^2) - I_1(h/2)}{15} = \frac{4^2 I_1(h/2^2) - I_1(h/2)}{4^2 - 1} \tag{29.36}$$

is the second Romberg's improvement.

In general,

$$I_m(h/2^k) = \frac{4^k I_{m-1}(h/2^k) - I_{m-1}(h/2^{k-1})}{4^k - 1}, \tag{29.37}$$

where $m = 1, 2, \ldots; k = m, m+1, \ldots$ and $I_0(h) = I(h)$.

Thus

$$I = I_m(h/2^m) + O(h^{2m+2}). \tag{29.38}$$

Now,

$$\begin{aligned} I_1(h/2) &= \frac{4I(h/2) - I(h)}{3} = \frac{1}{3}\left[4.\frac{h}{4}(y_0 + 2y_1 + y_2) - \frac{h}{2}(y_0 + y_2)\right] \\ &= \frac{h/2}{3}[y_0 + 4y_1 + y_2]. \end{aligned}$$

This is the Simpson's 1/3 rule with step size $h/2$. That is, the first improved value is equal to the value obtained by Simpson's 1/3 rule.

Now,

$$\begin{aligned} I_2(h/2^2) &= \frac{16I_1(h/2^2) - I_1(h/2)}{15} \\ &= \frac{1}{15}\left[16.\frac{h/4}{3}\left\{y_0 + 4(y_1 + y_3) + 2y_2 + y_4\right\} - \frac{h/2}{3}(y_0 + 4y_2 + y_4)\right] \\ &\quad \text{[Simpson's rule for 4 and 2 intervals]} \\ &= \frac{2(h/4)}{45}[7y_0 + 32y_1 + 12y_2 + 32y_3 + 7y_4]. \end{aligned}$$

This is the Boole's rule for step length $h/4$.

The Romberg's integration can be carried out using the triangular array of successive approximations to the integral as shown in Table 29.1.

The entries in the 0th order are computed directly and the other entries are calculated by using the formula (29.37). The values along the diagonal would converge to the integral.

The advantage of Romberg's method is that the method gives much more accurate result than the usual composite trapezoidal rule. A computational weakness of this method is that twice as many function evaluations are needed to decrease the error from $O(h^{2n})$ to $O(h^{2n+2})$. Practically, the computations are carried out rowwise until the desired accuracy is achieved.

Table 29.1: Romberg's integration table.

n	h	0th order trapezoidal	1st order Simpson	2nd order Boole	3rd order	4th order
1	h	$I_0(h)$				
2	$h/2$	$I_0(h/2)$	$I_1(h/2)$			
4	$h/2^2$	$I_0(h/2^2)$	$I_1(h/2^2)$	$I_2(h/2^2)$		
8	$h/2^3$	$I_0(h/2^3)$	$I_1(h/2^3)$	$I_2(h/2^3)$	$I_3(h/2^3)$	
16	$h/2^4$	$I_0(h/2^4)$	$I_1(h/2^4)$	$I_2(h/2^4)$	$I_3(h/2^4)$	$I_4(h/2^4)$

Note 29.9.1 If the 0th order (starting) approximation is calculated using Simpson's rule then first order approximation I_1 gives the Boole's approximation and so on.

Example 29.9.1 Find the value of $\int_0^1 \frac{dx}{1+x^2}$ using Romberg's method starting with trapezoidal rule.

Solution. Let $x_0 = 0, x_1 = 1, h = \frac{1-0}{n}, x_i = x_0 + ih$.

The initial approximations are computed by trapezoidal rule.

$n = 1, h = 1$. Then $I_0 = \frac{h}{2}(y_0 + y_1) = 0.5(1 + 0.5) = 0.75000$.

$n = 2, h = 0.5$. $I_0 = \frac{h}{2}(y_0 + 2y_1 + y_2) = 0.25(1 + 2 \times 0.8 + 0.5) = 0.775$.

$n = 4, h = 0.25$. $I_0 = \frac{h}{2}[y_0 + 2(y_1 + y_2 + y_3) + y_4]$

$= 0.125(1 + 2(0.94118 + 0.8 + 0.64) + 0.5) = 0.7828$.

$n = 8, h = 0.125$. $I_0 = \frac{h}{2}[y_0 + 2(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7) + y_8]$

$= 0.0625[1 + 2(0.98462 + 0.94118 + 0.87671 + 0.8 + 0.7191$

$+0.64 + 0.56637) + 0.5] = 0.78475$.

$$I_1(h/2) = \frac{4I_0(0.5) - I_0(1.0)}{3} = \frac{4 \times 0.77500 - 0.75000}{3} = 0.78333$$

$$I_1(h/2^2) = \frac{4I_0(0.25) - I_0(0.5)}{3} = \frac{4 \times 0.78280 - 0.77500}{3} = 0.78540$$

$$I_1(h/2^3) = \frac{4I_0(0.125) - I_0(0.25)}{3} = \frac{4 \times 0.78475 - 0.78280}{3} = 0.78540.$$

All the calculations are shown in the following table.

n	h	I_0	I_1	I_2	I_3
1	1.000	0.75000			
2	0.500	0.77500	0.78333		
4	0.250	0.78280	0.78540	0.78554	
8	0.125	0.78475	0.78540	0.78540	0.78540

The exact integral is $\pi/4 = 0.78540$. In each column the numbers are converging to the value 0.78540. The values in Simpson's rule column (I_1) converge faster than the values in the trapezoidal rule column (I_0). Ultimately, the value of the integral is 0.78540 correct up to five decimal places.

Methods Based on Undetermined Coefficients

In the Newton-Cotes method all the nodes $x_i, i = 0, 1, 2, \ldots, n$ are known and equispaced. Also, the formulae obtained from Newton-Cotes method are exact for the polynomials of degree up to n. When the nodes $x_i, i = 0, 1, 2, \ldots, n$ are unknown then one can devise some methods which give exact result for the polynomials of degree up to $2n - 1$. These methods are called **Gaussian quadrature methods.**

29.10 Gaussian Quadrature

The Gaussian quadrature is of the form

$$\int_a^b \psi(x) f(x) dx = \sum_{i=1}^{n} w_i f(x_i), \tag{29.39}$$

where x_i and w_i are respectively called nodes and weights and $\psi(x)$ is called the weight function. Depending on the weight function different quadrature formula can be obtained.

The fundamental theorem of Gaussian quadrature states that *the optimal nodes of the m-point Gaussian quadrature formula are precisely the zeros of the orthogonal polynomial for the same interval and weight function.* Gaussian quadrature is optimal because it fits all polynomial up to degree $2m$ exactly.

To determine the weights corresponding to the Gaussian nodes x_i, compute a Lagrange's interpolating polynomial for $f(x)$ by assuming

$$\pi(x) = \prod_{j=1}^{m} (x - x_j). \tag{29.40}$$

Then

$$\pi'(x_j) = \prod_{\substack{i=1 \\ i \neq j}}^{m} (x_j - x_i). \tag{29.41}$$

Then Lagrange's interpolating polynomial through m points is

$$\phi(x) = \sum_{j=1}^{m} \frac{\pi(x)}{(x - x_j)\pi'(x_j)} f(x_j) \tag{29.42}$$

for arbitrary points x. Now, determine a set of points x_j and w_j such that for a weight function $\psi(x)$ the following relation is valid.

$$\begin{aligned} \int_a^b \phi(x)\psi(x)dx &= \int_a^b \sum_{j=1}^{m} \frac{\pi(x)\psi(x)}{(x - x_j)\pi'(x_j)} f(x_j)dx \\ &= \sum_{j=1}^{m} w_j f(x_j), \end{aligned} \tag{29.43}$$

where weights w_j are obtained from

$$w_j = \frac{1}{\pi'(x_j)} \int_a^b \frac{\pi(x)\psi(x)}{x - x_j} dx. \tag{29.44}$$

The weights w_j are sometimes called the **Christofell numbers.**

It can be shown that the error is given by

$$E = \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b \psi(x)[\pi(x)]^2 dx. \tag{29.45}$$

Any finite interval $[a, b]$ can be transferred to the interval $[-1, 1]$ using linear transformation

$$x = \frac{b - a}{2} t + \frac{b + a}{2} = qt + p. \tag{29.46}$$

Then,

$$\int_a^b f(x)\, dx = \int_{-1}^{1} f(qt + p)\, q\, dt. \tag{29.47}$$

Thus to study the Gaussian quadrature, we consider the integral in the form

$$\int_{-1}^{1} \psi(x) f(x) dx = \sum_{i=1}^{n} w_i f(x_i) + E. \tag{29.48}$$

29.10.1 Gauss-Legendre integration methods

Here $\psi(x)$ is taken as 1 and so the formula (29.48) reduces to

$$\int_{-1}^{1} f(x) dx = \sum_{i=1}^{n} w_i f(x_i). \tag{29.49}$$

It may be noted that w_i and x_i are $2n$ parameters and therefore the weights and nodes can be determined such that the formula is exact when $f(x)$ is a polynomial of degree not exceeding $2n-1$.

Let

$$f(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{2n-1}x^{2n-1}. \tag{29.50}$$

Therefore,

$$\begin{aligned}\int_{-1}^{1} f(x)dx &= \int_{-1}^{1} [c_0 + c_1x + c_2x^2 + \cdots + c_{2n-1}x^{2n-1}]dx \\ &= 2c_0 + \frac{2}{3}c_2 + \frac{2}{5}c_4 + \cdots .\end{aligned} \tag{29.51}$$

When $x = x_i$, equation (29.50) becomes

$$f(x_i) = c_0 + c_1x_i + c_2x_i^2 + c_3x_i^3 + \cdots + c_{2n-1}x_i^{2n-1}.$$

Substituting these values to the right hand side of (29.49) to get

$$\begin{aligned}\int_{-1}^{1} f(x)dx &= w_1[c_0 + c_1x_1 + c_2x_1^2 + \cdots + c_{2n-1}x_1^{2n-1}] \\ &\quad + w_2[c_0 + c_1x_2 + c_2x_2^2 + \cdots + c_{2n-1}x_2^{2n-1}] \\ &\quad + w_3[c_0 + c_1x_3 + c_2x_3^2 + \cdots + c_{2n-1}x_3^{2n-1}] \\ &\quad + \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ &\quad + w_n[c_0 + c_1x_n + c_2x_n^2 + \cdots + c_{2n-1}x_n^{2n-1}] \\ &= c_0(w_1 + w_2 + \cdots + w_n) + c_1(w_1x_1 + w_2x_2 + \cdots + w_nx_n) \\ &\quad + c_2(w_1x_1^2 + w_2x_2^2 + \cdots + w_nx_n^2) + \cdots \\ &\quad + c_{2n-1}(w_1x_1^{2n-1} + w_2x_2^{2n-1} + \cdots + w_nx_n^{2n-1}).\end{aligned} \tag{29.52}$$

Since (29.51) and (29.52) are identical, compare the coefficients of c_i, and find $2n$ equations as follows:

$$\begin{aligned} w_1 + w_2 + \cdots + w_n &= 2 \\ w_1x_1 + w_2x_2 + \cdots + w_nx_n &= 0 \\ w_1x_1^2 + w_2x_2^2 + \cdots + w_nx_n^2 &= \tfrac{2}{3} \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &\quad \cdots \\ w_1x_1^{2n-1} + w_2x_2^{2n-1} + \cdots + w_nx_n^{2n-1} &= 0. \end{aligned} \tag{29.53}$$

Now, equation (29.53) is a set of $2n$ non-linear equations consisting $2n$ unknowns w_i and x_i, $i = 1, 2, \ldots, n$. Solution of these equations gives the values of w_i and x_i. Let $w_i = w_i^*$ and $x_i = x_i^*, i = 1, 2, \ldots, n$ be the solution of (29.53). Then the Gauss-Legendre formula is finally given by

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i^* f(x_i^*). \tag{29.54}$$

Unfortunately, determination of general solution of the system (29.53) is very complicated. Thus we concentrate for its particular cases.

Case I. When $n = 1$, the formula is

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1), \text{ where } w_1 = 2 \text{ and } w_1 x_1 = 0, \text{ i.e., } x_1 = 0.$$

Thus for $n = 1$,

$$\int_{-1}^{1} f(x)dx = 2f(0). \tag{29.55}$$

Case II. When $n = 2$ then the integral is

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) + w_2 f(x_2). \tag{29.56}$$

and the system (29.53) reduces to

$$\begin{aligned} w_1 + w_2 &= 2 \\ w_1 x_1 + w_2 x_2 &= 0 \\ w_1 x_1^2 + w_2 x_2^2 &= \tfrac{2}{3} \\ w_1 x_1^3 + w_2 x_2^3 &= 0. \end{aligned} \tag{29.57}$$

The above equations can also be obtained by the following way:
The formula (29.56) is exact when $f(x)$ is a polynomial of degree ≤ 3. Substituting successively $f(x) = 1, x, x^2$ and x^3 in (29.56) and obtain the following system of equations

$$\begin{aligned} w_1 + w_2 &= 2 && (f(x) = 1) \\ w_1 x_1 + w_2 x_2 &= 0 && (f(x) = x) \\ w_1 x_1^2 + w_2 x_2^2 &= \tfrac{2}{3} && (f(x) = x^2) \\ w_1 x_1^3 + w_2 x_2^3 &= 0 && (f(x) = x^3). \end{aligned} \tag{29.58}$$

The solution of these equations is $w_1 = w_2 = 1, x_1 = -1/\sqrt{3}, x_2 = 1/\sqrt{3}$. Hence, the equation (29.56) becomes

$$\int_{-1}^{1} f(x)dx = f(-1/\sqrt{3}) + f(1/\sqrt{3}).$$

Case III. When $n = 3$ then the integral becomes

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3). \tag{29.59}$$

The six unknowns x_1, x_2, x_3 and w_1, w_2, w_3 are related as

$$\begin{aligned} w_1 + w_2 + w_3 &= 2 \\ w_1x_1 + w_2x_2 + w_3x_3 &= 0 \\ w_1x_1^2 + w_2x_2^2 + w_3x_2^3 &= \tfrac{2}{3} \\ w_1x_1^3 + w_2x_2^3 + w_3x_2^3 &= 0 \\ w_1x_1^4 + w_2x_2^4 + w_3x_2^4 &= \tfrac{2}{5} \\ w_1x_1^5 + w_2x_2^5 + w_3x_2^5 &= 0. \end{aligned}$$

These equations may also be obtained by substituting $f(x) = 1, x, x^2, x^3, x^4, x^5$ to the equation (29.59). Solution of this system of equations is

$$x_1 = -\sqrt{3/5}, x_2 = 0, x_3 = \sqrt{3/5}, w_1 = 5/9, w_2 = 8/9, w_3 = 5/9.$$

For these values, equation (29.59) becomes

$$\int_{-1}^{1} f(x)dx = \frac{1}{9}[5f(-\sqrt{3/5}) + 8f(0) + 5f(\sqrt{3/5})]. \tag{29.60}$$

In this way one can determine the formulae for higher values of n. The solution of the equations (29.53) for higher values of n is very complicated as they are non-linear with respect to the nodes $x_1, x_2, \ldots, x_n$. But, they are linear with respect to weights.

If can be shown that $x_i, i = 1, 2, \ldots, n$ are the zeros of the nth degree Legendre's polynomial

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[(x^2 - 1)^n],$$

which can be generated from the recurrence relation

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x) \tag{29.61}$$

where $P_0(x) = 1$ and $P_1(x) = x$. Some lower order Legendre polynomials are

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \tfrac{1}{2}(3x^2 - 1) \\ P_3(x) &= \tfrac{1}{2}(5x^3 - 3x) \\ P_4(x) &= \tfrac{1}{8}(35x^4 - 30x^2 + 3). \end{aligned} \tag{29.62}$$

The weights w_i are given by

$$w_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \left(\frac{x - x_i}{x_i - x_j}\right) dx \tag{29.63}$$

where x_i are known nodes.

Also, the weights w_i may be obtained from the relation

$$w_i = \frac{2}{(1-x_i^2)[P_n'(x_i)]^2}. \tag{29.64}$$

It can be shown that the error of this formula is

$$E = \frac{2^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi), -1 < \xi < 1. \tag{29.65}$$

The nodes and weights for different values of n are listed in Table 29.2.

Table 29.2: Values of x_i and w_i for Gauss-Legendre quadrature.

n	node x_i	weight w_i	order of truncation error
2	±0.57735027	1.00000000	$f^{(4)}(\xi)$
3	0.00000000	0.88888889	$f^{(6)}(\xi)$
	±0.77459667	0.55555556	
4	±0.33998104	0.65214515	$f^{(8)}(\xi)$
	±0.86113631	0.34785485	
5	0.00000000	0.56888889	
	±0.53846931	0.47862867	$f^{(10)}(\xi)$
	±0.90617985	0.23692689	
6	±0.23861919	0.46791393	
	±0.66120939	0.36076157	$f^{(12)}(\xi)$
	±0.93246951	0.17132449	

Example 29.10.1 Find the value of $\int_0^1 \frac{1}{1+x^2}dx$ by Gauss's formula for $n = 2, 4, 6$. Also, calculate the absolute errors.

Solution. To apply the Gauss's formula, the limits are transferred to $-1, 1$ by substituting

$$x = \frac{1}{2}u(1-0) + \frac{1}{2}(1+0) = \frac{1}{2}(u+1).$$

Then $I = \int_0^1 \frac{1}{1+x^2}dx = \int_{-1}^1 \frac{2du}{(u+1)^2+4} = 2\sum_{i=1}^{n} w_i f(u_i)$,

where $f(x_i) = \frac{1}{(x_i+1)^2+4}$.

For the two-point formula ($n = 2$)

$x_1 = -0.57735027, x_2 = 0.57735027, w_1 = w_2 = 1.$

Then $I = 2[1 \times 0.23931272 + 1 \times 0.15412990] = 0.78688524.$

For the four-point formula ($n = 4$)

$x_1 = -0.33998104, x_2 = -0.86113631, x_3 = -x_1, x_4 = -x_2,$
$w_1 = w_3 = 0.65214515, w_2 = w_4 = 0.34785485.$

Then $I = 2[w_1 f(x_1) + w_3 f(x_3) + w_2 f(x_2) + w_4 f(x_4)]$
$= 2[w_1\{f(x_1) + f(-x_1)\} + w_2\{f(x_2) + f(-x_2)\}]$
$= 2[0.65214515 \times (0.22544737 + 0.17254620) + 0.34785485 \times (0.24880059 + 0.13397950)]$
$= 0.78540297.$

For the six-point formula ($n = 6$)

$x_1 = -0.23861919, x_2 = -0.66120939, x_3 = -0.93246951, x_4 = -x_1,$
$x_5 = -x_2, x_6 = -x_3, w_1 = w_4 = 0.46791393, w_2 = w_5 = 0.36076157,$
$w_3 = w_6 = 0.17132449.$

Then $I = 2[w_1\{f(x_1) + f(-x_1)\} + w_2\{f(x_2) + f(-x_2) + w_3\{f(x_3) + f(-x_3)\}]$
$= 2[0.46791393 \times (0.21835488 + 0.18069532) + 0.36076157 \times (0.24302641 + 0.14793738)$
$+ 0.17132449 \times (0.24971530 + 0.12929187)]$
$= 0.78539814.$

The exact value is $\pi/4 = 0.78539816$.

The following table gives a comparison among the different Gauss's formulae.

n	Exact value	Gauss formula	Error
2	0.78539816	0.78688524	1.49×10^{-3}
4	0.78539816	0.78540297	4.81×10^{-6}
6	0.78539816	0.78539814	2.00×10^{-8}

29.10.2 Gauss-Chebyshev integration methods

Gauss-Chebyshev quadrature is also known as **Chebyshev quadrature**. Its weight function is taken as $\psi(x) = (1 - x^2)^{-1/2}$. The general form of this method is

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx = \sum_{i=1}^{n} w_i f(x_i). \tag{29.66}$$

These methods give exact result for polynomials of degree up to $2n - 1$. The nodes $x_i, i = 1, 2, \ldots, n$

are the zeros of the Chebyshev polynomials

$$T_n(x) = \cos(n \cos^{-1} x). \tag{29.67}$$

That is,

$$x_i = \cos\left(\frac{(2i-1)\pi}{2n}\right), i = 1, 2, \ldots, n. \tag{29.68}$$

For $n = 3$ the equation (29.66) becomes

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx = w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3). \tag{29.69}$$

Since the method gives exact value for the polynomials of degree up to $2n-1$. i.e., up to 5. Therefore, for $f(x) = 1, x, x^2, x^3, x^4, x^5$ the following equations are obtained from (29.69).

$$\begin{aligned}
w_1 + w_2 + w_3 &= \pi \\
w_1 x_1 + w_2 x_2 + w_3 x_3 &= 0 \\
w_1 x_1^2 + w_2 x_2^2 + w_3 x_3^2 &= \tfrac{\pi}{2} \\
w_1 x_1^3 + w_2 x_2^3 + w_3 x_3^3 &= 0 \\
w_1 x_1^4 + w_2 x_2^4 + w_3 x_3^4 &= \tfrac{3\pi}{8} \\
w_1 x_1^5 + w_2 x_2^5 + w_3 x_3^5 &= 0.
\end{aligned}$$

The nodes $x_i, i = 1, 2, 3$ can easily be obtained from the relation

$$x_i = \cos(2i-1)\frac{\pi}{6}, \qquad i = 1, 2, 3.$$

That is, $x_1 = \sqrt{3}/2, x_2 = 0, x_3 = -\sqrt{3}/2$. Then the solution of the above equations is $w_1 = w_2 = w_3 = \pi/3$.

Thus formula (29.69) finally becomes

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx = \frac{\pi}{3}\left[f(\sqrt{3}/2) + f(0) + f(-\sqrt{3}/2)\right]. \tag{29.70}$$

In general, the nodes are given by

$$x_i = \cos\left(\frac{(2i-1)\pi}{2n}\right), \qquad i = 1, 2, \ldots, n$$

and the weights are

$$w_i = -\frac{\pi}{T_{n+1}(x_i)T_n'(x_i)} = \frac{\pi}{n}, \qquad i = 1, 2, \ldots, n. \tag{29.71}$$

The error term is

$$E = \frac{2\pi}{2^{2n}(2n)!} f^{(2n)}(\xi), \qquad -1 < \xi < 1. \tag{29.72}$$

Table 29.3: Nodes and weights for Gauss-Chebyshev quadrature.

n	node x_i	weight w_i	order of truncation error
2	± 0.7071068	1.5707963	$f^{(4)}(\xi)$
3	0.0000000	1.0471976	
	± 0.8660254	1.0471976	$f^{(6)}(\xi)$
4	± 0.3826834	0.7853982	
	± 0.9238795	0.7853982	$f^{(8)}(\xi)$
5	0.0000000	0.6283185	
	± 0.5877853	0.6283185	$f^{(10)}(\xi)$
	± 0.9510565	0.6283185	

The explicit formula is then

$$\int_{-1}^{1} \frac{f(x)dx}{\sqrt{1-x^2}} = \frac{\pi}{n}\sum_{i=1}^{n} f\left[\cos\left\{\frac{(2i-1)}{2n}\pi\right\}\right] + \frac{2n}{2^{2n}(2n)!}f^{(2n)}(\xi). \tag{29.73}$$

Table 29.3 gives the values for the first few points and weights for Gauss-Chebyshev quadrature.

Example 29.10.2 Find the value of $\int_0^1 \frac{1}{1+x^2}\,dx$ using Gauss-Chebyshev four-point formula.

Solution. Let $f(x) = \frac{\sqrt{1-x^2}}{1+x^2}$. Here $x_1 = -0.3826834 = -x_2$, $x_3 = -0.9238795 = -x_4$ and $w_1 = w_2 = w_3 = w_4 = 0.7853982$.

Then

$$\begin{aligned} I &= \int_0^1 \frac{1}{1+x^2}dx = \frac{1}{2}\int_{-1}^{1} \frac{1}{1+x^2}dx = \frac{1}{2}\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}dx \\ &= \frac{1}{2}[w_1 f(x_1) + w_2 f(x_2) + w_3 f(x_3) + w_4 f(x_4)] \\ &= \frac{1}{2} \times 0.7853982[f(x_1) + f(x_2) + f(x_3) + f(x_4)] \\ &= \frac{1}{2} \times 0.7853982[2 \times 0.8058636 + 2 \times 0.2064594] = 0.7950767, \end{aligned}$$

while the exact value is $\pi/4 = 0.7853982$. The absolute error is 0.0096785.

Remark 29.10.1 *It may be noted that the Gauss-Legendre four-point formula gives better result for this problem.*

29.11 Worked out Examples

Example 29.11.1 If $f(x)$ is a polynomial of degree 2, prove that

$$\int_0^1 f(x)dx = \frac{1}{12}[5f(0) + 8f(1) - f(2)].$$

Solution. Let the formula be
$\int_0^1 f(x)dx = w_1 f(0) + w_2 f(1) + w_3 f(2)$. The formula is exact for $f(x) = 1, x, x^2$.
Substituting $f(x) = 1, x, x^2$ successively to the above formula and find the following set of equations.

$$1 = w_1 + w_2 + w_3,\ 1/2 = w_2 + 2w_3,\ 1/3 = w_2 + 4w_3.$$

Solution of these equations is $w_1 = \frac{5}{12}, w_2 = \frac{2}{3}, w_3 = -\frac{1}{12}$.
Hence the formula becomes $\int_0^1 f(x)dx = \frac{1}{12}[5f(0) + 8f(1) - f(2)]$.

Example 29.11.2 Deduce the following quadrature formula

$$\int_{-1}^1 f(x)dx = \frac{2}{3}[f(0) + f(1/\sqrt{2}) + f(-1/\sqrt{2})].$$

Solution. Let $\int_{-1}^1 f(x)dx = w_1 f(0) + w_2 f(1/\sqrt{2}) + w_3 f(-1/\sqrt{2})$.
As in previous examples, the system of equations for $f(x) = 1, x, x^2$ are
$2 = w_1 + w_2 + w_3,\ 0 = \frac{1}{\sqrt{2}}w_2 - \frac{1}{\sqrt{2}}w_3,\ \frac{2}{3} = \frac{1}{2}w_2 + \frac{1}{2}w_3$.
Solution of these equations is $w_1 = w_2 = w_3 = 2/3$. Hence the result follows.

Example 29.11.3 Write down the quadrature polynomial which takes the same values as $f(x)$ at $x = -1, 0, 1$ and integrate it to obtain the integration formula

$$\int_{-1}^1 f(x)dx = \frac{1}{3}[f(-1) + 4f(0) + f(1)].$$

Assuming the error to have the form $Af^{iv}(\xi), -1 < \xi < 1$, find the value of A.

Solution. To find the quadratic polynomial, the Lagrange's interpolation formula is used. The Lagrange's interpolation for the points $-1, 0, 1$ is

$$\begin{aligned} f(x) &= \frac{(x-0)(x-1)}{(-1-0)(-1-1)}f(-1) + \frac{(x+1)(x-1)}{(0+1)(0-1)}f(0) + \frac{(x+1)(x-0)}{(1+1)(1-0)}f(1) \\ &= \frac{1}{2}(x^2 - x)f(-1) - (x^2 - 1)f(0) + \frac{1}{2}(x^2 + x)f(1) \\ &= \left[\frac{1}{2}f(-1) - f(0) + \frac{1}{2}f(1)\right]x^2 + \left[\frac{1}{2}f(1) - \frac{1}{2}f(-1)\right]x + f(0). \end{aligned}$$

This is the required quadratic polynomial. Integrating it between -1 and 1.

$$\int_{-1}^{1} f(x)dx = \frac{1}{2}f(-1)\int_{-1}^{1}(x^2 - x)dx - f(0)\int_{-1}^{1}(x^2 - 1)dx + \frac{1}{2}f(1)\int_{-1}^{1}(x^2 + x)dx$$

$$= \frac{1}{2}f(-1).\frac{2}{3} - f(0)\Big(-\frac{4}{3}\Big) + \frac{1}{2}f(1).\frac{2}{3}$$
$$= \frac{1}{3}[f(-1) + 4f(0) + f(1)].$$

Here the error is of the form $Af^{iv}(\xi)$. Let the error be $E = Af^{iv}(\xi) = \frac{C}{4!}f^{iv}(\xi)$, $-1 < \xi < 1$. This indicates that the above formula gives exact result for the polynomials of degree up to 3 and has error for the polynomial of degree 4. Let $f(x) = x^4$. Then the value of $\int_{-1}^{1} x^4 dx$ obtain from the above formula is $\frac{1}{3}[(-1)^4 + 4.0^4 + 1^4] = \frac{2}{3}$ and the exact value is $\int_{-1}^{1} x^4 dx = \frac{2}{5}$.

Therefore, $C = \int_{-1}^{1} x^4 dx - \frac{2}{3} = -\frac{4}{15}$.

Hence $A = \frac{C}{4!} = -\frac{1}{90}$.

29.12 Least Squares Approximation

In science and engineering, the experimental data are usually viewed by plotting as graphs on plane paper. But, the problem is: *what is the 'best curve' for a given set of data ?* If n data points $(x_i, y_i), i = 1, 2, \ldots, n$ are given, then an nth degree polynomial can be constructed using interpolation methods, such as Lagrange's, Newton's etc. But, the handling of higher degree polynomial is practically difficult, though it gives exact values at the given nodes $x_0, x_1, \ldots, x_n$, and errors at the other points. For the given data points we can construct lower degree polynomials such as linear, quadratic etc. and other types of curve, viz., geometric, exponential etc., using least squares method, which minimizes the sum of squares of the absolute errors. The curve fitted by this method not always give exact values at the given nodes $x_0, x_1, \ldots, x_n$ and other points.

The least squares method is used to fit polynomials of different degrees, other special curves and also to fit orthogonal polynomials, etc.

29.13 General Least Squares Method

Let $(x_i, y_i), i = 1, 2, \ldots, n$ be a bivariate sample of size n. The problem is to fit the curve

$$y = g(x; a_0, a_1, \ldots, a_k), \tag{29.74}$$

where $a_0, a_1, \ldots, a_k$ are the unknown parameters to be determined based on the given sample values such that the error is minimum.

When $x = x_i$, the value of y obtained from (29.74) is denoted by Y_i and it is given by

$$Y_i = g(x_i; a_0, a_1, \ldots, a_k). \tag{29.75}$$

The quantity Y_i is called the **expected** or **predicted** value of y corresponding to $x = x_i$ and y_i is called the **observed value** of y. These two values, in general, are different as the observed values not necessarily lie on the curve (29.74).

The difference $(y_i - Y_i)$ is called the **residual** corresponding to $x = x_i$. The parameters $a_0, a_1, \ldots, a_k$ are chosen by least squares method in such a way that the sum of the squares of residuals, S, is minimum.

Now,

$$S = \sum_{i=1}^{n}(y_i - Y_i)^2 = \sum_{i=1}^{n}[y_i - g(x_i; a_0, a_1, \ldots, a_k)]^2. \tag{29.76}$$

The expression for S contains $(k+1)$ unknowns $a_0, a_1, \ldots, a_k$.

The value of S will be minimum if

$$\frac{\partial S}{\partial a_0} = 0, \quad \frac{\partial S}{\partial a_1} = 0, \quad \ldots, \quad \frac{\partial S}{\partial a_k} = 0. \tag{29.77}$$

These equations are called **normal equations**. Solution of these $(k+1)$ equations give the values of the parameters $a_0, a_1, \ldots, a_k$. Let $a_0 = a_0^*, a_1 = a_1^*, \ldots, a_k = a_k^*$ be the solution of the system of equations (29.77).

Then the fitted curve is

$$y = g(x; a_0^*, a_1^*, \ldots, a_k^*). \tag{29.78}$$

The sum of the squares of residuals is obtained from the equation

$$S = \sum_{i=1}^{n}(y_i - Y_i)^2 = \sum_{i=1}^{n}[y_i - g(x_i; a_0^*, a_1^*, \ldots, a_k^*)]^2. \tag{29.79}$$

29.14 Least Squares Method for Continuous Data

In the previous sections, the least squares method is considered for the discrete data. This method is also applicable for continuous data.

Let $y = f(x)$ be a continuous function on $[a, b]$ and it is to be approximated by the kth degree polynomial

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_k x^k. \tag{29.80}$$

In this case, the sum of the squares of residuals S is defined as

$$S = \int_a^b w(x)[y - (a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)]^2\, dx \tag{29.81}$$

where $w(x)$ is a suitable weight function.

The necessary conditions for minimum S are

$$\frac{\partial S}{\partial a_0} = \frac{\partial S}{\partial a_1} = \cdots = \frac{\partial S}{\partial a_k} = 0. \tag{29.82}$$

These give the normal equations as

$$\begin{aligned}
-2\int_a^b w(x)[y - (a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)]\, dx &= 0\\
-2\int_a^b w(x)[y - (a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)]x\, dx &= 0\\
-2\int_a^b w(x)[y - (a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)]x^2\, dx &= 0\\
\vdots \qquad\qquad & \qquad \vdots\\
-2\int_a^b w(x)[y - (a_0 + a_1x + a_2x^2 + \cdots + a_kx^k)]x^k\, dx &= 0.
\end{aligned}$$

After simplification these equations reduce to

$$\begin{aligned}
a_0\int_a^b w(x)dx + a_1\int_a^b xw(x)\, dx + \cdots + a_k\int_a^b x^kw(x)\, dx &= \int_a^b w(x)y\, dx\\
a_0\int_a^b xw(x)dx + a_1\int_a^b x^2w(x)\, dx + \cdots + a_k\int_a^b x^{k+1}w(x)\, dx &= \int_a^b w(x)xy\, dx\\
a_0\int_a^b x^2w(x)dx + a_1\int_a^b x^3w(x)\, dx + \cdots + a_k\int_a^b x^{k+2}w(x)\, dx &= \int_a^b w(x)x^2y\, dx\\
\vdots \qquad\qquad & \qquad \vdots\\
a_0\int_a^b x^kw(x)dx + a_1\int_a^b x^{k+1}w(x)\, dx + \cdots + a_k\int_a^b x^{2k}w(x)\, dx &= \int_a^b w(x)x^ky\, dx
\end{aligned} \tag{29.83}$$

Since $w(x)$ and $y = f(x)$ are known, the above equations form a system of linear equations with $(k+1)$ unknowns $a_0, a_1, \ldots, a_k$. This system of equations possesses a unique solution. If

$$a_0 = a_0^*, a_1 = a_1^*, \ldots, a_k = a_k^*$$

is the solution for $a_0, a_1, \ldots, a_k$ then the approximate polynomial is

$$y = a_0^* + a_1^*x + a_2^*x^2 + \cdots + a_k^*x^k.$$

Example 29.14.1 Construct a least squares quadratic approximation to the function

$$y = e^x$$

on $[0, 1]$.

Solution. Let the weight function be $w(x) = 1$ and $y = a_0 + a_1x + a_2x^2$ be the required quadratic approximation. Then the normal equations are

$$\begin{aligned} a_0\int_0^1 dx + a_1\int_0^1 x\,dx + a_2\int_0^1 x^2\,dx &= \int_0^1 e^x\,dx \\ a_0\int_0^1 x\,dx + a_1\int_0^1 x^2\,dx + a_2\int_0^1 x^3\,dx &= \int_0^1 xe^x\,dx \\ a_0\int_0^1 x^2\,dx + a_1\int_0^1 x^3\,dx + a_2\int_0^1 x^4\,dx &= \int_0^1 x^2e^x\,dx. \end{aligned}$$

After simplification these equations reduce to

$$\begin{aligned} a_0 + \frac{1}{2}a_1 + \frac{1}{3}a_2 &= e - 1, \\ \frac{1}{2}a_0 + \frac{1}{3}a_1 + \frac{1}{4}a_2 &= 1, \\ \frac{1}{3}a_0 + \frac{1}{4}a_1 + \frac{1}{5}a_2 &= e - 2. \end{aligned}$$

The solution of these system of equations is

$$a_0 = 1.0129913, a_1 = 0.8511251, a_2 = 0.8391840.$$

Thus the least squares approximation to $y = e^x$ is

$$y_l = 1.0129913 + 0.8511251x + 0.8391840x^2. \tag{29.84}$$

It is well known that the Taylor's series expansion of $y = e^x$ up to second and third degree terms are

$$y_2 = 1 + x + \frac{x^2}{2} \tag{29.85}$$

$$y_3 = 1 + x + \frac{x^2}{2} + \frac{x^3}{6}. \tag{29.86}$$

The values of y obtained from (29.84), (29.85) and (29.86) are listed below.

x	y_1	y_2	y_3	Exact
0.0	1.01299	1.00000	1.00000	1.00000
0.1	1.10650	1.10500	1.10517	1.10517
0.2	1.21678	1.22000	1.22133	1.22140
0.3	1.34386	1.34500	1.34950	1.34986
0.4	1.48771	1.48000	1.49067	1.49182
0.5	1.64835	1.62500	1.64583	1.64872
0.6	1.82577	1.78000	1.81600	1.82212
0.7	2.01998	1.94500	2.00217	2.01375
0.8	2.23097	2.12000	2.20533	2.22554
0.9	2.45874	2.30500	2.42650	2.45960
1.0	2.70330	2.50000	2.66667	2.71828

This table shows that the least squares quadratic approximation gives more better result as compared to second degree Taylor's series approximation, even to third degree approximation.

29.15 Approximation Using Orthogonal Polynomials

In the previous section, a function is approximated as a polynomial containing the terms 1, $x, x^2, \ldots, x^k$. These terms are called **base functions**, because, any function or even discrete data are approximated based on these functions.

Now, we assume that the base functions are some orthogonal polynomials $f_0(x)$, $f_1(x)$, ..., $f_k(x)$. Let the given function be approximated as

$$y = a_0 f_0(x) + a_1 f_1(x) + \cdots + a_k f_k(x), \tag{29.87}$$

where $f_i(x)$ is a polynomial in x of degree i. Then as in the previous cases, the residue is given by

$$S = \int_a^b w(x)[y - \{a_0 f_0(x) + a_1 f_1(x) + \cdots + a_k f_k(x)\}]^2 \, dx. \tag{29.88}$$

For minimum S,

$$\frac{\partial S}{\partial a_0} = 0, \frac{\partial S}{\partial a_1} = 0, \ldots, \frac{\partial S}{\partial a_k} = 0.$$

These equations give the following normal equations.

$$-2\int_a^b w(x)[y - \{a_0 f_0(x) + a_1 f_1(x) + \cdots + a_k f_k(x)\}] f_0(x) \, dx = 0$$

$$-2\int_a^b w(x)[y - \{a_0 f_0(x) + a_1 f_1(x) + \cdots + a_k f_k(x)\}] f_1(x) \, dx = 0$$

$$\vdots \qquad\qquad \vdots$$

$$-2\int_a^b w(x)[y - \{a_0 f_0(x) + a_1 f_1(x) + \cdots + a_k f_k(x)\}] f_k(x) \, dx = 0.$$

After simplification, the ith equation becomes

$$\begin{aligned} & a_0 \int_a^b w(x) f_0(x) f_i(x)\, dx + a_1 \int_a^b w(x) f_1(x) f_i(x)\, dx + \cdots \\ & + a_i \int_a^b w(x) f_i^2(x)\, dx + \cdots + a_k \int_a^b w(x) f_k(x) f_i(x)\, dx \\ & = \int_a^b w(x)\, y\, f_i(x)\, dx, \end{aligned} \tag{29.89}$$

$i = 0, 1, 2, \ldots, n$.

A set of polynomial $\{f_0(x), f_1(x), \ldots, f_k(x)\}$ is said to be **orthogonal** with respect to the weight function $w(x)$ if

$$\int_a^b f_i(x) f_j(x) w(x)\, dx = \begin{cases} 0, & \text{if } i \neq j \\ \displaystyle\int_a^b f_i^2(x) w(x)\, dx, & \text{if } i = j. \end{cases} \tag{29.90}$$

Incorporating this property, equation (29.89) becomes

$$a_i \int_a^b w(x) f_i^2(x)\, dx = \int_a^b w(x) y f_i(x)\, dx, \qquad i = 0, 1, 2, \ldots, n.$$

That is,

$$a_i = \frac{\displaystyle\int_a^b w(x)\, y\, f_i(x)\, dx}{\displaystyle\int_a^b w(x) f_i^2(x)\, dx}, \qquad i = 0, 1, 2, \ldots, n. \tag{29.91}$$

From this relation one can determine the values of $a_0, a_1, \ldots, a_k$ and the least squares approximation is obtained by substituting these values in (29.87). But, the functions $f_0(x), f_1(x), \ldots, f_k(x)$ are unknown. Several orthogonal functions are available in literature, some of them are listed in Table 29.4.

Table 29.4: Some standard orthogonal polynomials.

Name	$f_i(x)$	Interval	$w(x)$
Legendre	$P_n(x)$	$[-1, 1]$	1
Leguerre	$L_n(x)$	$[0, \infty)$	e^{-x}
Hermite	$H_n(x)$	$(-\infty, \infty)$	e^{-x^2}
Chebyshev	$T_n(x)$	$[-1, 1]$	$(1 - x^2)^{-1/2}$

Based on the problem, any one of them can be selected to fit a function.

Gram-Schmidt orthogonalization process

Let $f_i(x)$ be a polynomial in x of degree i and $\{f_i(x)\}$ be a given sequence of polynomials. Then the sequence of orthogonal polynomials $\{f_i^*(x)\}$ over the interval $[a,b]$ with respect to the weight function $w(x)$ can be generated by the relation

$$f_i^*(x) = x^i - \sum_{r=0}^{i-1} a_{ir} f_r^*(x), \qquad i = 1, 2, \ldots, n, \tag{29.92}$$

for suitable constants a_{ir}, and $f_0^*(x) = 1$.

To find a_{ir}, multiplying (29.92) by $w(x)f_k^*(x), 0 \le k \le i-1$ and integrating over $[a,b]$. Then

$$\int_a^b f_i^*(x) f_k^*(x) w(x)\, dx = \int_a^b x^i f_k^*(x) w(x)\, dx - \int_a^b \sum_{r=0}^{i-1} a_{ir} f_r^*(x) f_k^*(x) w(x)\, dx.$$

Using the property of orthogonal polynomial, this equation reduces to

$$\int_a^b x^i f_k^*(x) w(x)\, dx - \int_a^b a_{ik} f^{*2}{}_k(x) w(x)\, dx = 0$$

$$\text{or,} \quad a_{ik} = \frac{\int_a^b x^i \ f_k^*(x) \ w(x)\, dx}{\int_a^b f_k^{*2}(x) \ w(x)\, dx}, \qquad 0 \le k \le i-1.$$

Thus the set of orthogonal polynomials $\{f_i^*(x)\}$ are given by

$$f_0^*(x) = 1$$

$$f_i^*(x) = x^i - \sum_{r=0}^{i-1} a_{ir} f_r^*(x), \qquad i = 1, 2, \ldots, n$$

$$\text{where } a_{ir} = \frac{\int_a^b x^i \ f_k^*(x) \ w(x)\, dx}{\int_a^b f_k^{*2}(x) \ w(x)\, dx}. \tag{29.93}$$

For the discrete data, the integral is replaced by summation.

Note 29.15.1 It may be noted that Gram-Schmidt process generates a sequence of monic (leading coefficient unity) orthogonal polynomials.

Example 29.15.1 Use Gram-Schmidt orthogonalization process to determine the first four orthogonal polynomials on $[-1,1]$ with respect to the weight function $w(x) = 1$.

Solution. Let $f_0^*(x) = 1$.

Then $f_1^*(x) = x - a_{10} f_0^*(x)$, where $a_{10} = \dfrac{\int_{-1}^1 x\, dx}{\int_{-1}^1 dx} = 0$.

Thus $f_1^*(x) = x$.

The second orthogonal polynomial is
$f_2^*(x) = x^2 - a_{20}f_0^*(x) - a_{21}f_1^*(x)$

$$\text{where} \qquad a_{20} = \frac{\int_{-1}^{1} x^2\,dx}{\int_{-1}^{1} dx} = \frac{1}{3}, \qquad a_{21} = \frac{\int_{-1}^{1} x^2.x\,dx}{\int_{-1}^{1} x^2\,dx} = 0.$$

Thus $f_2^*(x) = x^2 - \frac{1}{3} = \frac{1}{3}(3x^2 - 1)$.

Again, $f_3^*(x) = x^3 - a_{30}f_0^*(x) - a_{31}f_1^*(x) - a_{32}f_2^*(x)$
where

$$a_{30} = \frac{\int_{-1}^{1} x^3\,dx}{\int_{-1}^{1} dx} = 0, \quad a_{31} = \frac{\int_{-1}^{1} x^3.x\,dx}{\int_{-1}^{1} x^2\,dx} = \frac{3}{5}, \quad a_{32} = \frac{\int_{-1}^{1} x^3.\frac{1}{3}(3x^2-1)\,dx}{\int_{-1}^{1} \frac{1}{9}(3x^2-1)^2\,dx} = 0.$$

Hence $f_3^*(x) = x^3 - \frac{3}{5}x = \frac{1}{5}(5x^3 - 3x)$.

Thus the first three orthogonal polynomials are $f_0^*(x) = 1, f_1^*(x) = x$, $f_2^*(x) = \frac{1}{3}(3x^2 - 1)$ and $f_3^*(x) = \frac{1}{5}(5x^3 - 3x)$.

These polynomials are called (monic) Legendre polynomials.

29.16 Approximation of Functions

The problem of approximation of functions is an important problem in numerical analysis, due to its wide applications in science and engineering, specially in computer graphics, to plot two- and three-dimensional figures. Under some restriction, almost all functions can be approximated using Taylor's series with base functions $1, x, x^2, \ldots$. It is mentioned earlier that the method of approximation of a function using Taylor's series is not economic. But, the approximation using orthogonal polynomials is economic. Among several orthogonal polynomials, Chebyshev polynomials seem to be more economic.

29.16.1 Chebyshev polynomials

The Chebyshev polynomial, $T_n(x)$, of degree n over the interval $[-1, 1]$ is defined by

$$T_n(x) = \cos(n\cos^{-1}x), \qquad n = 0, 1, 2, \ldots. \tag{29.94}$$

This expression can also be written as

$$T_n(x) = \cos n\theta, \qquad \text{where} \qquad x = \cos\theta. \tag{29.95}$$

Thus $T_0(x) = 1$ and $T_1(x) = x$.

From equation (29.94), it is easy to observe that $T_n(x) = T_{-n}(x)$.

Also, $T_{2n}(x) = T_{2n}(-x)$ and $T_{2n+1}(-x) = -T_{2n+1}(x)$, i.e., $T_n(x)$ is even or odd functions according as n is even or odd.

Now, from the trigonometric formula

$$\cos(n-1)\theta + \cos(n+1)\theta = 2\cos n\theta . \cos\theta,$$

the recurrence relation for Chebyshev polynomial is obtained as

$$T_{n-1}(x) + T_{n+1}(x) = 2xT_n(x),$$

$$\text{i.e., } T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \qquad n = 1, 2, 3, \ldots . \tag{29.96}$$

The zeros of Chebyshev polynomial $T_n(x)$ lie between $[-1, 1]$, being all distinct and given by

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n}\right), \qquad i = 0, 1, 2, \ldots, n-1. \tag{29.97}$$

These values are called the **Chebyshev abscissas or nodes.**

From the equation (29.95), it is obvious that $|T_n(x)| \leq 1$ for $-1 \leq x \leq 1$. Thus extreme values of $T_n(x)$ are -1 and 1.

From the recurrence relation (29.96), it is observed that the relation doubles the leading coefficient of $T_n(x)$ to get the leading coefficient of $T_{n+1}(x)$. Also $T_1(x) = x$. Thus the coefficient of x^n in $T_n(x)$ is $2^{n-1}, n \geq 1$.

This polynomial satisfies a second order differential equation

$$(1-x^2)\frac{d^2y}{dx^2} - x\frac{dy}{dx} + n^2 y = 0. \tag{29.98}$$

To justify it, let $y = T_n(x) = \cos n\theta, x = \cos\theta$.

$$\text{Then } \frac{dy}{dx} = \frac{n\sin n\theta}{\sin\theta} \text{ and } \frac{d^2y}{dx^2} = \frac{-n^2\cos n\theta + n\sin n\theta\cot\theta}{\sin^2\theta} = \frac{-n^2 y + x\dfrac{dy}{dx}}{1-x^2}.$$

This gives $(1-x^2)\dfrac{d^2y}{dx^2} - x\dfrac{dy}{dx} + n^2 y = 0.$

Orthogonal property

We mentioned earlier that the Chebyshev polynomials are orthogonal and they are orthogonal with respect to the weight function $(1-x^2)^{-1/2}$, i.e.,

$$\int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\,dx = 0 \qquad \text{for } m \neq n.$$

To prove it, let $x = \cos\theta$. Then

$$I = \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\,dx = \int_0^{\pi} T_n(\cos\theta)T_m(\cos\theta)\,d\theta$$

$$= \int_0^{\pi} \cos n\theta\, \cos m\theta\, d\theta = \frac{1}{2}\int_0^{\pi} [\cos(m+n)\theta + \cos(m-n)\theta]\,d\theta$$

$$= \frac{1}{2}\left[\frac{\sin(m+n)\theta}{m+n} + \frac{\sin(m-n)\theta}{m-n}\right]_0^{\pi}.$$

Now, when $m \neq n$ it gives $I = 0$, when $m = n = 0$ then $I = \pi$ and when $m = n \neq 0$ then $I = \frac{\pi}{2}$.
Thus

$$\int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}}\,dx = \begin{cases} 0, & \text{if } m \neq n \\ \pi, & \text{if } m = n = 0 \\ \pi/2, & \text{if } m = n \neq 0 \end{cases} \tag{29.99}$$

Example 29.16.1 Use Chebyshev polynomials to find least squares approximation of second degree for $f(x) = \sqrt{1-x^2}$ on $[-1, 1]$.

Solution. Let $f(x) \simeq a_0T_0(x) + a_1T_1(x) + a_2T_2(x)$. Then the residue is

$$S = \int_{-1}^{1} w(x)[f(x) - \{a_0T_0(x) + a_1T_1(x) + a_2T_2(x)\}]^2\,dx,$$

where $w(x) = (1-x^2)^{-1/2}$.

For minimum S,

$$\frac{\partial S}{\partial a_0} = 0, \qquad \frac{\partial S}{\partial a_1} = 0, \qquad \frac{\partial S}{\partial a_2} = 0.$$

Now,

$$\frac{\partial S}{\partial a_0} = -2\int_{-1}^{1} w(x)[f(x) - \{a_0T_0(x) + a_1T_1(x) + a_2T_2(x)\}]T_0(x)\,dx = 0.$$

Using the property of orthogonal polynomials, this equation is simplified as

$$\int_{-1}^{1} w(x)f(x)T_0(x)\,dx - \int_{-1}^{1} w(x)a_0T_0^2(x)\,dx = 0.$$

This equation gives

$$a_0 = \frac{\int_{-1}^{1} w(x)f(x)T_0(x)\,dx}{\int_{-1}^{1} w(x)T_0^2(x)\,dx} = \frac{1}{\pi}\int_{-1}^{1} \frac{\sqrt{1-x^2}}{\sqrt{1-x^2}}\,dx = \frac{2}{\pi}$$

[Using (29.99) and $T_0(x) = 1$].

Similarly, from the relations $\frac{\partial S}{\partial a_1} = 0$ and $\frac{\partial S}{\partial a_2} = 0$, we obtain

$$a_1 = \frac{\int_{-1}^{1} w(x)f(x)T_1(x)\,dx}{\int_{-1}^{1} w(x)T_1^2(x)\,dx} = \frac{2}{\pi}\int_{-1}^{1} x\,dx = 0$$

$$a_2 = \frac{\int_{-1}^{1} w(x)f(x)T_2(x)\,dx}{\int_{-1}^{1} w(x)T_2^2(x)\,dx} = \frac{2}{\pi}\int_{-1}^{1} (2x^2-1)\,dx = -\frac{4}{3\pi}.$$

Thus the Chebyshev approximation is

$$f(x) = \frac{2}{\pi}T_0(x) - \frac{4}{3\pi}T_2(x) = \frac{2}{\pi} - \frac{4}{3\pi}(2x^2-1) = 1.061033 - 0.8488264x^2.$$

Using the recurrence relation (29.96) and $T_0(x) = 1, T_1(x) = x$, one can generate Chebyshev polynomials. The first seven Chebyshev polynomials are

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \\ T_5(x) &= 16x^5 - 20x^3 + 5x \\ T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\ T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x. \end{aligned} \tag{29.100}$$

The graph of first four Chebyshev polynomials are shown in Figure 29.1.

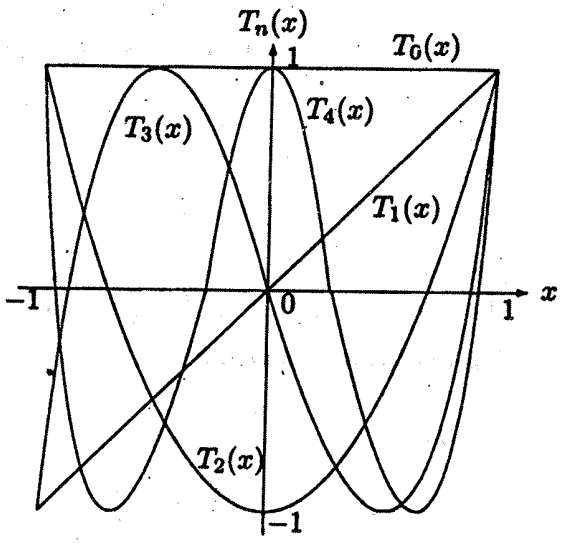


Figure 29.1: Chebyshev polynomials $T_n(x), n = 0, 1, 2, 3, 4$.

The equation (29.100) suggests that all powers of x can be expressed in terms of Chebyshev polynomials, as

$$
\begin{aligned}
1 &= T_0(x) \\
x &= T_1(x) \\
x^2 &= \tfrac{1}{2}[T_0(x) + T_2(x)] \\
x^3 &= \tfrac{1}{4}[3T_1(x) + T_3(x)] \\
x^4 &= \tfrac{1}{8}[3T_0(x) + 4T_2(x) + T_4(x)] \\
x^5 &= \tfrac{1}{16}[10T_1(x) + 5T_3(x) + T_5(x)] \\
x^6 &= \tfrac{1}{32}[10T_0(x) + 15T_2(x) + 6T_4(x) + T_6(x)] \\
x^7 &= \tfrac{1}{64}[35T_1(x) + 21T_3(x) + 7T_5(x) + T_7(x)].
\end{aligned} \tag{29.101}
$$

Thus every polynomial can be approximated using Chebyshev polynomials.

Example 29.16.2 Express the polynomial $x^3 + 2x^2 - 7$ in terms of Chebyshev polynomials.

Solution. $x^3 + 2x^2 - 7 = \frac{1}{4}[3T_1(x) + T_3(x)] + 2.\frac{1}{2}[T_0(x) + T_2(x)] - 7T_0(x)$
$= \frac{1}{4}T_3(x) + T_2(x) + \frac{3}{4}T_1(x) - 6T_0(x).$

29.16.2 Expansion of function using Chebyshev polynomials

Let $y = f(x)$ be a function to be approximated by Chebyshev polynomials. This can be done as $f(x) = a_0T_0(x) + a_1T_1(x) + a_2T_2(x) + \cdots + a_kT_k(x)$, where the coefficients a_i are given by

$$
a_i = \frac{\displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\, yT_i(x)\, dx}{\displaystyle\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}}\, T_i^2(x)\, dx}, \qquad i = 0, 1, 2, \ldots, n.
$$

But, the integral in the denominator of a_i is improper and it is not easy to evaluate. So, a discretization technique is adopted here to approximate a function using Chebyshev polynomials. The orthogonal property for discrete case is stated below.

$$
\sum_{k=0}^{n} T_i(x_k)T_j(x_k) = \begin{cases} 0, & \text{if } i \neq j \\ \dfrac{n+1}{2}, & \text{if } i = j \neq 0 \\ n+1, & \text{if } i = j = 0 \end{cases} \tag{29.102}
$$

where $x_k = \cos\left(\dfrac{(2k+1)\pi}{2n+2}\right), \quad k = 0, 1, 2, \ldots, n.$

This result is used to establish the following theorem.

Theorem 29.1 (Chebyshev approximation). *The function $f(x)$ can be approximated over $[-1, 1]$ by Chebyshev polynomials as*

$$f(x) \simeq \sum_{i=0}^{n} a_i T_i(x). \tag{29.103}$$

The coefficients a_i are given by

$$a_0 = \frac{1}{n+1}\sum_{j=0}^{n} f(x_j)T_0(x_j) = \frac{1}{n+1}\sum_{j=0}^{n} f(x_j), \tag{29.104}$$

$$x_j = \cos\left(\frac{(2j+1)\pi}{2n+2}\right)$$

$$\text{and } a_i = \frac{2}{n+1}\sum_{j=0}^{n} f(x_j)T_i(x_j)$$

$$= \frac{2}{n+1}\sum_{j=0}^{n} f(x_j)\cos\left(\frac{(2j+1)i\pi}{2n+2}\right) \tag{29.105}$$

$$\text{for } i = 1, 2, \ldots, n.$$

Example 29.16.3 Approximate $f(x) = e^x$ to second order Chebyshev approximation over the interval [0, 1].

Solution. The second order Chebyshev approximation is

$$f(x) \simeq \sum_{i=0}^{2} a_i T_i(x).$$

The Chebyshev nodes are given by

$$x_j = \cos\left(\frac{(2j+1)\pi}{6}\right), j = 0, 1, 2.$$

$x_0 = 0.86660254, x_1 = 0, x_2 = -0.8660254.$

$$a_0 = \frac{1}{3}[f(x_0) + f(x_1) + f(x_2)] = 1.2660209$$

$$a_1 = \frac{2}{3}\sum_{j=0}^{2} f(x_j)\cos\left(\frac{(2j+1)\pi}{6}\right) = 1.1297721$$

$$a_2 = \frac{2}{3}\sum_{j=0}^{2} f(x_j)\cos\left(\frac{2(2j+1)\pi}{6}\right) = 0.26602093.$$

Therefore, $f(x) \simeq 1.2660209\ T_0(x) + 1.1297721\ T_1(x) + 0.2660209\ T_2(x)$.

Minimax principle

It is well known that the error in polynomial interpolation is

$$E_n(x) = w_{n+1}(x)\frac{f^{(n+1)}(\xi)}{(n+1)!},$$

where $w_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$ is a polynomial of degree $(n + 1)$.

Now,

$$|E_n(x)| \leq \max_{-1\leq x\leq 1} |w_{n+1}(x)| \frac{\max_{-1\leq x\leq 1} |f^{(n+1)}(\xi)|}{(n+1)!}. \tag{29.106}$$

The expression $\max_{-1\leq x\leq 1} |f^{(n+1)}(\xi)|$ is fixed for a given function $f(x)$. Thus the error bound depends on the value of $|w_{n+1}(x)|$ and the value of $|w_{n+1}(x)|$ depends on the choice of the nodes $x_0, x_1, \ldots, x_n$. The maximum error depends on the product of $\max_{-1\leq x\leq 1} |w_{n+1}(x)|$ and $\max_{-1\leq x\leq 1} |f^{(n+1)}(\xi)|$. Russian mathematician Chebyshev invented that $x_0, x_1, \ldots, x_n$ should be chosen such that $w_{n+1}(x) = 2^{-n}T_{n+1}(x)$. The polynomial $2^{-n}T_{n+1}(x)$ is the monic Chebyshev polynomial.

If n is fixed, then all possible choices for $w_{n+1}(x)$ and thus among all possible choices for the nodes $x_0, x_1, \ldots, x_n$ on $[-1, 1]$, the polynomial $\tilde{T}_{n+1}(x) = 2^{-n}T_{n+1}(x)$ is the unique choice that satisfies the relation

$$\max_{-1\leq x\leq 1}\{|\tilde{T}_{n+1}(x)|\} \leq \max_{-1\leq x\leq 1}\{|w_{n+1}(x)|\}.$$

Moreover, $\max_{-1\leq x\leq 1}\{|\tilde{T}_{n+1}(x)|\} = 2^{-n}$ as $|T_{n+1}(x)| \leq 1$.

This property is called **minimax principle** and the polynomial

$$\tilde{T}_{n+1}(x) = 2^{-n}T_{n+1}(x) \qquad \text{or,} \qquad \tilde{T}_n(x) = 2^{1-n}T_n(x)$$

is called **minimax polynomial.**

29.16.3 Economization of power series

From the equation (29.101), it is observed that every polynomial of degree n can be expressed in terms of Chebyshev polynomials of same degree. If

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \tag{29.107}$$

be the given polynomial, then it can be expressed in the form

$$f(x) = b_0 + b_1T_1(x) + b_2T_2(x) + \cdots + b_nT_n(x). \tag{29.108}$$

For a large number of functions, it is computationally observed that the expansion of the form (29.108) converges more rapidly than the form (29.107). Thus representation of the form (29.108) is computationally better and the process is called **economization of the power series.** This is illustrated in the following example.

Example 29.16.4 Economize the power series

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

correct to four significant digits.

Solution. Since the result is required to four significant digits and the coefficients of the term x^8, $1/8! = 0.0000248$ can affect the fifth decimal place only, thus the terms after fourth term may be truncated.

Hence

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}. \tag{29.109}$$

Now express the above series using Chebyshev polynomials, as

$$\begin{aligned}\cos x &= T_0(x) - \frac{1}{2!}\cdot\frac{1}{2}[T_0(x) + T_2(x)] + \frac{1}{4!}\cdot\frac{1}{8}[3T_0(x) + 4T_2(x) + T_4(x)] \\ &\quad -\frac{1}{6!}\cdot\frac{1}{32}[10T_0(x) + 15T_2(x) + 6T_4(x) + T_6(x)] \\ &= 0.7651910 - 0.2298177\ T_2(x) + 0.0049479\ T_4(x) - 0.0000434\ T_6(x).\end{aligned}$$

Again, the term $0.0000434\ T_6(x)$ does not affect the fourth decimal place, so it is discarded and the economized series is

$$\cos x = 0.7651910 - 0.2298177\ T_2(x) + 0.0049479\ T_4(x).$$

In terms of x, the series is

$$\begin{aligned}\cos x &= 0.7651910 - 0.2298177\ (2x^2 - 1) + 0.0049479\ (8x^4 - 8x^2 + 1) \\ &= 0.9999566 - 0.4992186\ x^2 + 0.0395832\ x^4.\end{aligned}$$

This expression gives the values of $\cos x$ correct up to four decimal places.

29.17 Module Summary

In this module, numerical differentiation and integration methods are discussed. The Romberg's integration method is presented here. The very important quadrature formulae – Gauss-Legendre and Gauss-Chebyshev methods are studied here. The least square method for continuous case is presented. The Chebyshev polynomial is introduced and studied its several properties. The approximation of function by orthogonal polynomials and Chebyshev polynomial is also discussed. We have shown that a function can be expressed in terms of Chebyshev polynomial.

29.18 Self Assessment Questions

1. From the following table of values, estimate $y'(1.05)$ and $y''(1.05)$:

x :	1.00	1.05	1.10	1.15	1.20	1.25
y :	1.1000	1.1347	1.1688	1.1564	1.2344	1.2345

2. A slider in a machine moves along a fixed straight rod. Its distance x cm along the rod is given below for various values of time t (second). Find the velocity of the slider and its acceleration when $t = 0.3$ sec.

t :	0.3	0.4	0.5	0.6	0.7	0.8
x :	3.364	3.395	3.381	3.324	3.321	3.312

Use the formula based on Newton's forward difference interpolation to find the velocity and acceleration.

3. Use two-point and three-point formulae to find the values of $f'(2.0)$ and $f''(2.0)$.

x :	0.5	1.0	1.5	2.0	2.5	3.0	3.5
$f(x)$:	-0.30103	0.00000	0.17609	0.30103	0.39794	0.47712	0.54407

4. Find the value of $\int_0^2 (1 + e^{-x} \sin 4x)\, dx$ using Simpson's 3/8 rule.

5. Evaluate the following integral using Simpson's 3/8 rule.

$$\int_0^{\pi/2} \frac{dx}{\sin^2 x + 2\cos^2 x}.$$

6. Find the values of the following integrals using Romberg integration starting from trapezoidal rule, correct up to five decimal points.
(a) $\int_1^2 \sqrt{4x - x^2}\, dx$, (b) $\int_{1/(2\pi)}^2 \sin(1/x)\, dx$.

7. Use three-point Gauss-Legendre formula to evaluate the integrals
(a) $\int_{-1}^1 \frac{1}{1+x^2} dx$, (b) $\int_0^{\pi/2} \sin x\, dx$.

8. The three-point Gauss-Legendre formula is

$$\int_{-1}^1 f(x)dx = \frac{5f(-\sqrt{0.6}) + 8f(0) + 5f(\sqrt{0.6})}{9}.$$

Show that the formula is exact for $f(x) = 1, x, x^2, x^3, x^4, x^5$.

9. Find the value of $\int_0^2 \frac{x}{1+x^3}\,dx$ using four-point and six-point Gauss-Legendre quadrature formulae.

10. Find the value of $\int_{-1}^{1} (1-x^2)\sin x\,dx$ using three-point Gauss-Chebyshev quadrature.

11. Find the weights w_1, w_2, w_3 so that the relation

$$\int_{-1}^{1} f(x)\,dx = w_1 f(-\sqrt{0.6}) + w_2 f(0) + w_3 f(\sqrt{0.6})$$

is exact for the functions $f(x) = 1, x, x^2$.

12. Express the following as a polynomials in x.
(a) $T_5(x) + 5T_3(x) - 11T_2(x) + 3T_1(x)$, and
(b) $2T_4(x) - 7T_3(x) + 2T_2(x) + 5T_0(x)$.

13. Suppose $y = 1 - \frac{x}{2!} + \frac{x^2}{4!} - \frac{x^3}{6!} + \frac{x^4}{8!} - \cdots$. Economize this series if the fourth decimal place is not to be affected, near $x = 1$.

29.19 References

1. M.Pal, Numerical Analysis for Scientists and Engineers: Theory and C Programs, Narosa, 2007.

2. M.K. Jain, S.R.K. Iyengar and R.K. Jain, Numerical Methods for Scientific and Engineering Computation, New Age International (P) Limited, New Delhi, 1984.

3. J.H. Mathews, Numerical Methods for Mathematics, Science, and Engineering, 2nd ed., Prentice-Hall, Inc., N.J., U.S.A., 1992.

4. S.S.Sastry, Introductory Method of Numerical Analysis, PHI, 2006.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III

Group-B

Module No.- 30

Numerical Analysis

(Solution of a System of Equations)

Module Structure:

30.1 Introduction

In B.Sc., you have learnt Crammer's rule, matrix inverse method, Gauss-Seidal iteration method to solve a system of linear equations. Here you will learn matrix inverse method to solve a system of linear equations and Newton-Rapshon method to solve a system of non-linear equations.

A system of m linear equations in n unknowns (variables) is written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &\quad \cdots \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n &= b_i \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &\quad \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m. \end{aligned} \tag{30.1}$$

The quantities $x_1, x_2, \ldots, x_n$ are the **unknowns (variables)** of the system and $a_{11}, a_{12}, \ldots, a_{mn}$ are the **coefficients** of the unknowns of the system. The numbers $b_1, b_2, \ldots, b_m$ are **constant or free terms** of the system.

The above system of equations (30.1) can be written as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, 2, \ldots, m. \tag{30.2}$$

Also, the system of equations (30.1) can be written in matrix form as

$$\mathbf{AX} = \mathbf{b}, \tag{30.3}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{bmatrix} \text{ and } X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_m \end{bmatrix}. \tag{30.4}$$

The system of linear equation (30.1) is **consistent** if it has a solution. If a system of linear equations has no solution, then it is **inconsistent** (or **incompatible**). A consistent system of linear equations may have one solution or several solutions and is said to be **determinate** if there is one solution and **indeterminate** if there are more than one solution.

A system in which the constant terms $b_1, b_2, \ldots, b_m$ are zero is called a **homogeneous** system.

Two basic techniques are used to solve a system of linear equations:

(i) direct method, and (ii) iteration method.

Several direct methods are used to solve a system of equations, among them following are most useful. (i) Cramer's rule, (ii) matrix inversion, (iii) Gauss elimination, (iv) decomposition, etc.

The most widely used iteration methods are (i) Jacobi's iteration, (ii) Gauss-Seidal's iteration, etc.

30.2 Objectives

Gone through this module the students will learn the following:

- Matrix inverse by partial pivoting
- Matrix inverse method to solve system of linear equations
- Convergence of Gauss-Seidal iteration method
- Newton-Rapshon method to solve a system of non-linear equations.

30.3 Keywords

Matrix inverse method, Gauss-Jordan method, Gauss-Seidal iteration method, rate of convergence, Newton-Rapshon method.

Direct Methods

30.4 Inverse of a Matrix

From the theory of matrices, it is well known that every square non-singular matrix has unique inverse. The inverse of a matrix **A** is defined by

$$\mathbf{A}^{-1} = \frac{\mathbf{adj}\ \mathbf{A}}{|\mathbf{A}|}. \tag{30.5}$$

The matrix **adj A** is called adjoint of **A** and defined as

$$\mathbf{adj}\ \mathbf{A} = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \cdots & \cdots & \cdots & \cdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \end{bmatrix},$$

where A_{ij} being the cofactor of a_{ij} in $|\mathbf{A}|$.

The main difficulty of this method is to compute the inverse of the matrix **A**. From the definition of **adj A** it is easy to observe that to compute the matrix **adj A**, we have to determine n^2 determinants each

of order $(n-1)$. So, it is very much time consuming. Many efficient methods are available to find the inverse of a matrix, among them **Gauss-Jordan** is most popular. In the following Gauss-Jordan method is discussed to find the inverse of a square non-singular matrix.

30.4.1 Gauss-Jordan Method

In this method, the given matrix **A** is augmented with a unit matrix of same size, i.e., if the order of **A** is $n \times n$ then the order of the augmented matrix $[\mathbf{A}\vdots\mathbf{I}]$ will be $n \times 2n$. The augmented matrix looks like

$$[\mathbf{A}\vdots\mathbf{I}] = \left[\begin{array}{cccc:cccc} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{array}\right]. \tag{30.6}$$

Then the inverse of **A** is computed in two stages. In the first stage, **A** is converted into an upper triangular form, using only elementary row operations. In the second stage, the upper triangular matrix (obtained in first stage) is reduced to an identity matrix by row operations. All these operations are operated on the augmented matrix $[\mathbf{A}\vdots\mathbf{I}]$. After completion of these stages, the augmented matrix $[\mathbf{A}\vdots\mathbf{I}]$ is turned to $[\mathbf{I}\vdots\mathbf{A}^{-1}]$, i.e., the inverse of **A** is obtained from the right half of augmented matrix.

Thus

$$[\mathbf{A}\vdots\mathbf{I}] \xrightarrow{\text{Gauss}-\text{Jordan}} [\mathbf{I}\vdots\mathbf{A}^{-1}].$$

At the end of the operations the matrix shown in (30.6) reduces to the following form:

$$\left[\begin{array}{cccc:cccc} 1 & 0 & \cdots & 0 & a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & 1 & \cdots & 0 & a'_{21} & a'_{22} & \cdots & a'_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & a'_{n1} & a'_{n2} & \cdots & a'_{nn} \end{array}\right]. \tag{30.7}$$

Example 30.4.1 Find the inverse of the following matrix $\mathbf{A} = \begin{bmatrix} 2 & 4 & 5 \\ 1 & -1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$.

Solution. The augmented matrix $[\mathbf{A}\vdots\mathbf{I}]$ can be written as

$$[\mathbf{A}\vdots\mathbf{I}] = \left[\begin{array}{ccc:ccc} 2 & 4 & 5 & 1 & 0 & 0 \\ 1 & -1 & 2 & 0 & 1 & 0 \\ 3 & 4 & 5 & 0 & 0 & 1 \end{array}\right]. \tag{30.8}$$

Stage I. *(Reduction to upper triangular form):*

In the first column 3 is the largest element, thus interchanging first (R_1) and third (R_3) rows to bring the pivot element 3 to the a_{11} position. Then (30.8) becomes

$$\left[\begin{array}{ccc:ccc} 3 & 4 & 5 & 0 & 0 & 1 \\ 1 & -1 & 2 & 0 & 1 & 0 \\ 2 & 4 & 5 & 1 & 0 & 0 \end{array}\right]$$

$$\sim \left[\begin{array}{ccc:ccc} 1 & 4/3 & 5/3 & 0 & 0 & 1/3 \\ 1 & -1 & 2 & 0 & 1 & 0 \\ 2 & 4 & 5 & 1 & 0 & 0 \end{array}\right] \quad R_1' = \tfrac{1}{3}R_1$$

$$\sim \left[\begin{array}{ccc:ccc} 1 & 4/3 & 5/3 & 0 & 0 & 1/3 \\ 0 & -7/3 & 1/3 & 0 & 1 & -1/3 \\ 0 & 4/3 & 5/3 & 1 & 0 & -2/3 \end{array}\right] \quad R_2' = R_2 - R_1;\ R_3' = R_3 - 2R_1$$

(The largest element (in magnitude) in the second column is $-\frac{7}{3}$, which is at the a_{22} position and so there is no need to interchange any rows).

$$\sim \left[\begin{array}{ccc:ccc} 1 & 4/3 & 5/3 & 0 & 0 & 1/3 \\ 0 & 1 & -1/7 & 0 & -3/7 & 1/7 \\ 0 & 0 & 13/7 & 1 & 4/7 & -6/7 \end{array}\right] \quad R_2' = -\tfrac{3}{7}R_2\ ;\ R_3' = R_3 - \tfrac{4}{3}R_2'$$

$$\sim \left[\begin{array}{ccc:ccc} 1 & 4/3 & 5/3 & 0 & 0 & 1/3 \\ 0 & 1 & -1/7 & 0 & -3/7 & 1/7 \\ 0 & 0 & 1 & 7/13 & 4/13 & -6/13 \end{array}\right] \quad R_3' = \tfrac{7}{13}R_3$$

Stage II. *(Make the left half a unit matrix):*

$$\sim \left[\begin{array}{ccc:ccc} 1 & 0 & 13/7 & 0 & 4/7 & 1/7 \\ 0 & 1 & -1/7 & 0 & -3/7 & 1/7 \\ 0 & 0 & 1 & 7/13 & 4/13 & -6/13 \end{array}\right] \quad R_1' = R_1 - \tfrac{4}{3}R_2$$

$$\sim \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1/13 & -5/13 & 1/13 \\ 0 & 0 & 1 & 7/13 & 4/13 & -6/13 \end{array}\right] \quad R_1' = R_1 - \tfrac{13}{7}R_3\ ;\ R_2' = R_2 + \tfrac{1}{7}R_3$$

The left hand becomes a unit matrix, thus the inverse of the given matrix is

$$\begin{bmatrix} -1 & 0 & 1 \\ 1/13 & -5/13 & 1/13 \\ 7/13 & 4/13 & -6/13 \end{bmatrix}.$$

30.5 Matrix Inverse Method

The system of equations (30.1) can be written in the matrix form (30.3) as

$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A}, \mathbf{b}$ and $\mathbf{x}$ are defined in (30.4).

The solution of $\mathbf{Ax} = \mathbf{b}$ is given

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}, \tag{30.9}$$

where $\mathbf{A}^{-1}$ is the inverse of the matrix $\mathbf{A}$.

Once the inverse of $\mathbf{A}$ is known then post multiplication of it with $\mathbf{b}$ gives the solution vector $\mathbf{x}$.

Example 30.5.1 Solve the following system of equations by matrix inverse method
$x + 2y + 3z = 10,\ x + 3y - 2z = 7,\ 2x - y + z = 5.$

Solution. The given system of equations is $\mathbf{Ax} = \mathbf{b}$, where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & -2 \\ 2 & -1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 10 \\ 7 \\ 5 \end{bmatrix}.$$

Now, $|\mathbf{A}| = \begin{vmatrix} 1 & 2 & 3 \\ 1 & 3 & -2 \\ 2 & -1 & 1 \end{vmatrix} = -30 \neq 0.$

That is, $\mathbf{A}$ is non-singular and hence $\mathbf{A}^{-1}$ exists.

$$\text{adj } \mathbf{A} = \begin{bmatrix} 1 & -5 & -13 \\ -5 & -5 & 5 \\ -7 & 5 & 1 \end{bmatrix}.$$

$$\text{Thus, } \mathbf{A}^{-1} = \frac{\text{adj } \mathbf{A}}{|\mathbf{A}|} = \frac{1}{-30}\begin{bmatrix} 1 & -5 & -13 \\ -5 & -5 & 5 \\ -7 & 5 & 1 \end{bmatrix}.$$

$$\text{Therefore, } \mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \frac{1}{-30}\begin{bmatrix} 1 & -5 & -13 \\ -5 & -5 & 5 \\ -7 & 5 & 1 \end{bmatrix}\begin{bmatrix} 10 \\ 7 \\ 5 \end{bmatrix} = \frac{1}{30}\begin{bmatrix} 90 \\ 60 \\ 30 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

Hence the required solution is $x = 3, y = 2, z = 1$.

Iteration Methods

If the system of equations has a large number of variables, then the direct methods are not much suitable. In this case, the approximate numerical methods are used to determine the variables of the system.

The approximate methods for solving system of linear equations make it possible to obtain the values of the roots of the system with the specified accuracy as the limit of the sequence of some vectors. The process of constructing such a sequence is known as the **iterative process.**

The efficiency of the application of approximate methods depends on the choice of the initial vector and the rate of convergence of the process.

The following two approximate methods are widely used to solve a system of linear equations:

(i) method of iteration (Jacobi's iteration method), and

(ii) Gauss-Seidal's iteration method.

Before presenting the iteration methods, some terms are introduced to analyse the methods.

Let $x_i^{(k)}$, $i = 1, 2, \ldots, n$ be the kth ($k = 1, 2, \ldots$) iterated value of the variable x_i and $\mathbf{x^{(k)}} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})^t$ be the solution vector obtained at the kth iteration.

The sequence $\{\mathbf{x^{(k)}}\}$, $k = 1, 2, \ldots$ is said to converge to a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$ if for each $i\ (= 1, 2, \ldots, n)$

$$x_i^{(k)} \longrightarrow x_i \quad \text{as} \quad k \longrightarrow \infty. \tag{30.10}$$

Let $\xi = (\xi_1, \xi_2, \ldots, \xi_n)^t$ be the exact solution of the system of linear equations. Then the error $\varepsilon_i^{(k)}$ in the ith variable x_i committed in the kth iteration is given by

$$\varepsilon_i^{(k)} = \xi_i - x_i^{(k)}. \tag{30.11}$$

The error vector $\boldsymbol{\varepsilon^{(k)}}$ at the kth iteration is then given by

$$\boldsymbol{\varepsilon^{(k)}} = (\varepsilon_1^{(k)}, \varepsilon_2^{(k)}, \ldots, \varepsilon_n^{(k)})^t. \tag{30.12}$$

The error difference $\mathbf{e^{(k)}}$ at two consecutive iterations is given by

$$\mathbf{e^{(k)}} = \mathbf{x^{(k+1)}} - \mathbf{x^{(k)}} = \boldsymbol{\varepsilon^{(k)}} - \boldsymbol{\varepsilon^{(k+1)}}, \tag{30.13}$$

where $e_i^{(k)} = x_i^{(k+1)} - x_i^{(k)}$.

An iteration method is said to be of order $p \geq 1$ if there exists a positive constant A such that for all k

$$\|\boldsymbol{\varepsilon^{(k+1)}}\| \leq A\|\boldsymbol{\varepsilon^{(k)}}\|^p. \tag{30.14}$$

30.6 Gauss-Seidal's Iteration Method

A simple modification of Jacobi's iteration sometimes give faster convergence. The modified method is known as Gauss-Seidal's iteration method.

Let us consider a system of n linear equations with n variables.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots &\cdots \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n. \end{aligned} \tag{30.15}$$

Assume that the diagonal coefficients a_{ii}, $i = 1, 2, \ldots, n$ are diagonally dominant. If this is not the case then the above system of equations are re-arranged in such a way that the above condition holds.

The equations (30.15) are rewritten in the following form:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n) \\ \cdots &\cdots \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1}). \end{aligned} \tag{30.16}$$

To solve these equations an initial approximation $x_2^{(0)}, x_3^{(0)}, \ldots, x_n^{(0)}$ for the variables $x_2, x_3, \ldots, x_n$ respectively is considered. Substituting these values to the above system and get the first approximate value of x_1, denoted by $x_1^{(1)}$. Now, substituting $x_1^{(1)}$ for x_1 and $x_3^{(0)}, x_4^{(0)}, \ldots, x_n^{(0)}$ for $x_3, x_4, \ldots, x_n$ respectively and we find $x_2^{(1)}$ from second equation of (30.16), the first approximate value of x_2. Then substituting $x_1^{(1)}, x_2^{(1)}, \ldots, x_{i-1}^{(1)}, x_{i+1}^{(0)}, \ldots, x_n^{(0)}$ for $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ to the ith equation of (30.16) respectively and obtain $x_i^{(1)}$, and so on.

If $x_i^{(k)}$, $i = 1, 2, \ldots, n$ be the kth approximate value of x_i, then the $(k+1)$th approximate value of $x_1, x_2, \ldots, x_n$ are given by

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}) \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_i^{(k+1)} &= \frac{1}{a_{ii}}(b_i - a_{i1}x_1^{(k+1)} - \cdots - a_{ii-1}x_{i-1}^{(k+1)} - a_{ii+1}x_{i+1}^{(k)} - \cdots - a_{nn-1}x_{n-1}^{(k)}) \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ x_n^{(k+1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \cdots - a_{nn-1}x_{n-1}^{(k+1)}). \\ & k = 0, 1, 2, \ldots. \end{aligned} \tag{30.17}$$

That is,

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\Big), \quad i = 1, 2, \ldots, n \text{ and } k = 0, 1, 2, \ldots.$$

The method is repeated until $|x_i^{(k+1)} - x_i^{(k)}| < \varepsilon$ for all $i = 1, 2, \ldots, n$, where $\varepsilon > 0$ is any pre-assigned number called the error tolerance. This method is called Gauss-Seidal's iteration method.

Example 30.6.1 Solve the following system of equations by Gauss-Seidal's iteration method, correct up to four decimal places.

$$27x + 6y - z = 54$$
$$6x + 15y + 2z = 72$$
$$x + y + 54z = 110$$

Solution. The iteration scheme is

$$x^{(k+1)} = \frac{1}{27}(54 - 6y^{(k)} + z^{(k)})$$
$$y^{(k+1)} = \frac{1}{15}(72 - 6x^{(k+1)} - 2z^{(k)})$$
$$z^{(k+1)} = \frac{1}{54}(110 - x^{(k+1)} - y^{(k+1)}).$$

Let $y = 0$, $z = 0$ be the initial solution. The successive iterations are shown below.

k	x	y	z
0	–	0	0
1	2.00000	4.00000	1.92593
2	1.18244	4.07023	1.93977
3	1.16735	4.07442	1.93997
4	1.16642	4.07477	1.93998
5	1.16635	4.07480	1.93998
6	1.16634	4.07480	1.93998

The solution correct up to four decimal places is $x = 1.1663$, $y = 4.0748$, $z = 1.9400$.

Note 30.6.1 This solution is achieved in eleven iterations using Gauss-Jacobi's method while only six iterations are used in Gauss-Seidal's method.

The sufficient condition for convergence of this method is that the diagonal elements of the coefficient matrix are diagonally dominant. This is justified in the following.

30.6.1 Convergence of Gauss-Seidal's method

Let $A = \max_i \left\{ \frac{1}{a_{ii}} \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right\}$ and let $A_i = \frac{1}{a_{ii}} \sum_{j=1}^{i-1} |a_{ij}|, \quad i = 1, 2, \ldots, n.$

It can easily be verified that $0 \leq A_i \leq A < 1$.

Then by Gauss-Jacobi's case

$$\begin{aligned} |\varepsilon_i^{(k+1)}| &\leq \frac{1}{|a_{ii}|} \left[\sum_{j<i} |a_{ij}| \, |\varepsilon_j^{(k+1)}| + \sum_{j>i} |a_{ij}| \, |\varepsilon_j^{(k)}| \right] \\ &\leq \frac{1}{|a_{ii}|} \left[\sum_{j<i} |a_{ij}| \, \|\varepsilon^{(k+1)}\| + \sum_{j>i} |a_{ij}| \, \|\varepsilon^{(k)}\| \right] \\ &\leq A_i \|\varepsilon^{(k+1)}\| + (A - A_i) \|\varepsilon^{(k)}\|. \end{aligned}$$

Then for some i,

$$\|\varepsilon^{(k+1)}\| \leq A_i \|\varepsilon^{(k+1)}\| + (A - A_i) \|\varepsilon^{(k)}\|$$

That is,

$$\|\varepsilon^{(k+1)}\| \leq \frac{A - A_i}{1 - A_i} \|\varepsilon^{(k)}\|.$$

Since $0 \leq A_i \leq A < 1$ then $\frac{A - A_i}{1 - A_i} \leq A$.

Therefore the above relation reduces to

$$\|\varepsilon^{(k+1)}\| \leq A \|\varepsilon^{(k)}\|. \tag{30.18}$$

This shows that the rate of convergence of Gauss-Seidal's iteration is also linear. The successive substitutions give

$$\|\varepsilon^{(k)}\| \leq A^k \|\varepsilon^{(0)}\|.$$

Now, if $A < 1$ then $\|\varepsilon^{(k)}\| \to 0$ as $k \to \infty$, i.e., the sequence $\{x^{(k)}\}$ is sure to converge when $A < 1$ i.e.,

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

In other words the sufficient condition for Gauss-Seidal's iteration is that the coefficient matrix is diagonally dominant. The absolute error at the $(k+1)$th iteration is given by

$$\|\varepsilon^{(k+1)}\| \leq \frac{A}{1 - A} \|e^{(k)}\| \quad \text{when} \quad A < 1,$$

as in previous section.

Note 30.6.2 Usually, the Gauss-Seidal's method converges rapidly than the Gauss-Jacobi's method. But, this is not always true. There are some examples in which the Gauss-Jacobi's method converges faster than the Gauss-Seidal's method.

Example 30.6.2 Solve the following system of equations by Gauss-Seidal's method correct to four significant figures: $3x + y + z = 3, 2x + y + 5z = 5, x + 4y + z = 2$.

Solution. It may be noted that the given system is not diagonally dominant, but, the rearranged system $3x + y + z = 3, x + 4y + z = 2, 2x + y + 5z = 5$ is diagonally dominant.

Then the Gauss-Seidal's iteration scheme is

$$x^{(k+1)} = \frac{1}{3}(3 - y^{(k)} - z^{(k)})$$
$$y^{(k+1)} = \frac{1}{4}(2 - x^{(k+1)} - z^{(k)})$$
$$z^{(k+1)} = \frac{1}{5}(5 - 2x^{(k+1)} - y^{(k+1)}).$$

Let $y = 0$, $z = 0$ be the initial values. The successive iterations are shown below.

k	x	y	z
0	–	0	0
1	1.00000	0.25000	0.55000
2	0.73333	0.17917	0.67083
3	0.71667	0.15313	0.68271
4	0.72139	0.14898	0.68165
5	0.72312	0.14881	0.68099
6	0.72340	0.14890	0.68086
7	0.72341	0.14893	0.68085

Therefore, the solution correct up to four significant figures is
$x = 0.7234$, $y = 0.1489$, $z = 0.6808$.

Example 30.6.3 Solve the following system of equations using Gauss-Seidal's method: $3x + y + 2z = 6, -x + 4y + 2z = 5, 2x + y + 4z = 7$.

Solution. The iteration scheme is

$$x^{(k+1)} = \frac{1}{3}(6 - y^{(k)} - 2z^{(k)})$$
$$y^{(k+1)} = \frac{1}{4}(5 + x^{(k+1)} - 2z^{(k)})$$
$$z^{(k+1)} = \frac{1}{4}(7 - 2x^{(k+1)} - y^{(k+1)}).$$

Let $y = 0, z = 0$ be the initial solution and other approximate values are shown below.

k	x	y	z
0	–	0	0
1	2.00000	1.75000	0.31250
2	1.20833	1.39583	0.79688
3	1.00347	1.10243	1.10243
4	0.89757	0.92328	1.07042
5	0.97866	0.95946	1.02081
6	0.99964	0.98951	1.00280
7	1.00163	0.99901	0.99943
8	1.00071	1.00046	0.99953
9	1.00016	1.00028	0.99985
10	1.00001	1.00008	0.99998

Therefore, the solution correct up to four decimal places is

$$x = 1.0000,\ y = 1.0000,\ z = 1.0000.$$

It may be noted that the given system is not diagonally dominant while the iteration scheme converges to the exact solution.

Another interesting problem is considered in the following. The system of equations

$$x_1 + x_2 = 2, x_1 - 3x_2 = 1$$

converges when the iteration scheme is taken as

$$x_1^{(k+1)} = 2 - x_2^{(k)}, \qquad x_2^{(k+1)} = \frac{1}{3}(-1 + x_1^{(k+1)}) \tag{30.19}$$

While the Gauss-Seidal's iteration method diverges when the iteration scheme is

$$x_1^{(k+1)} = 1 + 3x_2^{(k)}, \qquad x_2^{(k+1)} = 2 - x_1^{(k+1)} \tag{30.20}$$

(It may be noted that the system is not diagonally dominant).

The calculations for these two schemes are shown below and the behaviour of the solutions are shown in the figures 30.1 and 30.2.

For the scheme (30.19)

k	x_1	x_2
0	-	0
1	2	0.333
2	1.667	0.222
3	1.778	0.259
4	1.741	0.247
5	1.753	0.251
6	1.749	0.250
7	1.750	0.250

For the scheme (30.20)

k	x_1	x_2
0	-	0
1	1	1
2	4	-2
3	-5	7
4	22	-20
5	-59	61
6	184	-182
7	547	-545

The exact solution of the equations is

$$x_1 = 1.75,\ x_2 = 0.25.$$

This example shows that the condition 'diagonally dominant' is a sufficient condition, not necessary for Gauss-Seidal's iteration method.

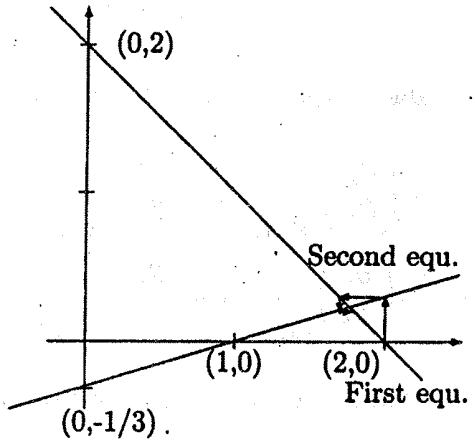


Figure 30.1: Illustration of Gauss-Seidal's method for the convergent scheme (30.19.)

30.7 Solution of Systems of Nonlinear Equations

In this section we shall discuss Newton-Raphson method to solve a system of nonlinear equations.

30.7.1 Newton-Raphson method

Let (x_0, y_0) be an initial guess to the root (ξ, η) of the equations

$$f(x, y) = 0 \text{ and } g(x, y) = 0. \tag{30.21}$$

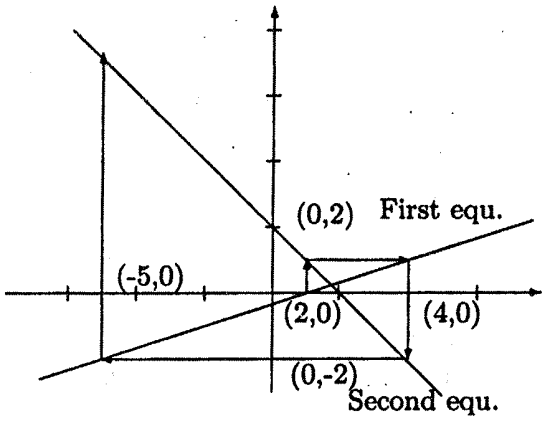


Figure 30.2: Illustration of Gauss-Seidal's method for the divergent scheme (30.20).

If $(x_0 + h, y_0 + k)$ is the root of the above system then

$$\begin{aligned} f(x_0 + h, y_0 + k) &= 0 \\ g(x_0 + h, y_0 + k) &= 0. \end{aligned} \tag{30.22}$$

Assume that f and g are differentiable. Expanding (30.22) by Taylor's series.

$$\begin{aligned} f(x_0, y_0) + h\left(\frac{\partial f}{\partial x}\right)_{(x_0,y_0)} + k\left(\frac{\partial f}{\partial y}\right)_{(x_0,y_0)} + \cdots &= 0 \\ g(x_0, y_0) + h\left(\frac{\partial g}{\partial x}\right)_{(x_0,y_0)} + k\left(\frac{\partial g}{\partial y}\right)_{(x_0,y_0)} + \cdots &= 0 \end{aligned} \tag{30.23}$$

Neglecting square and higher order terms, the above equations simplified as

$$\begin{aligned} h\frac{\partial f_0}{\partial x} + k\frac{\partial f_0}{\partial y} &= -f_0 \\ h\frac{\partial g_0}{\partial x} + k\frac{\partial g_0}{\partial y} &= -g_0 \end{aligned} \tag{30.24}$$

where $f_0 = f(x_0, y_0)$, $\dfrac{\partial f_0}{\partial x} = \left(\dfrac{\partial f}{\partial x}\right)_{(x_0,y_0)}$ etc.

The above system can be written as

$$\begin{bmatrix} \dfrac{\partial f_0}{\partial x} & \dfrac{\partial f_0}{\partial y} \\ \dfrac{\partial g_0}{\partial x} & \dfrac{\partial g_0}{\partial y} \end{bmatrix} \begin{bmatrix} h \\ k \end{bmatrix} = \begin{bmatrix} -f_0 \\ -g_0 \end{bmatrix} \text{ or } \begin{bmatrix} h \\ k \end{bmatrix} = J^{-1} \begin{bmatrix} -f_0 \\ -g_0 \end{bmatrix}.$$

Alternatively, h and k can be evaluated as

$$h = \frac{1}{J}\begin{vmatrix} -f_0 & \frac{\partial f_0}{\partial y} \\ -g_0 & \frac{\partial g_0}{\partial y} \end{vmatrix}, \quad k = \frac{1}{J}\begin{vmatrix} \frac{\partial f_0}{\partial x} & -f_0 \\ \frac{\partial g_0}{\partial x} & -g_0 \end{vmatrix}, \text{ where } J = \begin{vmatrix} \frac{\partial f_0}{\partial x} & \frac{\partial f_0}{\partial y} \\ \frac{\partial g_0}{\partial x} & \frac{\partial g_0}{\partial y} \end{vmatrix}. \tag{30.25}$$

Thus h and k are determined by one of the above two ways. Therefore, the new approximations are then given by

$$x_1 = x_0 + h, \qquad y_1 = y_0 + k. \tag{30.26}$$

The process is to be repeated until the roots are achieved to the desired accuracy. The general formula is $x_{n+1} = x_n + h, y_{n+1} = y_n + k$; h, k are evaluated at (x_n, y_n) instead at (x_0, y_0).

If the iteration converges (the condition is stated below) then the rate of convergence is quadratic.

Theorem 30.1 *Let (x_0, y_0) be an initial guess to a root (ξ, η) of the system $f(x, y) = 0, g(x, y) = 0$ in a closed neighbourhood R containing (ξ, η). If*

(i) f, g and their first order partial derivatives are continuous and bounded in R, and

(ii) $J \neq 0$ in R, then the sequence of approximation $x_{n+1} = x_n + h, y_{n+1} = y_n + k$, where h and k are given by (30.25), converges to the root (ξ, η).

Example 30.7.1 Use Newton-Raphson method to solve the system $x^2 - 2x - y + 0.5 = 0, x^2 + 4y^2 - 4 = 0$ with the starting value $(x_0, y_0) = (2.00, 0.25)$.

Solution. Let $f(x, y) = x^2 - 2x - y + 0.5$ and $g(x, y) = x^2 + 4y^2 - 4$.

$$\frac{\partial f}{\partial x} = 2x - 2, \frac{\partial f}{\partial y} = -1, \frac{\partial g}{\partial x} = 2x, \frac{\partial g}{\partial y} = 8y.$$

Therefore,

$$J = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x - 2 & -1 \\ 2x & 8y \end{bmatrix}, \qquad \begin{bmatrix} -f_0 \\ -g_0 \end{bmatrix} = \begin{bmatrix} -0.25 \\ -0.25 \end{bmatrix}.$$

At (x_0, y_0), $J_0 = \begin{bmatrix} 2 & -1 \\ 4 & 2 \end{bmatrix}$.

Therefore,

$$\begin{bmatrix} 2 & -1 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} h \\ k \end{bmatrix} = \begin{bmatrix} -0.25 \\ -0.25 \end{bmatrix}.$$

$$\text{or, } \begin{bmatrix} h \\ k \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 2 & 1 \\ -4 & 2 \end{bmatrix} \begin{bmatrix} -0.25 \\ -0.25 \end{bmatrix} = \begin{bmatrix} -0.09375 \\ 0.06250 \end{bmatrix}$$

Thus, $x_1 = x_0 + h = 2.00 - 0.09375 = 1.90625$,

$y_1 = y_0 + k = 0.25 + 0.0625 = 0.31250$.

At (x_1, y_1), $J_1 = \begin{bmatrix} 1.81250 & -1.00000 \\ 3.81250 & 2.50000 \end{bmatrix}$, $\begin{bmatrix} -f_1 \\ -g_1 \end{bmatrix} = \begin{bmatrix} -0.00879 \\ -0.02441 \end{bmatrix}$.

$$J_1 \begin{bmatrix} h \\ k \end{bmatrix} = \begin{bmatrix} -f_1 \\ -g_1 \end{bmatrix}$$

$$\text{or, } \begin{bmatrix} h \\ k \end{bmatrix} = \frac{1}{8.34375} \begin{bmatrix} 2.50000 & 1.00000 \\ -3.81250 & 1.81250 \end{bmatrix} \begin{bmatrix} -0.00879 \\ -0.02441 \end{bmatrix} = \begin{bmatrix} -0.00556 \\ -0.00129 \end{bmatrix}.$$

Therefore, $x_2 = x_1 + h = 1.90625 - 0.00556 = 1.90069$,

$y_2 = y_1 + k = 0.31250 - 0.00129 = 0.31121$.

At (x_2, y_2), $J_2 = \begin{bmatrix} 1.80138 & -1.00000 \\ 3.80138 & 2.48968 \end{bmatrix}$, $\begin{bmatrix} -f_2 \\ -g_2 \end{bmatrix} = \begin{bmatrix} -0.00003 \\ -0.00003 \end{bmatrix}$.

$$J_2 \begin{bmatrix} h \\ k \end{bmatrix} = \begin{bmatrix} -f_2 \\ -g_2 \end{bmatrix}$$

$$\text{or, } \begin{bmatrix} h \\ k \end{bmatrix} = \frac{1}{8.28624} \begin{bmatrix} 2.48968 & 1.00000 \\ -3.80138 & 1.80138 \end{bmatrix} \begin{bmatrix} -0.00003 \\ -0.00003 \end{bmatrix} = \begin{bmatrix} -0.00001 \\ 0.00001 \end{bmatrix}.$$

Hence, $x_3 = x_2 + h = 1.90069 - 0.00001 = 1.90068$,

$y_3 = y_2 + k = 0.31121 + 0.00001 = 0.31122$.

Thus, one root is $x = 1.9007, y = 0.3112$ correct up to four decimal places.

30.8 Comparison of Direct and Iterative Methods

The direct and iterative, both the methods have some advantages and also some disadvantages and a choice between them is based on the given system of equations.

(i) The direct method is applicable for all types of problems (when the coefficient determinant is not zero) where as iterative methods are useful only for particular types of problems.

(ii) The rounding errors may become large particularly for ill-conditioned systems while in iterative method the rounding error is small, since it is committed in the last iteration. Thus for ill-conditioned systems an iterative method is a good choice.

(iii) In each iteration, the computational effect is large in direct method (it is $2n^3/3$ for elimination method) and it is low in iteration method ($2n^2$ in Gauss-Jacobi's and Gauss-Seidal's methods).

(iv) Most of the direct methods are applied on the coefficient matrix and for this purpose, the entire matrix to be stored into primary memory of the computer. But, the iteration methods are applied

in a single equation at a time, and hence only a single equation is to be stored at a time in primary memory. Thus iterative methods are efficient then direct method with respect to space.

30.9 Module Summary

In this module, a systematic method (partial pivoting) is discussed to find the inverse of a matrix. The matrix inverse method is also studied here to solve a system of linear equations. An iterative method, called Gauss-Seidal iteration process is presented and computed its rate of convergence. The Newton-Raphson method is presented here to find the roots of a system of non-linear equations.

30.10 Self Assessment Questions

1. Find the inverse of the matrix

$$\begin{bmatrix} 11 & 3 & -1 \\ 2 & 5 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

using Gauss-Jordan method and solve the following system of equations.

$$\begin{aligned} 11x_1 + 3x_2 - x_3 &= 15 \\ 2x_1 + 5x_2 + 5x_3 &= -11 \\ x_1 + x_2 + x_3 &= 1. \end{aligned}$$

2. Find the inverses of the following matrices (using partial pivoting).

(i) $\begin{bmatrix} 0 & 1 & 2 \\ 3 & 5 & 1 \\ 6 & 8 & 9 \end{bmatrix}$ (ii) $\begin{bmatrix} -1 & 2 & 0 \\ 1 & 0 & 5 \\ 3 & 8 & 7 \end{bmatrix}$.

3. Solve the following equations by Gauss-Seidal's method, correct up to four significant figures:

$$\begin{aligned} 9x + 2y + 4z &= 20 \\ x + 10y + 4z &= 6 \\ 2x - 4y + 10z &= -15. \end{aligned}$$

4. Test if the following systems of equations are diagonally dominant and hence solve them using Gauss-Seidal's method.

(i)
$$\begin{aligned} 10x + 15y + 3z &= 14 \\ -30x + y + 5z &= 17 \\ x + y + 4z &= 3 \end{aligned}$$

(ii)
$$\begin{aligned} x + 3y + 4z &= 7 \\ 3x + 2y + 5z &= 10 \\ x - 5y + 7z &= 3. \end{aligned}$$

5. Solve the following systems of nonlinear equations using Newton-Raphson method
   (i) $3x^2 - 2y^2 - 1 = 0, x^2 - 2x + 2y - 8 = 0$ start with initial guess (2.5,3),
   (ii) $x^2 - x + y^2 + z^2 - 5 = 0, x^2 + y^2 - y + z^2 - 4 = 0, x^2 + y^2 + z^2 + z - 6 = 0$ start with $(-0.8, 0.2, 1.8)$ and $(1.2, 2.2, -0.2)$.

30.11 References

1. M.Pal, Numerical Analysis for Scientists and Engineers: Theory and C Programs, Narosa, 2007.
2. M.K. Jain, S.R.K. Iyengar and R.K. Jain, Numerical Methods for Scientific and Engineering Computation, New Age International (P) Limited, New Delhi, 1984.
3. J.H. Mathews, Numerical Methods for Mathematics, Science, and Engineering, 2nd ed., Prentice-Hall, Inc., N.J., U.S.A., 1992.
4. S.S.Sastry, Introductory Method of Numerical Analysis, PHI, 2006.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III Group-B

Module No.- 31

Numerical Analysis

(Eigenvalues and Eigenvectors of a Matrix)

Module Structure:

31.1 Introduction

In undergraduate class you have learnt the eigenvalue and eigenfunction of a matrix. In that class, the eigenvalues are computed from the corresponding characteristic function. That is, the eigenvalues are obtained by finding the roots of a polynomial of degree n for a matrix of order $n \times n$. But, this method is very tedious and needs more arithmetic effort.

Here will shall discuss some relatively better methods to find eigenvalues and eigenfunctions of a square matrix.

Let $\mathbf{A}$ be a square matrix of order n. Suppose $\mathbf{X}$ is a column matrix and λ is a scalar such that

$$\mathbf{AX} = \lambda\mathbf{X}. \tag{31.1}$$

The scalar λ is called an **eigenvalue** or **characteristic value** of the matrix $\mathbf{A}$ and $\mathbf{X}$ is called the corresponding **eigenvector**. The equation (31.1) can be written as $(\mathbf{A} - \lambda\mathbf{I})\mathbf{X} = 0$.

The equation

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \tag{31.2}$$

that is,

$$\begin{vmatrix} a_{11}-\lambda & a_{12} & a_{13}\cdots & a_{1n} \\ a_{21} & a_{22}-\lambda & a_{23}\cdots & a_{2n} \\ \cdots & \cdots & \cdots\cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3}\cdots & a_{nn}-\lambda \end{vmatrix} = 0 \tag{31.3}$$

is a polynomial in λ of degree n, called **characteristic equation** of the matrix $\mathbf{A}$. The roots $\lambda_i, i = 1, 2, \ldots, n$, of the equation (31.2) are the eigenvalues of $\mathbf{A}$. For each value of λ_i, there exists an $\mathbf{X}_i$ such that

$$\mathbf{AX}_i = \lambda_i\mathbf{X}_i. \tag{31.4}$$

The eigenvalues λ_i may be either distinct or repeated, they may be real or complex. If the matrix is real symmetric then all the eigenvalues are real. If the matrix is skew-symmetric then the eigenvalues are either zero or purely imaginary. Sometimes, the set of all eigenvalues, λ_i, of a matrix $\mathbf{A}$ is called the **spectrum** of $\mathbf{A}$ and the largest value of $|\lambda_i|$ is called the **spectral radius** of $\mathbf{A}$.

31.2 Objectives

Gone through this module the students will learn the following:

- Power method to find largest eigenvalue
- Jacobi's method to find eigenvalues and eigenvectors of a symmetric matrix.

31.3 Keywords

Eigenvalues, eigenfunctions, power method, Jacobi's method.

31.4 A Result

Theorem 31.1 (Shifting eigenvalues). *Suppose λ be an eigenvalue and $\mathbf{X}$ be its corresponding eigenvector of $\mathbf{A}$. If c is any constant, then $\lambda - c$ is an eigenvalue of the matrix $\mathbf{A} - c\mathbf{I}$ with same eigenvector $\mathbf{X}$.*

Let the characteristic polynomial of the matrix $\mathbf{A}$ be

$$det(\mathbf{A} - \lambda\mathbf{I}) = \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \cdots + c_n, \tag{31.5}$$

where

$-c_1 = \sum_{i=1}^{n} a_{ii} = Tr.\ \mathbf{A}$, which is the sum of all diagonal elements of $\mathbf{A}$, called the **trace.**

$c_2 = \sum_{i<j} \begin{vmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{vmatrix}$, is the sum of all principal minors of order two of $\mathbf{A}$,

$-c_3 = \sum_{i<j<k} \begin{vmatrix} a_{ii} & a_{ij} & a_{ik} \\ a_{ji} & a_{jj} & a_{jk} \\ a_{ki} & a_{kj} & a_{kk} \end{vmatrix}$, is the sum of all principal minors of order three of $\mathbf{A}$,

and finally $(-1)^n c_n = det\mathbf{A}$, is the determinant of the matrix $\mathbf{A}$.

31.5 Eigenvalues for Arbitrary Matrices

Several methods are available to determine the eigenvalues and eigenvectors of a matrix. Here, we discuss Power methods to find the largest eigenvalue (in magnitude) of an arbitrary matrix.

31.5.1 Power method

Power method is generally used to find the eigenvalue, largest in magnitude, (sometimes called first eigenvalue) of a matrix $\mathbf{A}$. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be all eigenvalues of the matrix $\mathbf{A}$. We assume that

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots > |\lambda_n|,$$

i.e., λ_1 is largest in magnitude and $\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_n}$ be the eigenvectors corresponding to the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ respectively. The method is applicable if the matrix $\mathbf{A}$ has n independent eigenvectors. Then

any vector $\mathbf{X}$ in the (vector) space of eigenvectors $\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_n}$ can be written as

$$\mathbf{X} = c_1\mathbf{X_1} + c_2\mathbf{X_2} + \cdots + c_n\mathbf{X_n}. \tag{31.6}$$

Multiplying this relation by $\mathbf{A}$ and using the results $\mathbf{AX_1} = \lambda_1\mathbf{X_1}$, $\mathbf{AX_2} = \lambda_2\mathbf{X_2}$, ..., $\mathbf{AX_n} = \lambda_n\mathbf{X_n}$, we obtain

$$\begin{aligned}\mathbf{AX} &= c_1\lambda_1\mathbf{X_1} + c_2\lambda_2\mathbf{X_2} + \cdots + c_n\lambda_n\mathbf{X_n} \\ &= \lambda_1\left[c_1\mathbf{X_1} + c_2\left(\frac{\lambda_2}{\lambda_1}\right)\mathbf{X_2} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)\mathbf{X_n}\right].\end{aligned} \tag{31.7}$$

Again, multiplying this relation by $\mathbf{A}, \mathbf{A^2}, \ldots, \mathbf{A^k}$ successively and obtain the relations

$$\mathbf{A^2X} = \lambda_1^2\left[c_1\mathbf{X_1} + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^2\mathbf{X_2} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^2\mathbf{X_n}\right]. \tag{31.8}$$

$$\vdots \qquad\qquad \vdots$$

$$\mathbf{A^kX} = \lambda_1^k\left[c_1\mathbf{X_1} + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{X_2} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{X_n}\right]. \tag{31.9}$$

$$\mathbf{A^{k+1}X} = \lambda_1^{k+1}\left[c_1\mathbf{X_1} + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^{k+1}\mathbf{X_2} + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^{k+1}\mathbf{X_n}\right]. \tag{31.10}$$

When $k \to \infty$, then right hand sides of (31.9) and (31.10) tend to $\lambda_1^k c_1\mathbf{X_1}$ and $\lambda_1^{k+1}c_1\mathbf{X_1}$, since $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$ for $i = 2, \ldots, n$. Thus for $k \to \infty$, $\mathbf{A^kX} = \lambda_1^k c_1\mathbf{X_1}$ and $\mathbf{A^{k+1}X} = \lambda_1^{k+1}c_1\mathbf{X_1}$. That is, for $k \to \infty$, $\lambda_1\mathbf{A^k}\lambda = \mathbf{A^{k+1}X}$. It is well known that two vectors are equal if their corresponding components are same. That is,

$$\lambda_1 = \lim_{k\to\infty} \frac{\left(\mathbf{A^{k+1}X}\right)_r}{\left(\mathbf{A^kX}\right)_r}, \; r = 1, 2, \ldots, n. \tag{31.11}$$

The symbol $(\mathbf{A^kX})_r$ denotes the rth component of the vector $\mathbf{A^kX}$.

If $|\lambda_2| \ll |\lambda_1|$, then the term within square bracket of (31.10) tend faster to $c_1\mathbf{X_1}$, i.e., the rate of convergence is fast.

To reduce the round off error, the method is carried out by normalizing (reducing the largest element to unity) the eigenvector at each iteration. Let $\mathbf{X_0}$ be a non-null initial (arbitrary) vector (non-orthogonal to $\mathbf{X_1}$) and we compute

$$\begin{aligned}\mathbf{Y_{i+1}} &= \mathbf{AX_i} \\ \mathbf{X_{i+1}} &= \frac{\mathbf{Y_{i+1}}}{\lambda^{(i+1)}}, \qquad \text{for } i = 0, 1, 2, \ldots.\end{aligned} \tag{31.12}$$

where $\lambda^{(i+1)}$ is the largest element in magnitude of $\mathbf{Y_{i+1}}$ and it is the $(i+1)$th approximate value of λ_1. Then

$$\lambda_1 = \lim_{k\to\infty} \frac{(\mathbf{Y_{k+1}})_r}{(\mathbf{X_k})_r}, \qquad r = 1, 2, \ldots, n. \tag{31.13}$$

and $\mathbf{X_{k+1}}$ is the eigenvector corresponding to the eigenvalue λ_1.

Note 31.5.1 The initial vector $\mathbf{X_0}$ is usually chosen as $\mathbf{X_0} = (1, 1, \cdots, 1)^T$. But, if the initial vector $\mathbf{X_0}$ is poor, then the formula (31.13) does not give λ_1, i.e., the limit of the ratio $\frac{(Y_{k+1})_r}{(X_k)_r}$ may not exist. If this situation occurs, then the initial vector must be changed.

Note 31.5.2 The power method is also used to find the least eigenvalue of a matrix $\mathbf{A}$. If $\mathbf{X}$ is the eigenvector corresponding to the eigenvalue λ then $\mathbf{AX} = \lambda\mathbf{X}$. If $\mathbf{A}$ is non-singular then $\mathbf{A^{-1}}$ exist. Therefore, $\mathbf{A^{-1}(AX)} = \lambda\mathbf{A^{-1}X}$ or, $\mathbf{A^{-1}X} = \frac{1}{\lambda}\mathbf{X}$. This means that if λ is an eigenvalue of $\mathbf{A}$ then $\frac{1}{\lambda}$ is an eigenvalue of $\mathbf{A^{-1}}$ and the same eigenvector $\mathbf{X}$ corresponds to the eigenvalue $1/\lambda$ of the matrix $\mathbf{A^{-1}}$. Thus, if λ is largest (in magnitude) eigenvalue of $\mathbf{A}$ then $1/\lambda$ is the least eigenvalue of $\mathbf{A^{-1}}$.

Note 31.5.3 We observed that the coefficient X_j in (31.9) goes to zero in proposition to $(\lambda_j/\lambda_1)^k$ and that the speed of convergence is governed by the terms $(\lambda_2/\lambda_1)^k$. Consequently, the rate of convergence is linear.

Example 31.5.1 Find the largest eigenvalue in magnitude and corresponding eigenvector of the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 2 \\ -1 & 0 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

Solution. Let the initial vector be $\mathbf{X_0} = (1, 1, 1)^T$.

The first iteration is given by

$$\mathbf{Y_1} = \mathbf{AX_0} = \begin{bmatrix} 1 & 3 & 2 \\ -1 & 0 & 2 \\ 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 12 \end{bmatrix}.$$

Therefore $\lambda^{(1)} = 12$ and $\mathbf{X_1} = \dfrac{\mathbf{Y_1}}{12} = \begin{bmatrix} 0.50000 \\ 0.08333 \\ 1.0000 \end{bmatrix}$.

$$\mathbf{Y_2} = \mathbf{AX_1} = \begin{bmatrix} 1 & 3 & 2 \\ -1 & 0 & 2 \\ 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 0.50000 \\ 0.08333 \\ 1.00000 \end{bmatrix} = \begin{bmatrix} 2.75 \\ 1.5 \\ 6.83333 \end{bmatrix}$$

$$\lambda^{(2)} = 6.83333, \quad \mathbf{X_2} = \frac{\mathbf{Y_2}}{6.83333} = \begin{bmatrix} 0.40244 \\ 0.21951 \\ 1.0000 \end{bmatrix}.$$

$$\mathbf{Y_3} = \mathbf{AX_2} = \begin{bmatrix} 1 & 3 & 2 \\ -1 & 0 & 2 \\ 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 0.40244 \\ 0.21951 \\ 1.00000 \end{bmatrix} = \begin{bmatrix} 3.06098 \\ 1.59756 \\ 7.08537 \end{bmatrix}$$

$$\lambda^{(3)} = 7.08537, \quad \mathbf{X_3} = \begin{bmatrix} 0.43201 \\ 0.22547 \\ 1.00000 \end{bmatrix}.$$

$$\mathbf{Y_4} = \begin{bmatrix} 3.10843 \\ 1.56799 \\ 7.19793 \end{bmatrix}, \quad \mathbf{X_4} = \begin{bmatrix} 0.43185 \\ 0.21784 \\ 1. \end{bmatrix}, \quad \lambda^{(4)} = 7.19793.$$

$$\mathbf{Y_5} = \begin{bmatrix} 3.08691 \\ 1.56950 \\ 7.16672 \end{bmatrix}, \quad \mathbf{X_5} = \begin{bmatrix} 0.43050 \\ 0.21880 \\ 1. \end{bmatrix}, \quad \lambda^{(5)} = 7.16691.$$

$$\mathbf{Y_6} = \begin{bmatrix} 3.08691 \\ 1.56950 \\ 7.16672 \end{bmatrix}, \quad \mathbf{X_6} = \begin{bmatrix} 0.43073 \\ 0.21900 \\ 1. \end{bmatrix}, \quad \lambda^{(6)} = 7.16672.$$

$$\mathbf{Y_7} = \begin{bmatrix} 3.08772 \\ 1.56927 \\ 7.16818 \end{bmatrix}, \quad \mathbf{X_7} = \begin{bmatrix} 0.43075 \\ 0.21892 \\ 1.0 \end{bmatrix}, \quad \lambda^{(7)} = 7.16818.$$

$$\mathbf{Y_8} = \begin{bmatrix} 3.08752 \\ 1.56925 \\ 7.16795 \end{bmatrix}, \quad \mathbf{X_8} = \begin{bmatrix} 0.43074 \\ 0.21893 \\ 1.0 \end{bmatrix}, \quad \lambda^{(8)} = 7.16795.$$

$$\mathbf{Y_9} = \begin{bmatrix} 3.08752 \\ 1.56926 \\ 7.16792 \end{bmatrix}, \quad \mathbf{X_9} = \begin{bmatrix} 0.43074 \\ 0.21893 \\ 1.0 \end{bmatrix}, \quad \lambda^{(9)} = 7.16792.$$

$$\mathbf{Y_{10}} = \begin{bmatrix} 3.08753 \\ 1.56926 \\ 7.16794 \end{bmatrix}, \quad \mathbf{X_{10}} = \begin{bmatrix} 0.43074 \\ 0.21893 \\ 1.0 \end{bmatrix}, \quad \lambda^{(10)} = 7.16794.$$

The required largest eigenvalue is 7.1679 correct up to four decimal places and the corresponding eigenvector is

$$\begin{bmatrix} 0.43074 \\ 0.21893 \\ 1.00000 \end{bmatrix}.$$

31.5.2 Power method for least eigenvalue

It is mentioned earlier that if λ is the largest eigenvalue of $\mathbf{A}$ then $1/\lambda$ is the smallest eigenvalue of $\mathbf{A}$ and $1/\lambda$ can be obtained by finding the largest eigenvalue of $\mathbf{A^{-1}}$. But, computation of $\mathbf{A^{-1}}$ is a labourious

process, so a simple process is needed. One simple method is introduced here.

If the largest eigenvalue (in magnitude) λ_1, of an $n \times n$ matrix $\mathbf{A}$ is known then the smallest magnitude eigenvalue can be computed by using power method for the matrix $\mathbf{B} = (\mathbf{A} - \lambda_1\mathbf{I})$ instead of $\mathbf{A}$. The eigenvalues of the matrix $\mathbf{B}$ are $\lambda_i' = (\lambda_i - \lambda_1)$ (called shifting eigenvalues), $i = 1, 2, \ldots, n$, where λ_i are the eigenvalues of $\mathbf{A}$. Obviously, λ_n' is the largest magnitude eigenvalue of $\mathbf{B}$. Again, if $\mathbf{X_n}$ is the corresponding eigenvector, then $\mathbf{BX_n} = \lambda_n'\mathbf{X_n}$ or, $(\mathbf{A} - \lambda_1\mathbf{I})\mathbf{X_n} = (\lambda_n - \lambda_1)\mathbf{X_n}$, i.e., $\mathbf{AX_n} = \lambda_n\mathbf{X_n}$. Hence $\mathbf{X_n}$ is also the eigenvector of $\mathbf{A}$, corresponding to the eigenvalue λ_n.

31.6 Eigenvalues for Symmetric Matrices

The methods discussed earlier are also applicable for symmetric matrices, but, due to the special properties of symmetric matrices several efficient methods viz., Jacobi, Givens and Householder, etc. are available. Here, we shall discuss Jacobi's method only.

31.6.1 Jacobi's method

This method is widely used to find the eigenvalues and eigenvectors of a real symmetric matrix. Since all the eigenvalues of $\mathbf{A}$ are real and there exist a real orthogonal matrix $\mathbf{S}$ such that $\mathbf{S^{-1}AS}$ is a diagonal matrix $\mathbf{D}$. As $\mathbf{D}$ and $\mathbf{A}$ are similar, the diagonal elements of $\mathbf{D}$ are the eigenvalues of $\mathbf{A}$. But, the computation of the matrix $\mathbf{S}$ is not a simple task. It is obtained by a series of orthogonal transformations $\mathbf{S_1}, \mathbf{S_2}, \ldots, \mathbf{S_n}, \ldots$ as discussed below.

Let $|a_{ij}|$ be the largest element among the off-diagonal elements of $\mathbf{A}$. Now, we construct an orthogonal matrix $\mathbf{S_1}$ whose elements are defined as

$$s_{ij} = -\sin\theta,\ s_{ji} = \sin\theta,\ s_{ii} = \cos\theta,\ s_{jj} = \cos\theta, \tag{31.14}$$

all other off-diagonal elements are zero and all other diagonal elements are unity. Thus $\mathbf{S_1}$ is of the form

$$\mathbf{S_1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \tag{31.15}$$

where $\cos\theta$, $-\sin\theta$, $\sin\theta$ and $\cos\theta$ are at the positions $(i,i), (i,j), (j,i)$ and (j,j) respectively.

Let $\mathbf{A_1} = \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix}$ be a sub-matrix of $\mathbf{A}$ formed by the elements a_{ii}, a_{ij}, a_{ji} and a_{jj}. To reduce $\mathbf{A_1}$ to a diagonal matrix, an orthogonal transformation is applied which is defined as $\overline{\mathbf{S_1}} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, where θ is an unknown quantity and it will be selected in such a way that $\mathbf{A_1}$ becomes diagonal.

Now,

$$\begin{aligned} &\overline{\mathbf{S_1}}^{-1}\mathbf{A_1}\overline{\mathbf{S_1}} \\ &= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} a_{ii}\cos^2\theta + a_{ij}\sin 2\theta + a_{jj}\sin^2\theta & (a_{jj} - a_{ii})\sin\theta\cos\theta + a_{ij}\cos 2\theta \\ (a_{jj} - a_{ii})\sin\theta\cos\theta + a_{ij}\cos 2\theta & a_{ii}\sin^2\theta - a_{ij}\sin 2\theta + a_{jj}\cos^2\theta \end{bmatrix}. \end{aligned}$$

This matrix becomes a diagonal matrix if $(a_{jj} - a_{ii})\sin\theta\cos\theta + a_{ij}\cos 2\theta = 0$,
That is , if

$$\tan 2\theta = \frac{2a_{ij}}{a_{ii} - a_{jj}}. \tag{31.16}$$

The value of θ can be obtained from the following relation.

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2a_{ij}}{a_{ii} - a_{jj}}\right). \tag{31.17}$$

This expression gives four values of θ, but, to get smallest rotation, θ should lie in $-\pi/4 \le \theta \le \pi/4$. The equation (31.17) is valid for all i, j if $a_{ii} \neq a_{jj}$. If $a_{ii} = a_{jj}$ then

$$\theta = \begin{cases} \frac{\pi}{4}, & \text{if } a_{ij} > 0 \\ -\frac{\pi}{4}, & \text{if } a_{ij} < 0. \end{cases} \tag{31.18}$$

Thus the off-diagonal elements s_{ij} and s_{ji} of $\overline{\mathbf{S_1}}^{-1}\mathbf{A_1}\overline{\mathbf{S_1}}$ vanish and the diagonal elements are modified. The first diagonal matrix is obtained by computing $\mathbf{D_1} = \mathbf{S_1^{-1}A_1S_1}$. In the next step largest off-diagonal (in magnitude) element is selected from the matrix $\mathbf{D_1}$ and the above process is repeated to generate another orthogonal matrix $\mathbf{S_2}$ to compute $\mathbf{D_2}$. That is,

$$\mathbf{D_2} = \mathbf{S_2^{-1}D_1S_2} = \mathbf{S_2^{-1}(S_1^{-1}AS_1)S_2} = \mathbf{(S_1S_2)^{-1}A(S_1S_2)}.$$

In this way, a series of two-dimensional rotations are performed. At the end of k transformations the matrix $\mathbf{D_k}$ is obtained as

$$\begin{aligned} \mathbf{D_k} &= \mathbf{S_k^{-1}S_{k-1}^{-1}\cdots S_1^{-1}AS_1S_2\cdots S_{k-1}S_k} \\ &= \mathbf{(S_1S_2\cdots S_k)^{-1}A(S_1S_2\cdots S_k)} \\ &= \mathbf{S^{-1}AS} \end{aligned} \tag{31.19}$$

where $S = S_1 S_2 \cdots S_k$.

As $k \to \infty$, D_k tends to a diagonal matrix. The diagonal elements of D_k are the eigenvalues and the columns of S are the corresponding eigenvectors.

The method has a drawback. The elements those are transferred to zero during diagonalisation may not necessarily remain zero during subsequent rotations. The value of θ must be verified for its accuracy by checking whether $|\sin^2\theta + \cos^2\theta - 1|$ is sufficiently small.

Note 31.6.1 It may be noted that for orthogonal matrix S, $S^{-1} = S^T$.

Note 31.6.2 It can be shown that the minimum number of rotations required to transform a real symmetric matrix into a diagonal matrix is $n(n-1)/2$.

Example 31.6.1 Find the eigenvalues and eigenvectors of the symmetric matrix $A = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}$ using Jacobi's method.

Solution. The largest off-diagonal element is 2 at (1,2), (1,3) and (2, 3) positions. The rotational angle θ is given by $\tan 2\theta = \dfrac{2a_{12}}{a_{11} - a_{22}} = \dfrac{4}{0} = \infty$ i.e., $\theta = \dfrac{\pi}{4}$.

Thus the orthogonal matrix S_1 is

$$S_1 = \begin{bmatrix} \cos\pi/4 & -\sin\pi/4 & 0 \\ \sin\pi/4 & \cos\pi/4 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Then the first rotation yields

$$D_1 = S_1^{-1} A S_1 = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 0 & 4/\sqrt{2} \\ 0 & -1 & 0 \\ 4/\sqrt{2} & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 & 2.82843 \\ 0 & -1 & 0 \\ 2.82843 & 0 & 1 \end{bmatrix}.$$

The largest off-diagonal element of D_1 is now 2.82843 situated at (1,3) position and hence the rotational angle is

$$\theta = \frac{1}{2}\tan^{-1}\frac{2a_{13}}{a_{11} - a_{33}} = 0.61548.$$

The second orthogonal matrix S_2 is

$$S_2 = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} = \begin{bmatrix} 0.81650 & 0 & -0.57735 \\ 0 & 1 & 0 \\ 0.57735 & 0 & 0.81650 \end{bmatrix}.$$

Then second rotation gives

$$\mathbf{D_2} = \mathbf{S_2^{-1}D_1S_2}$$

$$= \begin{bmatrix} 0.81650 & 0 & 0.57735 \\ 0 & 1 & 0 \\ -0.577351 & 0 & 0.81650 \end{bmatrix} \begin{bmatrix} 3 & 0 & 2.82843 \\ 0 & -1 & 0 \\ 2.82843 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.81650 & 0 & -0.57735 \\ 0 & 1 & 0 \\ 0.57735 & 0 & 0.81650 \end{bmatrix}.$$

$$= \begin{bmatrix} 5 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Thus $\mathbf{D_2}$ becomes a diagonal matrix and hence the eigenvalues are $5, -1, -1$.
The eigenvectors are the columns of $\mathbf{S}$, where

$$\mathbf{S} = \mathbf{S_1S_2} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.81650 & 0 & -0.57735 \\ 0 & 1 & 0 \\ 0.57735 & 0 & 0.81650 \end{bmatrix}$$

$$= \begin{bmatrix} 0.57735 & -0.70711 & -0.40825 \\ 0.57735 & 0.70711 & -0.40825 \\ 0.57735 & 0 & 0.81650 \end{bmatrix}.$$

Hence the eigenvalues are $5, -1, -1$ and the corresponding eigenvectors are $(0.57735, 0.57735, 0.57735)^T$, $(-0.70711, -0.70711, 0)^T$, $(-0.40825, -0.40825, 0.81650)^T$ respectively.

Note that the eigenvectors are normalized and two independent eigenvectors (last two vectors) for the eigenvalue -1 are obtained by this method.

In this problem, two rotations are used to convert A into a diagonal matrix. But, this does not happen in general. The following example shows that at least six rotations are needed to diagonalise a symmetric matrix.

Example 31.6.2 Find all the eigenvalues and eigenvectors of the matrix $\begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ by Jacobi's method.

Solution. Let $\mathbf{A} = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

It is a real symmetric matrix and the Jacobi's method is applicable.

The largest off-diagonal element is at $a_{12} = a_{21}$ and it is 3.

Then $\tan 2\theta = \dfrac{2a_{12}}{a_{11} - a_{22}} = \dfrac{6}{0} = \infty$, and this gives $\theta = \dfrac{\pi}{4}$.

Thus the orthogonal matrix $\mathbf{S_1}$ is

$$\mathbf{S_1} = \begin{bmatrix} \cos\pi/4 & -\sin\pi/4 & 0 \\ \sin\pi/4 & \cos\pi/4 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The first rotation gives

$$\mathbf{D_1} = \mathbf{S_1^{-1}AS_1} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 2 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 0 & 3/\sqrt{2} \\ 0 & -1 & 1/\sqrt{2} \\ 3/\sqrt{2} & 1/\sqrt{2} & 1 \end{bmatrix}.$$

The largest off-diagonal element of $\mathbf{D_1}$ is $3/\sqrt{2}$, situated at $(1,3)$ position. Then

$\tan 2\theta = \dfrac{2a_{13}}{a_{11} - a_{33}} = \dfrac{6/\sqrt{2}}{5-1} = 1.06066.$ or, $\theta = \frac{1}{2}\tan^{-1}(0.69883) = 0.407413$

So, the next orthogonal matrix $\mathbf{S_2}$ is $\mathbf{S_2} = \begin{bmatrix} 0.91815 & 0 & -0.39624 \\ 0 & 1 & 0 \\ 0.39624 & 0 & 0.91815 \end{bmatrix}$.

$$\mathbf{D_2} = \mathbf{S_2^{-1}D_1S_2}$$

$$= \begin{bmatrix} 0.91815 & 0 & 0.39624 \\ 0 & 1 & 0 \\ -0.39624 & 0 & 0.91815 \end{bmatrix} \begin{bmatrix} 5 & 0 & 2.12132 \\ 0 & -1 & 0.70711 \\ 2.12132 & 0.70711 & 1 \end{bmatrix} \begin{bmatrix} 0.91815 & 0 & -0.39624 \\ 0 & 1 & 0 \\ 0.39624 & 0 & 0.91815 \end{bmatrix}$$

$$= \begin{bmatrix} 5.91548 & 0.28018 & 0 \\ 0.28018 & -1.0 & 0.64923 \\ 0 & 0.64923 & 0.08452 \end{bmatrix}.$$

The largest off-diagonal element of $\mathbf{D_2}$ is 0.64923, present at the position $(2,3)$. Then

$$\tan 2\theta = \frac{2a_{23}}{a_{22} - a_{33}} = -1.19727 \quad \text{or,} \quad \theta = \frac{1}{2}\tan^{-1}(-1.19727) = -0.43747.$$

Therefore, $\mathbf{S_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.90583 & 0.42365 \\ 0 & -0.42365 & 0.90583 \end{bmatrix}$.

$$\mathbf{D_3} = \mathbf{S_3^{-1}D_2S_3} = \begin{bmatrix} 5.91548 & 0.25379 & 0.11870 \\ 0.25379 & -1.30364 & 0 \\ 0.11870 & 0 & 0.38816 \end{bmatrix}.$$

Again, largest off-diagonal element is 0.25379, located at $(1,2)$ position.

Therefore, $\theta = \frac{1}{2}\tan^{-1}\frac{2a_{12}}{a_{11}-a_{22}} = 0.03510.$

$$\mathbf{S_4} = \begin{bmatrix} 0.99938 & -0.03509 & 0 \\ 0.03509 & 0.99938 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$\mathbf{D_4} = \mathbf{S_4^{-1}D_3S_4} = \begin{bmatrix} 5.92439 & 0 & 0.11863 \\ 0 & -1.31255 & -0.00417 \\ 0.11863 & -0.00417 & 0.38816 \end{bmatrix}.$$

Here largest off-diagonal element is 0.11863 at $(1,3)$ position. In this case

$$\theta = \frac{1}{2}\tan^{-1}\frac{2a_{13}}{a_{11}-a_{33}} = 0.02141.$$

$$\mathbf{S_5} = \begin{bmatrix} 0.99977 & 0 & -0.02141 \\ 0 & 1 & 0 \\ 0.02141 & 0 & 0.99977 \end{bmatrix}.$$

$$\mathbf{D_5} = \mathbf{S_5^{-1}D_4S_5} = \begin{bmatrix} 5.92693 & -0.00009 & 0 \\ -0.00009 & -1.31255 & -0.00417 \\ 0 & -0.00417 & 0.38562 \end{bmatrix}.$$

The largest off-diagonal element in magnitude is -0.00417 situated at $(2,3)$ position. Then

$$\theta = \frac{1}{2}\tan^{-1}\frac{2a_{23}}{a_{22}-a_{33}} = 0.00246.$$

$$\mathbf{S_6} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -0.00246 \\ 0 & 0.00246 & 1 \end{bmatrix}.$$

$$\mathbf{D_6} = \mathbf{S_6^{-1}D_5S_6} = \begin{bmatrix} 5.92693 & -0.00009 & 0 \\ -0.00009 & -1.31256 & 0 \\ 0 & 0 & 0.38563 \end{bmatrix}.$$

This matrix is almost diagonal and hence the eigenvalues are 5.9269, -1.3126 and 0.3856 correct up to four decimal places.

The eigenvectors are the columns of

$$\mathbf{S} = \mathbf{S_1S_2}\ldots\mathbf{S_6} = \begin{bmatrix} 0.61852 & -0.54567 & -0.56540 \\ 0.67629 & 0.73604 & 0.02948 \\ 0.40006 & -0.40061 & 0.82430 \end{bmatrix}$$

That is, the eigenvectors corresponding to the eigenvalues 5.9269, -1.3126 and 0.3856 are respectively $(0.61825, 0.67629, 0.40006)^T$,
$(-0.54567, 0.73604, -0.40061)^T$ and $(-0.56540, 0.02948, 0.82430)^T$.

31.7 Module Summary

Here, we have introduced Power method to find the largest (in magnitude) eigenvalue and the corresponding eigenfunction of an arbitrary matrix. This method is also available to find least eigenvalue of an arbitrary matrix. We also discussed Jacobi's method to find eigenvalues and eigenfunctions of a real symmetric matrix.

31.8 Self Assessment Questions

1. Use power method to find the largest (in magnitude) eigenvalues of the following matrix $\begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$.

2. Use power method to find the largest (in magnitude) eigenvalues of the following matrix $\begin{bmatrix} 4 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{bmatrix}$.

3. Use power method to find the largest (in magnitude) eigenvalues of the following matrix $\begin{bmatrix} 2 & -1 & 2 \\ 5 & -3 & 3 \\ -1 & 0 & -2 \end{bmatrix}$

4. Use power method to find the largest (in magnitude) eigenvalues of the following matrix $\begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 1 & 1 \end{bmatrix}$.

5. Use Jacobi's method to find the eigenvalues of the following matrix $\begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 3 \end{bmatrix}$.

6. Use Jacobi's method to find the eigenvalues of the following matrix $\begin{bmatrix} -2 & -2 & 6 \\ -2 & 5 & 4 \\ 6 & 4 & 1 \end{bmatrix}$.

7. Use Jacobi's method to find the eigenvalues of the following matrix $\begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}$.

31.9 References

1. M.Pal, Numerical Analysis for Scientists and Engineers: Theory and C Programs, Narosa, 2007.
2. M.K. Jain, S.R.K. Iyengar and R.K. Jain, Numerical Methods for Scientific and Engineering Computation, New Age International (P) Limited, New Delhi, 1984.
3. J.H. Mathews, Numerical Methods for Mathematics, Science, and Engineering, 2nd ed., Prentice-Hall, Inc., N.J., U.S.A., 1992.
4. S.S.Sastry, Introductory Method of Numerical Analysis, PHI, 2006.

M.Sc. Course
in
Applied Mathematics with Oceanology
and
Computer Programming

PART-I

Paper-III Group-B

Module No.- 32

Numerical Analysis

(Differential Equations)

Module Structure:

32.1 Introduction

You have learn, the numerical solution of ordinary differential equation by Euler's method and modified Euler's method in undergraduate class. Here, you will learn some other methods to solve ordinary as well as partial differential equations.

Let us consider the general first order differential equation

$$\frac{dy}{dx} = f(x, y) \tag{32.1}$$

with initial condition

$$y(x_0) = y_0. \tag{32.2}$$

The solution of a differential equation can be done in one of the following two forms:

(i) A series solution for y in terms of powers of x. Then the values of y can be determined by substituting $x = x_0, x_1, \ldots, x_n$.

(ii) A set of tabulated values of y for $x = x_0, x_1, \ldots, x_n$ with spacing h.

In case (i), the solution of the differential equation is computed in terms of x, and the values of y are calculated from this solution, where as, in case (ii), the solution of the differential equation is obtained by applying the method repeatedly for each value of x $(= x_1, x_2, \ldots, x_n)$.

32.2 Objectives

Gone through this module the students will learn the following:

- Runge-Kutta method
- Milne's predictor-corrector method

- Stability of numerical solution
- Finite difference scheme
- Solution of heat equation
- Crank-Nicolson method
- Solution of wave equation.

32.3 Keywords

Runge-Kutta method, predictor-corrector method, single-step method, multi-step method, stability, heat equation, wave equation.

32.4 A Few Definitions

If the differential equation is of nth order then its general solution contains n arbitrary constants. To find the values of these constants, n conditions are needed. The problems in which all the conditions are specified at the **initial** point only, are called **initial value problems (IVPs)**. The problems of order two or more and for which the conditions are given at two or more points are called **boundary value problems (BVPs)**.

If the value of y_{i+1} depends only on the value of y_i then the method is called **single-step method** and if two or more values are required to evaluate the value of y_{i+1} then the method is known as **two-step** or **multistep method**.

Again, if the value of y_{i+1} depends only on the values of y_i, h and $f(x_i, y_i)$ then the method used to determine y_{i+1} is called **explicit method**, otherwise the method is called **implicit method.**

The order of a finite difference approximation of a differential equation is the rate at which the global error of the finite difference solution approaches to zero as the size of the grid spacing (h) approaches zero. When applied to a differential equation with a bounded solution, a finite difference equation is **stable** if it produces a bounded solution and is **unstable** if it produces an unbounded solution. It is quite possible that the numerical solution to a differential equation grows unbounded even though its exact solution is well behaved. Of course, there are cases for which the exact solution may be unbounded, but, for our discussion of stability we concentrate only on the cases in which the exact solution is bounded. In stability analysis, conditions are deduced in terms of the step size h for which the numerical solution remains bounded. In this connection, the numerical methods are of three classes.

(i) **Stable numerical scheme:** The numerical solution does not blow up with choice of step size.

(ii) **Unstable numerical scheme:** The numerical solution blows up with any choice of step size.

(iii) **Conditionally stable numerical scheme:** Numerical solution remains bounded with certain choices of step size.

A finite difference method is **convergent** if the solution of the finite difference equation approaches to a limit as the size of the grid spacing tends to zero. But, there is no guarantee, in general, that this limit corresponds to the exact solution of the differential equation.

32.5 Runge-Kutta Methods

The Euler's method is less efficient in practical problems because if h is not sufficiently small then this method gives inaccurate result.

The Runge-Kutta methods give more accurate result. One advantage of this method is it requires only the value of the function at some selected points on the subinterval and it is stable, and easy to program.

The Runge-Kutta methods perform several function evaluations at each step and avoid the computation of higher order derivatives. These methods can be constructed for any order, i.e., second, third, fourth, fifth, etc. The fourth-order Runge-Kutta method is more popular. These methods are single-step explicit methods.

32.5.1 Second-order Runge-Kutta method

The modified Euler's method to compute y_1 is

$$y_1 = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_1, y_1^{(0)})]. \tag{32.3}$$

If $y_1^{(0)} = y_0 + hf(x_0, y_0)$ is substituted in (32.3) then

$$y_1 = y_0 + \frac{h}{2}[f(x_0, y_0) + f(x_0 + h, y_0 + hf(x_0, y_0))].$$

Setting,

$$k_1 = hf(x_0, y_0) \text{ and}$$
$$k_2 = hf(x_0 + h, y_0 + hf(x_0, y_0)) = hf(x_0 + h, y_0 + k_1). \tag{32.4}$$

Then equation (32.3) becomes

$$y_1 = y_0 + \frac{1}{2}(k_1 + k_2). \tag{32.5}$$

This is known as second-order Runge-Kutta formula. The local truncation error of this formula is of $O(h^3)$.

General derivation

Assume that the solution is of the form

$$y_1 = y_0 + ak_1 + bk_2, \tag{32.6}$$

where

$k_1 = hf(x_0, y_0)$ and

$k_2 = hf(x_0 + \alpha h, y_0 + \beta k_1), a, b, \alpha$ and β are constants.

By Taylor's series,

$$\begin{aligned}
y_1 = y(x_0 + h) &= y_0 + hy_0' + \frac{h^2}{2}y_0'' + \frac{h^3}{6}y_0''' + \cdots \\
&= y_0 + hf(x_0, y_0) + \frac{h^2}{2}\left[\left(\frac{\partial f}{\partial x}\right)_{(x_0,y_0)} + f(x_0, y_0)\left(\frac{\partial f}{\partial y}\right)_{(x_0,y_0)}\right] + O(h^3) \\
&\quad \left[\text{As } \frac{df}{dx} = \frac{\partial f}{\partial x} + f(x, y)\frac{\partial f}{\partial y}\right] \\
k_2 &= hf(x_0 + \alpha h, y_0 + \beta k_1) \\
&= h\left[f(x_0, y_0) + \alpha h\left(\frac{\partial f}{\partial x}\right)_{(x_0,y_0)} + \beta k_1\left(\frac{\partial f}{\partial y}\right)_{(x_0,y_0)} + O(h^2)\right] \\
&= hf(x_0, y_0) + \alpha h^2\left(\frac{\partial f}{\partial x}\right)_{(x_0,y_0)} + \beta h^2 f(x_0, y_0)\left(\frac{\partial f}{\partial y}\right)_{(x_0,y_0)} + O(h^3).
\end{aligned}$$

Then the equation (32.6) becomes

$$\begin{aligned}
&y_0 + hf(x_0, y_0) + \frac{h^2}{2}[f_x(x_0, y_0) + f(x_0, y_0)f_y(x_0, y_0)] + O(h^3) \\
&\quad = y_0 + (a + b)hf(x_0, y_0) + bh^2[\alpha f_x(x_0, y_0) + \beta f(x_0, y_0)f_y(x_0, y_0)] + O(h^3).
\end{aligned}$$

The coefficients of f, f_x and f_y are compared and the following equations are obtained.

$$a + b = 1, \quad b\alpha = \frac{1}{2} \text{ and } b\beta = \frac{1}{2}. \tag{32.7}$$

Obviously, $\alpha = \beta$ and if α is assigned any value arbitrarily, then the remaining parameters can be determined uniquely. However, usually the parameters are chosen as $\alpha = \beta = 1$, then $a = b = \frac{1}{2}$.

Thus the formula is

$$y_1 = y_0 + \frac{1}{2}(k_1 + k_2) + O(h^3), \tag{32.8}$$

where $k_1 = hf(x_0, y_0)$ and $k_2 = hf(x_0 + h, y_0 + k_1)$.

It follows that there are several second-order Runge-Kutta formulae and (32.8) is just one among such formulae.

32.5.2 Fourth-order Runge-Kutta Method

The fourth-order Runge-Kutta formula is

$$y_1 = y_0 + ak_1 + bk_2 + ck_3 + dk_4, \tag{32.9}$$

where

$$\begin{aligned} k_1 &= hf(x_0, y_0) \\ k_2 &= hf(x_0 + \alpha_0 h, y_0 + \beta_0 k_1) \\ k_3 &= hf(x_0 + \alpha_1 h, y_0 + \beta_1 k_1 + \gamma_1 k_2) \\ \text{and} \quad k_4 &= hf(x_0 + \alpha_2 h, y_0 + \beta_2 k_1 + \gamma_2 k_2 + \delta_1 k_3). \end{aligned} \tag{32.10}$$

The parameters $a, b, c, d, \alpha_0, \beta_0, \alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2, \delta_1$ are to be determined by expanding both sides of (32.9) by Taylor's series retaining the terms up to and including those containing h^4. The choice of the parameters is arbitrary and depending on the choice of parameters, several fourth-order Runge-Kutta formulae can be generated.

The above functions are expanded by using Taylor's series method retaining terms up to fourth order. The local truncation error is $O(h^5)$. Runge and Kutta obtained the following system of equations.

$$\left.\begin{aligned} \beta_0 &= \alpha_0 \\ \beta_1 + \gamma_1 &= \alpha_1 \\ \beta_2 + \gamma_2 + \delta_1 &= \alpha_2 \\ a + b + c + d &= 1 \\ b\alpha_0 + c\alpha_1 + d\alpha_2 &= 1/2 \\ b\alpha_0^2 + c\alpha_1^2 + d\alpha_2^2 &= 1/3 \\ b\alpha_0^3 + c\alpha_1^3 + d\alpha_2^3 &= 1/4 \\ c\alpha_0\gamma_1 + d(\alpha_0\gamma_2 + \alpha_1\delta_1) &= 1/6 \\ c\alpha_0\alpha_1\gamma_1 + d\alpha_2(\alpha_0\gamma_2 + \alpha_1\delta_1) &= 1/8 \\ c\alpha_0^2\gamma_1 + d(\alpha_0^2\gamma_2 + \alpha_1^2\delta_1) &= 1/12 \\ d\alpha_0\gamma_1\delta_1 &= 1/24. \end{aligned}\right\} \tag{32.11}$$

The above system of equations contain 11 equations and 13 unknowns. Two more conditions are required to solve the system. But, these two conditions are taken arbitrarily. The most common choice is

$$\alpha_0 = \frac{1}{2}, \qquad \beta_1 = 0.$$

With these choice, the solution of the system (32.11) is

$$\alpha_1 = \frac{1}{2}, \alpha_2 = 1, \beta_0 = \frac{1}{2}, \gamma_1 = \frac{1}{2}, \beta_2 = 0, \gamma_2 = 0, \delta_1 = 1,$$
$$a = \frac{1}{6}, b = c = \frac{1}{3}, d = \frac{1}{6}. \tag{32.12}$$

The values of these variables are substituted in (32.9) and (32.10) and the fourth-order Runge-Kutta method is obtained as

$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{32.13}$$

where

$$k_1 = hf(x_0, y_0)$$
$$k_2 = hf(x_0 + h/2, y_0 + k_1/2)$$
$$k_3 = hf(x_0 + h/2, y_0 + k_2/2)$$
$$k_4 = hf(x_0 + h, y_0 + k_3).$$

Starting with the initial point (x_0, y_0), one can generate the sequence of solutions at $x_1, x_2, \ldots$ using the formula

$$y_{i+1} = y_i + \frac{1}{6}(k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}) \tag{32.14}$$

where

$$k_1^{(i)} = hf(x_i, y_i)$$
$$k_2^{(i)} = hf(x_i + h/2, y_i + k_1^{(i)}/2)$$
$$k_3^{(i)} = hf(x_i + h/2, y_i + k_2^{(i)}/2)$$
$$k_4^{(i)} = hf(x_i + h, y_i + k_3^{(i)}).$$

Example 32.5.1 Given $y' = y^2 - x^2$, where $y(0) = 2$. Find $y(0.1)$ and $y(0.2)$ by second-order Runge-Kutta method.

Solution. Here $h = 0.1, x_0 = 0, y_0 = 2, f(x, y) = y^2 - x^2$.

Then

$$k_1 = hf(x_0, y_0) = 0.1(2^2 - 0^2) = 0.4000$$
$$k_2 = hf(x_0 + h, y_0 + k_1) = 0.1 \times f(0 + 0.1, 2 + 0.4000)$$
$$= 0.1 \times (2.4^2 - 0.1^2) = 0.5750.$$

Therefore, $y_1 = y_0 + \frac{1}{2}(k_1 + k_2) = 2 + \frac{1}{2}(0.4000 + 0.5750) = 2.4875$, *i.e.*, $y(0.1) = 2.4875$.
To determine $y_2 = y(0.2)$, let $x_1 = 0.1$ and $y_1 = 2.4875$.

$$\begin{aligned} k_1 &= hf(x_1, y_1) = 0.1 \times f(0.1, 2.4875) = 0.1 \times (2.4875^2 - 0.1^2) \\ &= 0.6178. \\ k_2 &= hf(x_1 + h, y_1 + k_1) = 0.1 \times f(0.2, 2.4875 + 0.6178) \\ &= 0.1 \times f(0.2, 3.1053) = 0.1 \times (3.1053^2 - 0.2^2) = 0.9603. \end{aligned}$$

Therefore, $y_2 = y_1 + \frac{1}{2}(k_1 + k_2) = 2.4875 + \frac{1}{2}(0.6178 + 0.9603) = 3.2766$.
Hence, $y(0.2) = 3.2766$.

Example 32.5.2 Given $y' = x^2 + y^2$ with $x = 0, y = 1$. Find $y(0.1)$ by fourth-order Runge-Kutta method.

Solution. Here $h = 0.1, x_0 = 0, y_0 = 1, f(x, y) = x^2 + y^2$.

$$\begin{aligned} k_1 &= hf(x_0, y_0) = 0.1 \times (0^2 + 1^2) = 0.1000. \\ k_2 &= hf(x_0 + h/2, y_0 + k_1/2) = 0.1 \times f(0.05, 1.05) \\ &= 0.1 \times (0.05^2 + 1.05^2) = 0.1105. \\ k_3 &= hf(x_0 + h/2, y_0 + k_2/2) = 0.1 \times f(0.05, 1.0553) \\ &= 0.1 \times (0.05^2 + 1.0553^2) = 0.1116. \\ k_4 &= hf(x_0 + h, y_0 + k_3) = 0.1 \times f(0.1, 1.1116) \\ &= 0.1 \times (0.1^2 + 1.1116^2) = 0.1246. \end{aligned}$$

Therefore,

$$\begin{aligned} y_1 &= y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= 1 + \frac{1}{6}(0.1000 + 2 \times 0.1105 + 2 \times 0.1116 + 0.1246) \\ &= 1.1115. \end{aligned}$$

Note 32.5.1 The Runge-Kutta method gives better result, though, it has some disadvantages. This method uses numerous calculations of function to find y_{i+1}. When the function $f(x, y)$ has a complicated analytic form then the Runge-Kutta method is very laborious.

Error

The fourth-order Runge-Kutta formula is

$$y_1 = y_0 + \frac{(h/2)}{3}\left[k_1 + \frac{4(k_2 + k_3)}{2} + k_4\right].$$

This is similar to the Simpson's formula with step size $h/2$, so the local truncation error of this formula is $-\frac{h^5}{2880}y^{iv}(c_1)$, i.e., of $O(h^5)$ and after n steps, the accumulated error is

$$-\sum_{i=1}^{n}\frac{h^5}{2880}y^{iv}(c_i) = -\frac{x_n - x_0}{5760}y^{iv}(c)h^4 = O(h^4).$$

32.5.3 Runge-Kutta method for a pair of equations

The Runge-Kutta methods may also be used to solve a pair of first order differential equations.

Consider a pair of first-order differential equations

$$\begin{aligned}\frac{dy}{dx} &= f(x,y,z)\\ \frac{dz}{dx} &= g(x,y,z)\end{aligned} \tag{32.15}$$

with initial conditions

$$x = x_0,\ \ y(x_0) = y_0,\ \ z(x_0) = z_0. \tag{32.16}$$

Then the values of y_i and z_i at x_i are obtained by the formulae

$$\begin{aligned}y_{i+1} &= y_i + \frac{1}{6}[k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}],\\ z_{i+1} &= z_i + \frac{1}{6}[l_1^{(i)} + 2l_2^{(i)} + 2l_3^{(i)} + l_4^{(i)}],\end{aligned} \tag{32.17}$$

where

$$\begin{aligned}k_1^{(i)} &= hf(x_i, y_i, z_i)\\ l_1^{(i)} &= hg(x_i, y_i, z_i)\\ k_2^{(i)} &= hf(x_i + h/2, y_i + k_1^{(i)}/2, z_i + l_1^{(i)}/2)\\ l_2^{(i)} &= hg(x_i + h/2, y_i + k_1^{(i)}/2, z_i + l_1^{(i)}/2)\\ k_3^{(i)} &= hf(x_i + h/2, y_i + k_2^{(i)}/2, z_i + l_2^{(i)}/2)\\ l_3^{(i)} &= hg(x_i + h/2, y_i + k_2^{(i)}/2, z_i + l_2^{(i)}/2)\\ k_4^{(i)} &= hf(x_i + h, y_i + k_3^{(i)}, z_i + l_3^{(i)})\\ l_4^{(i)} &= hg(x_i + h, y_i + k_3^{(i)}, z_i + l_3^{(i)}).\end{aligned} \tag{32.18}$$

Example 32.5.3 Solve the following pair of differential equations $\frac{dy}{dx} = \frac{x+y}{z}$ and $\frac{dz}{dx} = xy + z$ with initial conditions $x_0 = 0.5, y_0 = 1.5, z_0 = 1$ for $x = 0.6$.

Solution. Let $h = 0.1$ and calculate the values of k_1, k_2, k_3, k_4; l_1, l_2, l_3, l_4.

Here $f(x,y) = \frac{x+y}{z}$, $g(x,y) = xy + z$.

$$k_1 = hf(x_0, y_0, z_0) = 0.1 \times \frac{0.5 + 1.5}{1} = 0.2$$
$$l_1 = hg(x_0, y_0, z_0) = 0.1 \times (0.5 \times 1.5 + 1) = 0.175$$
$$k_2 = hf(x_0 + h/2, y_0 + k_1/2, z_0 + l_1/2) = 0.1 \times \frac{0.55 + 1.6}{1.0875} = 0.197701$$
$$l_2 = hg(x_0 + h/2, y_0 + k_1/2, z_0 + l_1/2) = 0.1 \times (0.55 \times 1.6 + 1.0875)$$
$$= 0.19675$$
$$k_3 = hf(x_0 + h/2, y_0 + k_2/2, z_0 + l_2/2) = 0.1 \times \frac{0.55 + 1.59885}{1.098375} = 0.195639$$
$$l_3 = hg(x_0 + h/2, y_0 + k_2/2, z_0 + l_2/2) = 0.1 \times (0.55 \times 1.59885 + 1.098375)$$
$$= 0.197774$$
$$k_4 = hf(x_0 + h, y_0 + k_3, z_0 + l_3) = 0.1 \times \frac{0.6 + 1.695639}{1.197774} = 0.191659$$
$$l_4 = hg(x_0 + h, y_0 + k_3, z_0 + l_3) = 0.1 \times (0.6 \times 1.695639 + 1.197774)$$
$$= 0.221516.$$

Hence,

$$y(0.6) = y_1 = y_0 + \frac{1}{6}[k_1 + 2(k_2 + k_3) + k_4]$$
$$= 1.5 + \frac{1}{6}[0.2 + 2(0.197701 + 0.195639) + 0.191659] = 1.696390.$$
$$z(0.6) = z_1 = z_0 + \frac{1}{6}[l_1 + 2(l_2 + l_3) + l_4]$$
$$= 1.0 + \frac{1}{6}[0.175 + 2(0.19675 + 0.197774) + 0.221516] = 1.197594.$$

32.6 Predictor-Corrector Methods

The methods, viz., Taylor's series, Picard, Euler's and Runge-Kutta are **single-step methods** as these methods use only one previous values to compute the successive values, i.e., y_i is used to compute y_{i+1}. Certain efficient methods are available which need some more values to compute the successive values. These methods are called **multistep methods**. General k-step method needs $y_{i-\overline{k-1}}, y_{i-\overline{k-2}}, \cdots, y_{i-1}$ and y_i to compute y_{i+1}. The predictor-corrector method is a combination of two formulae. The first formula (called predictor) finds an approximate value of y_i and the second formula (called corrector) improves this value. The commonly used predictor-corrector multistep methods are due to Adams-Bashforth-Moulton and Milne-Simpson. Here, we discusse only Milne's method.

32.6.1 Milne's method

Another popular predictor-corrector formula is **Milne's method** which is also known as **Milne-Simpson method**. The differential equation $y' = f(x, y)$ with $y(x_0) = y_0$ is integrated between x_{i-3} and x_{i+1} and find

$$y_{i+1} = y_{i-3} + \int_{x_{i-3}}^{x_{i+1}} f(x, y)\, dx. \tag{32.19}$$

Now, the function $f(x,y)$ is replaced by Newton's forward difference formula in the form

$$f(x,y) = f_{i-3} + u\Delta f_{i-3} + \frac{u(u-1)}{2!}\Delta^2 f_{i-3} + \frac{u(u-1)(u-2)}{3!}\Delta^3 f_{i-3}, \tag{32.20}$$

where $u = \dfrac{x - x_{i-3}}{h}$.

The value of $f(x,y)$ is substituted from (32.20) to (32.19) and find

$$\begin{aligned} y_{i+1} &= y_{i-3} + h\int_0^4 \left[f_{i-3} + u\Delta f_{i-3} + \frac{u^2-u}{2}\Delta^2 f_{i-3} + \frac{u^3-3u^2+2u}{6}\Delta^3 f_{i-3}\right] du \\ &= y_{i-3} + h\left[4f_{i-3} + 8\Delta f_{i-3} + \frac{20}{3}\Delta^2 f_{i-3} + \frac{8}{3}\Delta^3 f_{i-3}\right] \\ &= y_{i-3} + \frac{4h}{3}[2f_{i-2} - f_{i-1} + 2f_i]. \end{aligned}$$

Thus the Milne's predictor formula is

$$y^p_{i+1} = y_{i-3} + \frac{4h}{3}[2f_{i-2} - f_{i-1} + 2f_i]. \tag{32.21}$$

The corrector formula is developed in a similar way. The value of y^p_{i+1} will now be used. Again, the given differential equation is integrated between x_{i-1} and x_{i+1} and the function $f(x,y)$ is replaced by the Newton's formula (32.20). Then

$$\begin{aligned} y_{i+1} &= y_{i-1} + \int_{x_{i-1}}^{x_{i+1}} \left[f_{i-1} + u\Delta f_{i-1} + \frac{u(u-1)}{2}\Delta^2 f_{i-1}\right] dx \\ &= y_{i-1} + h\int_0^2 \left[f_{i-1} + u\Delta f_{i-1} + \frac{u^2-u}{2}\Delta^2 f_{i-1}\right] du \\ &= y_{i-1} + h\left[2f_{i-1} + 2\Delta f_{i-1} + \frac{1}{3}\Delta^2 f_{i-1}\right] \\ &= y_{i-1} + \frac{h}{3}[f_{i-1} + 4f_i + f_{i+1}]. \end{aligned}$$

This formula is known as corrector formula and it is denoted by y^c_{i+1}. That is,

$$y^c_{i+1} = y_{i-1} + \frac{h}{3}[f(x_{i-1}, y_{i-1}) + 4f(x_i, y_i) + f(x_{i+1}, y^p_{i+1})]. \tag{32.22}$$

When y^p_{i+1} is computed using the formula (32.21), formula (32.22) can be used iteratively to obtain the value of y_{i+1} to the desired accuracy.

Example 32.6.1 Find the value of $y(0.20)$ for the initial value problem

$$\frac{dy}{dx} = y^2 \sin x \text{ with } y(0) = 1$$

using Milne's predictor-corrector method, taking $h = 0.05$.

Solution. Let $f(x,y) = y^2 \sin x$, $x_0 = 0, y_0 = 1, h = 0.05$.

Fourth-order Runge-Kutta method is used to compute the starting values y_1, y_2 and y_3.

i	x_i	y_i	k_1	k_2	k_3	k_4	y_{i+1}
0	0.00	1.000000	0.000000	0.001250	0.001251	0.002505	1.001251
1	0.05	1.001251	0.002505	0.003765	0.003770	0.005042	1.005021
2	0.10	1.005021	0.005042	0.006328	0.006336	0.007643	1.011356

The predictor value

$$\begin{aligned} y_4^p &= y_0 + \frac{4h}{3}[2f(x_1,y_1) - f(x_2,y_2) + 2f(x_3,y_3)] \\ &= 1 + \frac{4 \times 0.05}{3}[2f(0.05, 1.001251) - f(0.10, 1.005021) + 2f(0.15, 1.011356)] \\ &= 1 + \frac{4 \times 0.05}{3}[2 \times 0.0501043 - 0.1008385 + 2 \times 0.1528516] \\ &= 1.0203380 \end{aligned}$$

The corrector value is

$$\begin{aligned} y_4^c &= y_2 + \frac{h}{2}[f(x_2,y_2) + 4f(x_3,y_3) + f(x_4,y_4^p)] \\ &= 1,005021 + \frac{0.05}{2}[0.1008385 + 4 \times 0.1528516 + 0.2068326] \\ &= 1.0203390. \end{aligned}$$

Again, the corrector value y_4^c is calculated by using the formula

$$\begin{aligned} y_4^c &= y_2 + \frac{h}{2}[f(x_2,y_2) + 4f(x_3,y_3) + f(x_4,y_4^c)] \\ &= 1.005021 + \frac{0.05}{2}[0.1008385 + 4 \times 0.1528516 + 0.2068392] \\ &= 1.0203390. \end{aligned}$$

Since these two values are same, the required solution is
$y_4 = y(0.20) = 1.0203390$ correct up to seven decimal places.

Error

The local truncation error for prediction formula is

$$\frac{28}{90} y^v(c_{i+1}) h^5 = O(h^5)$$

and that of the corrector formula is $-\frac{1}{90} y^v(d_{i+1}) h^5 = O(h^5)$.

Note 32.6.1 Milne's predictor-corrector method is widely used formula. In this method there is a scope to improve the value of y by repeated use of corrector formula. So it gives more accurate result. But, this method needs the starting values y_1, y_2, y_3 to obtain y_4. These values may be obtained from any single-step method, such as Taylor's series, Euler's, Runge-Kutta or any similar method.

32.7 Finite Element Method

Finite element method (FEM) is widely used technique to solve many engineering problems. Here, a very brief introduction is presented to solve a BVP by using this method. The detailed discussion of FEM is beyond the scope of this book.

The main idea of this method is – the whole interval (of integration) is divided into a finite number of subintervals called **element** and over each element the continuous function is approximated by a suitable piecewise polynomial. This approximated problem is solved by Rayleigh-Ritz or Galerkin methods.

Let us consider the functional

$$J[y(x)] = \int_a^b F(x, y(x), y'(x))\, dx \tag{32.23}$$

subject to the boundary conditions

$$y(a) = y_a, \quad y(b) = y_b. \tag{32.24}$$

We assume that F is differentiable. The curve $y = y(x)$ which extremizing J under the boundary condition (32.24) is a solution of the **Euler equation**

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\left(\frac{\partial F}{\partial y'}\right) = 0. \tag{32.25}$$

The Euler's equation (32.25) has many solutions but, for a given boundary condition, it gives a unique solution.

Let us consider the boundary value problem

$$-\frac{d}{dx}[p(x)y'(x)] + q(x)y(x) = r(x) \tag{32.26}$$

with boundary condition (32.24). It can easily be verified that the variational form of (32.26) is given by

$$J[y(x)] = \frac{1}{2}\int_a^b [p(x)\,\{y'(x)\}^2 + q(x)\,\{y(x)\}^2 - 2r(x)\,y(x)]\, dx. \tag{32.27}$$

The main steps to solve a BVP using FEM are given below.

Step 1. Discretization of the interval

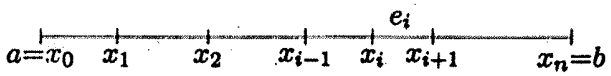


Figure 32.1: Division of interval into elements.

The interval $[a, b]$ is divided into a finite number of subintervals, called elements, of unequal length. Let $x_0, x_1, \ldots, x_n$, $a = x_0 < x_1 < \cdots < x_n = b$, be the division points, called the **nodes**. Let $e_i = [x_i, x_{i+1}]$ be the ith element of length $h_i = x_{i+1} - x_i$.

Step 2. Variational formulation of BVP over the element e_i

The variational form of (32.26) is given by

$$J[y(x)] = \frac{1}{2}\int_a^b [p(y')^2 + qy^2 - 2ry]\, dx. \tag{32.28}$$

Let J_i be the functional (called **element functional**) over the element $e_i = [x_i, x_{i+1}]$, $i = 0, 1, 2, \ldots, n-1$. Then

$$J_i = \frac{1}{2}\int_{x_i}^{x_{i+1}} \left[p\left(\frac{dy^{(i)}}{dx}\right)^2 + q\left(y^{(i)}\right)^2 - 2ry^{(i)}\right] dx, \tag{32.29}$$

where $y^{(i)}$ is the value of y over the element e_i and it is zero outside the element e_i. x_i, x_{i+1} are the end nodes of the element e_i and let $\phi^{(i)} = [y(x_i)\ \ y(x_{i+1})]^T$.

Thus the functional J over the whole interval $[a, b]$ is the sum of the n functionals J_i, that is,

$$J[y] = \sum_{i=0}^{n-1} J_i. \tag{32.30}$$

The element functional J_i will be extremum with respect to $\phi^{(i)}$ if

$$\frac{\partial J_i}{\partial \phi^{(i)}} = 0. \tag{32.31}$$

This gives the required finite element equation for the element e_i.

Step 3. Rayleigh-Ritz finite element approximation over e_i

The function $y(x)$ may be approximated over the element e_i by linear Lagrange's interpolation polynomial as

$$y^{(i)}(x) = L_i(x)y_i + L_{i+1}(x)y_{i+1} \tag{32.32}$$

where

$$y_i = y(x_i) \text{ and } L_i(x) = \frac{x_{i+1} - x}{x_{i+1} - x_i},\ L_{i+1}(x) = \frac{x - x_i}{x_{i+1} - x_i}. \tag{32.33}$$

The function $L_i(x)$ and $L_{i+1}(x)$ are called **shape functions.**

The equation (32.32) can be written as

$$y^{(i)}(x) = \begin{bmatrix} L_i(x) & L_{i+1} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} = \mathbf{L}^{(i)}\phi^{(i)} \tag{32.34}$$

where $\mathbf{L}^{(i)} = \begin{bmatrix} L_i & L_{i+1} \end{bmatrix}$ and $\phi^{(i)} = \begin{bmatrix} y_i & y_{i+1} \end{bmatrix}^T$.

It may be noted that

$$L_i(x_j) = \begin{cases} 0, & i \neq j \\ 1, & i = j. \end{cases}$$

From (32.33), it is easy to obtain

$$\frac{\partial L_i}{\partial x} = -\frac{1}{x_{i+1} - x_i} = -\frac{1}{h_i} \text{ and } \frac{\partial L_{i+1}}{\partial x} = \frac{1}{x_{i+1} - x_i} = \frac{1}{h_i}. \tag{32.35}$$

The value of $y^{(i)}(x)$ is substituted from (32.34) to (32.29) and obtain

$$\begin{aligned} J_i = \frac{1}{2}\int_{x_i}^{x_{i+1}} \Bigg[& p(x)\left\{ \begin{bmatrix} L_i' & L_{i+1}' \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\}^2 + q(x)\left\{ \begin{bmatrix} L_i & L_{i+1} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\}^2 \\ & - r(x)\left\{ \begin{bmatrix} L_i & L_{i+1} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\}^2 \Bigg] dx, \end{aligned}$$

where prime denotes differentiation with respect to x.

To extremize J_i, differentiating it with respect to y_i and y_{i+1} as

$$\begin{aligned} \frac{\partial J_i}{\partial y_i} = \int_{x_i}^{x_{i+1}} \Bigg[& p(x)L_i'\left\{ \begin{bmatrix} L_i' & L_{i+1}' \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\} \\ & + q(x)L_i\left\{ \begin{bmatrix} L_i & L_{i+1} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\} - r(x)L_i \Bigg] dx = 0. \end{aligned} \tag{32.36}$$

$$\begin{aligned} \text{and } \frac{\partial J_i}{\partial y_{i+1}} = \int_{x_i}^{x_{i+1}} \Bigg[& p(x)L_{i+1}'\left\{ \begin{bmatrix} L_i' & L_{i+1}' \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\} \\ & + q(x)L_{i+1}\left\{ \begin{bmatrix} L_i & L_{i+1} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \right\} - r(x)L_{i+1} \Bigg] dx = 0. \end{aligned} \tag{32.37}$$

These two equations can be written in matrix form as

$$\begin{aligned} \int_{x_i}^{x_{i+1}} & \left\{ p(x) \begin{bmatrix} L_i'L_i' & L_i'L_{i+1}' \\ L_{i+1}'L_i' & L_{i+1}'L_{i+1}' \end{bmatrix} + q(x) \begin{bmatrix} L_iL_i & L_iL_{i+1} \\ L_{i+1}L_i & L_{i+1}L_{i+1} \end{bmatrix} \right\} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} dx \\ & - \int_{x_i}^{x_{i+1}} r(x) \begin{bmatrix} L_i \\ L_{i+1} \end{bmatrix} dx = 0. \end{aligned}$$

This gives

$$\mathbf{A}^{(i)}\phi^{(i)} - \mathbf{b}^{(i)} = \mathbf{0}, \tag{32.38}$$

where

$$\mathbf{A}^{(i)} = \int_{x_i}^{x_{i+1}} \left\{ p(x) \begin{bmatrix} L_i'L_i' & L_i'L_{i+1}' \\ L_{i+1}'L_i' & L_{i+1}'L_{i+1}' \end{bmatrix} + q(x) \begin{bmatrix} L_iL_i & L_iL_{i+1} \\ L_{i+1}L_i & L_{i+1}L_{i+1} \end{bmatrix} \right\} dx,$$

$$\mathbf{b}^{(i)} = \int_{x_i}^{x_{i+1}} r(x) \begin{bmatrix} L_i \\ L_{i+1} \end{bmatrix} dx, \quad \text{and} \quad \phi^{(i)} = \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix}. \tag{32.39}$$

Step 4. Assembly of element equations

Let the elements of the matrix $\mathbf{A}^{(i)}$ be taken as

$$\mathbf{A}^{(i)} = \begin{bmatrix} a_{i,i}^{(i)} & a_{i,i+1}^{(i)} \\ a_{i+1,i}^{(i)} & a_{i+1,i+1}^{(i)} \end{bmatrix} \text{ and those of } \mathbf{b}^{(i)} \text{ as } \mathbf{b}^{(i)} = \begin{bmatrix} b_i^{(i)} \\ b_{i+1}^{(i)} \end{bmatrix}.$$

Substituting $i = 0, 1, 2, \ldots, n-1$ in (32.38) and taking summation over all the elements as

$$\begin{bmatrix} a_{0,0}^{(0)} & a_{0,1}^{(0)} \\ a_{1,0}^{(0)} & a_{1,1}^{(0)} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} + \begin{bmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} \\ a_{2,1}^{(1)} & a_{2,2}^{(1)} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \cdots + \begin{bmatrix} a_{i,i}^{(i)} & a_{i,i+1}^{(i)} \\ a_{i+1,i}^{(i)} & a_{i+1,i+1}^{(i)} \end{bmatrix} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix}$$
$$+ \cdots + \begin{bmatrix} a_{n-1,n-1}^{(n-1)} & a_{n-1,n}^{(n-1)} \\ a_{n,n-1}^{(n-1)} & a_{n,n}^{(n-1)} \end{bmatrix} \begin{bmatrix} y_{n-1} \\ y_n \end{bmatrix}$$
$$= \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} + \cdots + \begin{bmatrix} b_i^{(i)} \\ b_{i+1}^{(i)} \end{bmatrix} + \cdots + \begin{bmatrix} b_{n-1}^{(n-1)} \\ b_n^{(n-1)} \end{bmatrix}.$$

After simplification, the assembled matrix equation becomes

$$\begin{bmatrix} a_{0,0}^{(0)} & a_{0,1}^{(0)} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{1,0}^{(0)} & a_{1,1}^{(0)} + a_{1,1}^{(1)} & a_{1,2}^{(1)} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{2,1}^{(1)} & a_{2,2}^{(1)} + a_{2,2}^{(2)} & a_{2,2}^{(2)} & 0 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & a_{n,n-1}^{(n-1)} & a_{n,n}^{(n-1)} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} + b_1^{(1)} \\ b_2^{(1)} + b_2^{(2)} \\ \vdots \\ b_n^{(n-1)} \end{bmatrix}.$$

This equation can be written as

$$\mathbf{A}\phi = \mathbf{b}. \tag{32.40}$$

It may be noted that $\mathbf{A}$ is a tri-diagonal matrix.

Step 5. Incorporation of boundary conditions

For simplicity, let the system (32.40) be of the following form.

$$\begin{bmatrix} a_{0,0} & a_{0,1} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{1,0} & a_{1,1} & a_{12} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} & 0 & \cdots & 0 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}. \tag{32.41}$$

The second and nth equations are $a_{1,0}y_0 + a_{1,1}y_1 + a_{1,2}y_2 = b_1$ and $a_{n-1,n-2}y_{n-2} + a_{n-1,n-1}y_{n-1} + a_{n-1,n}y_n = b_{n-1}$. Now, the boundary conditions $y(a) = y_a$ and $y(b) = y_b$ are introduced to the above equations and they become

$$\begin{aligned} & a_{1,1}y_1 + a_{1,2}y_2 = b_1 - a_{1,0}y_a \\ \text{and} \quad & a_{n-1,n-2}y_{n-2} + a_{n-1,n-1}y_{n-1} = b_{n-1} - a_{n-1,n}y_n. \end{aligned} \tag{32.42}$$

Now, first and last rows and columns are removed from **A** also first and last elements are removed from **b** and the equations of (32.42) are incorporated.

The equation (32.41) finally reduces to

$$\begin{bmatrix} a_{1,1} & a_{12} & 0 & 0 & \cdots & 0 & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix}. \tag{32.43}$$

The above equation is a tri-diagonal system of equations containing $(n-1)$ unknowns $y_1, y_2, \ldots, y_{n-1}$.

32.8 Stability Analysis

Stability analysis is an important part in the study of the numerical methods to solve differential equations. Most of the methods used to solve differential equation are based on difference equation. To study the stability, the model differential and difference equations are defined in the following.

32.8.1 Model differential problem

For convenience and feasibility of analytical treatment and without loss of generality, stability analysis will be performed on the **model initial value differential equation**

$$y' = \lambda y, \qquad y(0) = y_0, \tag{32.44}$$

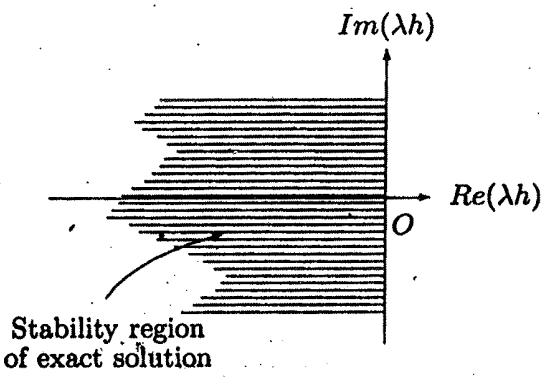


Figure 32.2: Stability region of exact solution.

where λ is a constant and it may be a real or a complex number. The solution of this problem is

$$y = e^{\lambda t} y_0. \tag{32.45}$$

In our treatment, let us consider $\lambda = \lambda_R + i\lambda_I$, where λ_R and λ_I represent respectively the real and imaginary parts of λ, with $\lambda_R \leq 0$.

32.8.2 Model difference problem

Similar to the differential problem, let us consider the single first order linear model initial value difference problem

$$y_{n+1} = \sigma y_n, \quad n = 0, 1, 2, \ldots \tag{32.46}$$

where y_0 is given and σ is, in general, a complex number. The solution of this problem is

$$y_n = \sigma^n y_0. \tag{32.47}$$

It may be noted that the solution remains bounded only if $|\sigma| \leq 1$.

The connection between the exact solution and the difference solution is evident if we evaluate the exact solution at $t_n = nh$, for $n = 0, 1, \ldots$ where $h > 0$ and

$$y_n = e^{\lambda t_n} y_0 = e^{\lambda n h} y_0 = \sigma^n y_0 \tag{32.48}$$

where $\sigma = e^{\lambda h}$.

If the exact solution is bounded then $|\sigma| = |e^{\lambda h}| \leq 1$. This is possible if $Re(\lambda h) = \lambda_R h \leq 0$.

That is, in the $Re(\lambda h)$-$Im(\lambda h)$ plane, the region of stability of the exact solution is the left half-plane as shown in Figure 32.2.

The single-step method is called **absolutely stable** if $|\sigma| \leq 1$ and **relatively stable** if $|\sigma| \leq e^{\lambda h}$. If λ is pure imaginary and $|\sigma| = 1$, then the absolute stability is called the **periodic stability (P-stability)**.

When the region of stability of a difference equation is identical to the region of stability of the differential equation, the finite difference scheme is sometimes referred to as **A-stable.**

32.8.3 Stability of Runge-Kutta methods

Let us consider the second-order Runge-Kutta method for the model equation $y' = \lambda y$. Then

$$\begin{aligned} k_1 &= hf(x_n, y_n) = \lambda h y_n \\ k_2 &= hf(x_n + h, y_n + k_1) = \lambda h(y_n + k_1) = \lambda h(y_n + \lambda h y_n) \\ &= \lambda h(1 + \lambda h) y_n \end{aligned}$$

and

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2) = y_n + \left(\lambda h + \frac{(\lambda h)^2}{2}\right) y_n \\ &= \left(1 + \lambda h + \frac{\lambda^2 h^2}{2}\right) y_n. \end{aligned} \tag{32.49}$$

This expression confirms that the method is of second order accuracy. For stability, $|\sigma| \leq 1$ where

$$\sigma = 1 + \lambda h + \frac{\lambda^2 h^2}{2}. \tag{32.50}$$

Now, the stability is discussed for different cases of λ.

(i) Real λ: $1 + \lambda h + \frac{\lambda^2 h^2}{2} \leq 1$ or, $-2 \leq \lambda h \leq 0$.

(ii) Pure imaginary λ: Let $\lambda = iw$. Then $|\sigma| = \sqrt{1 + \frac{1}{4} w^4 h^4} > 1$. That is, the method is unstable.

(iii) Complex λ: Let $1 + \lambda h + \frac{\lambda^2 h^2}{2} = e^{i\theta}$ and find the complex roots, λh, of this polynomial for different values of θ. Note that $|\sigma| = 1$ for all values of θ.

The resulting stability region is shown in Figure 32.3.

When fourth-order Runge-Kutta method is applied to the model equation $y' = \lambda y$ then

$$\begin{aligned} k_1 &= \lambda h y_n \\ k_2 &= \lambda h(y_n + k_1/2) = \lambda h(1 + \lambda h/2) y_n \\ k_3 &= \lambda h(y_n + k_2/2) = \lambda h\left(1 + \frac{1}{2}\lambda h + \frac{1}{4}\lambda^2 h^2\right) y_n \\ k_4 &= \lambda h(y_n + k_3) = \lambda h\left(1 + \lambda h + \frac{1}{2}\lambda^2 h^2 + \frac{1}{4}\lambda^3 h^3\right) y_n. \end{aligned}$$

Therefore,

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= \left\{1 + \lambda h + \frac{1}{2!}(\lambda h)^2 + \frac{1}{3!}(\lambda h)^3 + \frac{1}{4!}(\lambda h)^4\right\} y_n, \end{aligned} \tag{32.51}$$

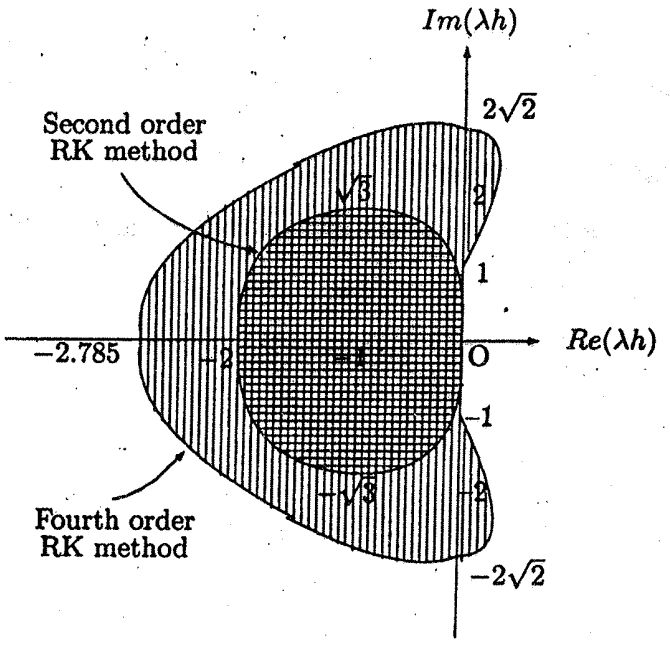


Figure 32.3: Stability regions of Runge-Kutta methods.

which confirms the fourth-order accuracy of the method.

For different λ, the stability of fourth-order Runge-Kutta method is discussed in the following:

(i) Real λ: $-2.785 \leq \lambda h \leq 0$.

(ii) Pure imaginary λ: $0 \leq |\lambda h| \leq 2\sqrt{2}$.

(iii) Complex λ: In this case, the stability region is obtained by finding the roots of the fourth-order polynomial with complex coefficients:
$1 + \lambda h + \frac{1}{2!}(\lambda h)^2 + \frac{1}{3!}(\lambda h)^3 + \frac{1}{4!}(\lambda h)^4 = e^{i\theta}$.

The region of stability is shown in Figure 32.3. It shows a significant improvement over the second-order Runge-Kutta scheme. In particular, it has a large stability region on the imaginary axis.

32.9 Partial Differential Equations

Several analytical methods are available to solve PDEs, but, these methods are based on advanced mathematical techniques. Also, several numerical methods are available to solve PDEs. The numerical methods are, in general, simple but generate erroneous result. Among several numerical methods, the finite-difference method is widely used for its simplicity. In this module, only finite-difference method is discussed to solve PDEs.

The general second order PDE is of the form

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu = G \tag{32.52}$$

i.e.,
$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G, \tag{32.53}$$

where A, B, C, D, E, F, G are functions of x and y.

The PDEs can be classified into three different types – elliptic, hyperbolic and parabolic. These classifications are done by computing the discriminant

$$\Delta = B^2 - 4AC.$$

The equation (32.53) is called elliptic, parabolic and hyperbolic according as the value of Δ at any point (x, y) are $<, =$ or > 0.

Elliptic equations

The simplest examples of this type of PDEs are Poisson's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = g(x, y) \tag{32.54}$$

and Laplace equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad \text{or} \qquad \nabla^2 u = 0. \tag{32.55}$$

Parabolic equation

In parabolic equation, time t is involved as an independent variable.

The simplest example of parabolic equation is the heat conduction equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}. \tag{32.56}$$

The solution u is the temperature at a distance x units of length from one end of a thermally insulated bar after t seconds of heat conduction. In this problem, the temperatures at the ends of a bar are known for all time, i.e., the boundary conditions are known.

Hyperbolic equation

In this equation also, time t appears as an independent variable.

The simplest example of hyperbolic equation is the one-dimensional wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}. \tag{32.57}$$

Here u is the transverse displacement at a distance x from one end of a vibrating string of length l after a time t.

Hyperbolic equations generally originate from vibration problems or from problems where discontinuities can persist in time.

The values of u at the ends of the string are usually known for all time (i.e., boundary conditions are known) and the shape and velocity of the string are given at initial time (i.e., initial conditions are known).

Generally, three types of problems occur in PDEs.

(i) **Dirichlet's problem**

Let R be a region bounded by a closed curve C and f be a continuous function on C.

Now the problem is to find u satisfying the Laplace's equation

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \text{ in } R \\ \text{and} \quad &u = f(x, y) \text{ on } C. \end{aligned} \tag{32.58}$$

(ii) **Cauchy's problem**

$$\begin{aligned} &\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \text{ for } t > 0 \\ &u(x, 0) = f(x) \\ \text{and} \quad &u_t(x, 0) = g(x) \\ &\text{where } f(x) \text{and } g(x) \text{ are arbitrary.} \end{aligned} \tag{32.59}$$

(iii)

$$\begin{aligned} &\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \text{ for } t > 0 \\ \text{and} \quad &u(x, 0) = f(x). \end{aligned} \tag{32.60}$$

The above cited problems are all well-defined (i.e., well-posed) and it can be shown that each of the above problems has unique solution.

But, the problem of Laplace's equation with Cauchy boundary conditions, i.e., the problem

$$\begin{aligned} &\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \\ &u(x, 0) = f(x) \\ \text{and} \quad &u_y(x, 0) = g(x) \end{aligned} \tag{32.61}$$

is an ill-posed problem.

32.10 Finite-Difference Approximations to Partial Derivatives

Let the xy plane be divided into sets of equal rectangles of sides $\Delta x = h$ and $\Delta y = k$ by drawing the equally spaced grid lines parallel to the coordinates axes, defined by

$$x_i = ih, \qquad i = 0, \pm 1, \pm 2, \ldots$$
$$y_j = jk, \qquad j = 0, \pm 1, \pm 2, \ldots.$$

The value of u at a mesh point (intersection of horizontal and vertical lines) $P(x_i, y_j)$ or, at $P(ih, jk)$ is denoted by $u_{i,j}$, i.e., $u_{i,j} = u(x_i, y_j) = u(ih, jk)$.

Now,

$$u_x(ih, jk) = \frac{u_{i+1,j} - u_{i,j}}{h} + O(h) \tag{32.62}$$

(forward difference approximation)

$$= \frac{u_{i,j} - u_{i-1,j}}{h} + O(h) \tag{32.63}$$

(backward difference approximation)

$$= \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O(h^2) \tag{32.64}$$

(central difference approximation)

Similarly,

$$u_y(ih, jk) = \frac{u_{i,j+1} - u_{i,j}}{k} + O(k) \tag{32.65}$$

(forward difference approximation)

$$= \frac{u_{i,j} - u_{i,j-1}}{k} + O(k) \tag{32.66}$$

(backward difference approximation)

$$= \frac{u_{i,j+1} - u_{i,j-1}}{2k} + O(k^2) \tag{32.67}$$

(central difference approximation)

$$u_{xx}(ih, jk) = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + O(h^2). \tag{32.68}$$

$$u_{yy}(ih, jk) = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{k^2} + O(k^2). \tag{32.69}$$

32.11 Parabolic Equations

32.11.1 An explicit method

Let us consider the heat conduction equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}. \tag{32.70}$$

Using the finite-difference approximation for u_t and u_{xx}, equation (32.70) reduces to

$$\frac{u_{i,j+1} - u_{i,j}}{k} \simeq \alpha \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}, \tag{32.71}$$

where $x_i = ih$ and $t_j = jk$; $i, j = 0, 1, 2, \ldots$.

This can be written as

$$u_{i,j+1} = ru_{i-1,j} + (1 - 2r)u_{i,j} + ru_{i+1,j}, \tag{32.72}$$

where $r = \alpha k/h^2$.

This formula gives the unknown 'temperature' $u_{i,j+1}$ at the mesh point $(i, j+1)$ when the values of $u_{i-1,j}, u_{i,j}$ and $u_{i+1,j}$ are known and hence the method is called the explicit method. It can be shown that (by stability analysis) the formula is valid for $0 < r \leq 1/2$.

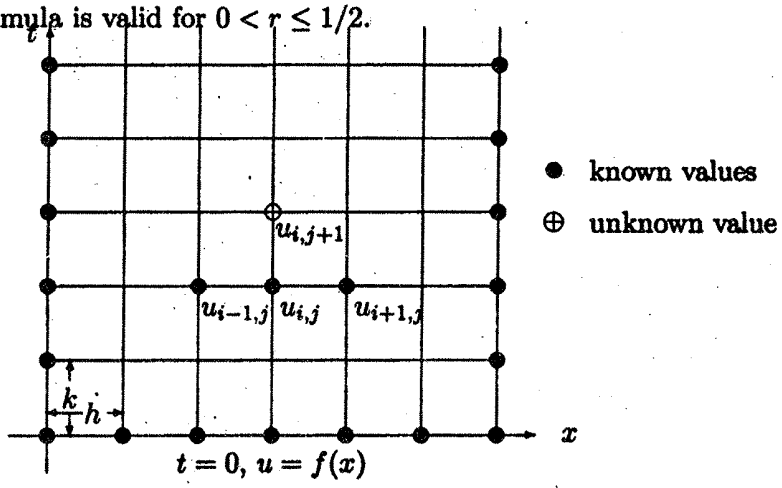


Figure 32.4: Known and unknown meshes of heat equation.

Example 32.11.1 Solve the heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

subject to the boundary conditions $u(0,t) = 0, u(1,t) = 2t$ and initial condition $u(x,0) = \frac{1}{2}x$.

Solution. Let $h = 0.2$ and $k = 0.01$, so $r = k/h^2 = 0.25 < 1/2$. The initial and boundary values are shown in the following table.

	$i=0$ $x=0$	$i=1$ $x=0.2$	$i=2$ $x=0.4$	$i=3$ $x=0.6$	$i=4$ $x=0.8$	$i=5$ $x=1.0$
$j=0, t=0.00$	0.0	0.1	0.2	0.3	0.4	0.00
$j=1, t=0.01$	0.0					0.02
$j=2, t=0.02$	0.0					0.04
$j=3, t=0.03$	0.0					0.06
$j=4, t=0.04$	0.0					0.08
$j=5, t=0.05$	0.0					0.10

For this problem the equation (32.72) reduces to

$$u_{i,j+1} = \frac{1}{4}(u_{i-1,j} + 2u_{i,j} + u_{i+1,j}).$$

Then

$$u_{1,1} = \frac{1}{4}(u_{0,0} + 2u_{1,0} + u_{2,0}) = 0.1$$
$$u_{2,1} = \frac{1}{4}(u_{1,0} + 2u_{2,0} + u_{3,0}) = 0.2$$

and so on.

The complete values are shown in the following table.

	$i=0$ $x=0$	$i=1$ $x=0.2$	$i=2$ $x=0.4$	$i=3$ $x=0.6$	$i=4$ $x=0.8$	$i=5$ $x=1.0$
$j=0, t=0.00$	0.0	0.1	0.2	0.3	0.4	0.00
$j=1, t=0.01$	0.0	0.10000	0.20000	0.30000	0.27500	0.02
$j=2, t=0.02$	0.0	0.10000	0.20000	0.26875	0.21750	0.04
$j=3, t=0.03$	0.0	0.10000	0.19219	0.23875	0.18594	0.06
$j=4, t=0.04$	0.0	0.09805	0.18078	0.21391	0.16766	0.08
$j=5, t=0.05$	0.0	0.09422	0.16838	0.19406	0.15730	0.10

32.11.2 Crank-Nicolson implicit method

The above explicit method is simple but it has one serious drawback. This method gives a meaningfull result if $0 < r \leq 1/2$, i.e., $0 < \alpha k/h^2 \leq 1/2$ or, $\alpha k \leq h^2/2$. This means, the time step k is necessarily small. Crank and Nicolson (1947) have developed a method that reduces the total computation time and is valid for all finite values of r. In this method, the equation is approximated by replacing both space and time derivatives by their central difference approximations at a point $(ih, (j+1/2)k)$, which is the midpoint of the points (ih, jk) and $(ih, (j+1)k)$. Thus the equation (32.70) can be written as

$$\left(\frac{\partial u}{\partial t}\right)_{i,j+1/2} = \alpha\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j+1/2} = \frac{\alpha}{2}\left[\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} + \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j+1}\right]. \tag{32.73}$$

Then by using central difference approximation the above equation reduces to

$$\frac{u_{i,j+1}-u_{i,j}}{k}=\frac{\alpha}{2}\left[\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{k^2}+\frac{u_{i+1,j+1}-2u_{i,j+1}+u_{i-1,j+1}}{k^2}\right]$$

and after simplification this equation gives

$$-ru_{i-1,j+1}+(2+2r)u_{i,j+1}-ru_{i+1,j+1}=ru_{i-1,j}+(2-2r)u_{i,j}+ru_{i+1,j}, \tag{32.74}$$

where $r=\alpha k/h^2$.

In general, the left hand side of (32.74) contains three unknowns and the right hand side has three known values of u.

For $j=0$ and $i=1,2,\ldots,N-1$, equation (32.74) generates N simultaneous equations for $N-1$ unknown $u_{1,1},u_{2,1},\ldots,u_{N-1,1}$ (of first row) in terms of known initial and boundary values $u_{0,0},u_{1,0},u_{2,0},\ldots,u_{N,0}$; $u_{0,0}$ and $u_{N,0}$ are the boundary values and $u_{1,0},u_{2,0},\ldots,u_{N-1,0}$ are the initial values.

Similarly, for $j=1$ and $i=1,2,\ldots,N-1$ we obtain another set of unknown values $u_{1,2},u_{2,2},\ldots,u_{N-1,2}$ in terms of calculated values for $j=0$, and so on.

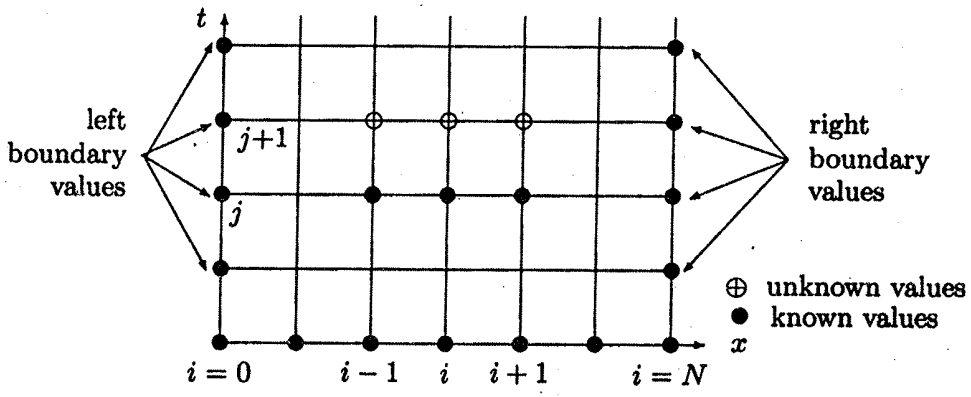


Figure 32.5: Meshes of Crank-Nicolson implicit method.

In this method, the value of $u_{i,j+1}$ is not given directly in terms of known $u_{i,j}$ at one time step earlier but is also a function of unknown values at previous time step, and hence the method is called an implicit method.

The system of equation (32.74) can be viewed in the following matrix notation.

$$\begin{bmatrix} 2+2r & -r & & & & & \\ -r & 2+2r & -r & & & & \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & -r & 2+2r & -r & \\ & & & & -r & 2+2r & \end{bmatrix}\begin{bmatrix} u_{1,j+1} \\ u_{2,j+1} \\ u_{3,j+1} \\ \vdots \\ u_{N-1,j+1} \end{bmatrix}=\begin{bmatrix} d_{1,j} \\ d_{2,j} \\ d_{3,j} \\ \vdots \\ d_{N-1,j} \end{bmatrix} \tag{32.75}$$

where

$$d_{1,j} = ru_{0,j} + (2-2r)u_{1,j} + ru_{2,j} + ru_{0,j+1}$$
$$d_{i,j} = ru_{i-1,j} + (2-2r)u_{i,j} + ru_{i+1,j}; \qquad i = 2, 3, \ldots, N-2$$
$$d_{N-1,j} = ru_{N-2,j} + (2-2r)u_{N-1,j} + ru_{N,j} + ru_{N,j+1}.$$

The right hand side of (32.75) is known.

Example 32.11.2 Use the Crank-Nicolson method to calculate a numerical solution of the problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad 0 < x < 1, t > 0$$

where $u(0,t) = u(1,t) = 0, t > 0, u(x,0) = 2x, t = 0$. Mention the value of $u\left(\frac{1}{2}, \frac{1}{8}\right)$ by taking $h = \frac{1}{2}, \frac{1}{4}$ and $k = \frac{1}{8}, \frac{1}{16}$.

Solution. Case I.
Let $h = \frac{1}{2}$ and $k = \frac{1}{8}$.

In this case $r = \frac{k}{h^2} = \frac{1}{2}$.
The Crank-Nicolson scheme (32.74) now becomes

$$-u_{i-1,j+1} + 6u_{i,j+1} - u_{i+1,j+1} = u_{i-1,j} + 2u_{i,j} + u_{i+1,j}.$$

The boundary and initial conditions are shown in Figure 32.6.

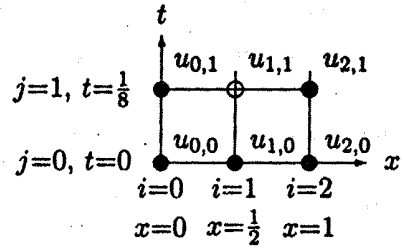


Figure 32.6: Boundary and initial values when $h = 1/2, k = 1/8$.

The initial and boundary values are

$$u_{0,0} = 0, u_{1,0} = 1, u_{2,0} = 2; u_{0,1} = 0, u_{2,1} = 0.$$

Substituting $i = 0$ and $j = 0$ to the Crank-Nicolson scheme, we obtain

$$-u_{0,1} + 6u_{1,1} - u_{2,1} = u_{0,0} + 2u_{1,0} + u_{2,0}.$$

Using initial and boundary conditions the above equation reduces to

$$6u_{1,1} = 2 + 2 = 4, \text{ i.e., } u_{1,1} = u(1/2, 1/8) = 2/3.$$

Case II.

Let $h = \frac{1}{4}$ and $k = \frac{1}{8}$. In this case $r = \frac{k}{h^2} = 2$.

The Crank-Nicolson scheme is

$$-u_{i-1,j+1} + 3u_{i,j+1} - u_{i+1,j+1} = u_{i-1,j} - u_{i,j} + u_{i+1,j}.$$

The initial and boundary conditions are shown in Figure 32.7.

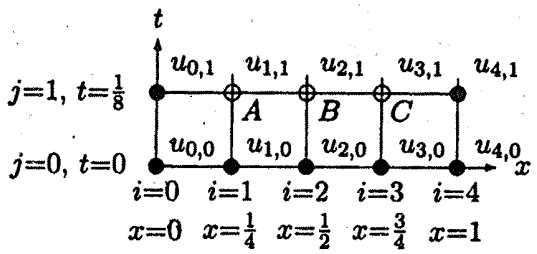


Figure 32.7: Boundary and initial values when $h = 1/4, k = 1/8$.

That is, $u_{0,0} = 0, u_{1,0} = \frac{1}{2}, u_{2,0} = 1, u_{3,0} = \frac{3}{2}, u_{4,0} = 2; u_{0,1} = 0, u_{4,1} = 0$.

The Crank-Nicolson equations for the mesh points $A(i = 1), B(i = 2)$ and $C(i = 3)$ are respectively.

$$-u_{0,1} + 3u_{1,1} - u_{2,1} = u_{0,0} - u_{1,0} + u_{2,0}$$
$$-u_{1,1} + 3u_{2,1} - u_{3,1} = u_{1,0} - u_{2,0} + u_{3,0}$$
$$-u_{2,1} + 3u_{3,1} - u_{4,1} = u_{2,0} - u_{3,0} + u_{4,0}.$$

Using initial and boundary conditions the above system becomes

$$0 + 3u_{1,1} - u_{2,1} = 0 - \frac{1}{2} + 1 = \frac{1}{2}$$
$$-u_{1,1} + 3u_{2,1} - u_{3,1} = \frac{1}{2} - 1 + \frac{3}{2} = 1$$
$$-u_{2,1} + 3u_{3,1} + 0 = 1 - \frac{3}{2} + 2 = \frac{3}{2}.$$

The above system has three equations and three unknowns and the solution is

$u_{1,1} = u\left(\frac{1}{4}, \frac{1}{8}\right) = \frac{17}{42}, \quad u_{2,1} = u\left(\frac{1}{2}, \frac{1}{8}\right) = \frac{5}{7}, \quad u_{3,1} = u\left(\frac{3}{4}, \frac{1}{8}\right) = \frac{31}{42}.$

Case III.

Let $h = \frac{1}{4}, k = \frac{1}{16}$. Then $r = 1$. To find the value of u at $t = \frac{1}{8}$ we have to apply two steps instead of one step as in Case I and Case II.

The Crank-Nicolson scheme for this case is

$$-u_{i-1,j+1} + 4u_{i,j+1} - u_{i+1,j+1} = u_{i-1,j} + u_{i+1,j}.$$

The initial and boundary conditions are shown in Figure 32.8.

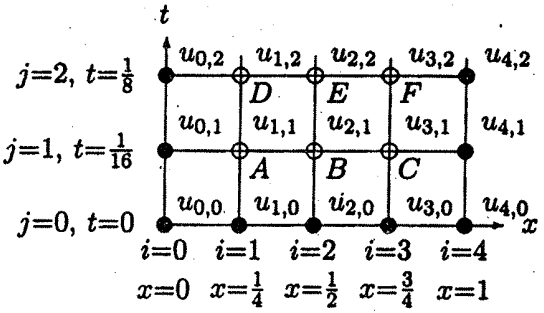


Figure 32.8: Boundary and initial values when $h = 1/4, k = 1/16$.

Also, $u_{0,0} = 0, u_{1,0} = \frac{1}{2}, u_{2,0} = 1, u_{3,0} = \frac{3}{2}, u_{4,0} = 2; u_{0,1} = 0, u_{4,1} = 0; u_{0,2} = 0, u_{4,2} = 0$

The Crank-Nicolson equations for the mesh points $A(i = 1, j = 1)$, $B(i = 2, j = 1)$ and $C(i = 3, j = 1)$ are respectively

$$-u_{0,1} + 4u_{1,1} - u_{2,1} = u_{0,0} + u_{2,0}$$
$$-u_{1,1} + 4u_{2,1} - u_{3,1} = u_{1,0} + u_{3,0}$$
$$-u_{2,1} + 4u_{3,1} - u_{4,1} = u_{2,0} + u_{4,0}.$$

That is,

$$4u_{1,1} - u_{2,1} = 1$$
$$-u_{1,1} + 4u_{2,1} - u_{3,1} = \frac{1}{2} + \frac{3}{2} = 2$$
$$-u_{2,1} + 4u_{3,1} = 3.$$

The solution of this system is

$$u_{1,1} = u\left(\frac{1}{4}, \frac{1}{16}\right) = \frac{13}{28}, u_{2,1} = u\left(\frac{1}{2}, \frac{1}{16}\right) = \frac{6}{7}, u_{3,1} = u\left(\frac{3}{4}, \frac{1}{16}\right) = \frac{27}{28}.$$

Again, the Crank-Nicolson equations for the mesh points $D(i = 1, j = 2), E(i = 2, j = 2)$ and $F(i = 3, j = 2)$ are respectively,

$$-u_{0,2} + 4u_{1,2} - u_{2,2} = u_{0,1} + u_{2,1}$$
$$-u_{1,2} + 4u_{2,2} - u_{3,2} = u_{1,1} + u_{3,1}$$
$$-u_{2,2} + 4u_{3,2} - u_{4,2} = u_{2,1} + u_{4,1}.$$

Using boundary conditions and values of right hand side obtained in first step, the above system becomes

$$\begin{aligned} 4u_{1,2} - u_{2,2} &= 0 + \frac{6}{7} = \frac{6}{7} \\ -u_{1,2} + 4u_{2,2} - u_{3,2} &= \frac{13}{28} + \frac{27}{28} = \frac{10}{7} \\ -u_{2,2} + 4u_{3,2} &= \frac{6}{7} + 0 = \frac{6}{7}. \end{aligned}$$

The solution of this system is

$$u_{1,2} = u\left(\frac{1}{4}, \frac{1}{8}\right) = \frac{17}{49}, u_{2,2} = u\left(\frac{1}{2}, \frac{1}{8}\right) = \frac{26}{49}, u_{3,2} = u\left(\frac{3}{4}, \frac{1}{8}\right) = \frac{17}{49}.$$

32.12 Hyperbolic Equations

Let us consider the wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}, t > 0, 0 < x < 1 \tag{32.76}$$

where initial conditions $u(x, 0) = f(x)$ and

$$\left.\frac{\partial u}{\partial t}\right)_{(x,0)} = g(x), 0 < x < 1 \tag{32.77}$$

and boundary conditions

$$u(0, t) = \phi(t) \text{ and } u(1, t) = \psi(t), t \geq 0. \tag{32.78}$$

This problem may occur in the transverse vibration of a stretched string. As in the previous cases, the central-difference approximation for u_{xx} and u_{tt} at the mesh points $(x_i, t_j) = (ih, jk)$ are

$$\begin{aligned} u_{xx} &= \frac{1}{h^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + O(h^2) \\ \text{and } u_{tt} &= \frac{1}{k^2}(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) + O(k^2), \end{aligned}$$

where $i, j = 0, 1, 2, \ldots$.

Using the value of u_{xx} and u_{tt}, the equation (32.76) becomes

$$\frac{1}{k^2}(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}) = \frac{c^2}{h^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

i.e.,

$$u_{i,j+1} = r^2 u_{i-1,j} + 2(1 - r^2)u_{i,j} + r^2 u_{i+1,j} - u_{i,j-1}, \tag{32.79}$$

where $r = ck/h$.

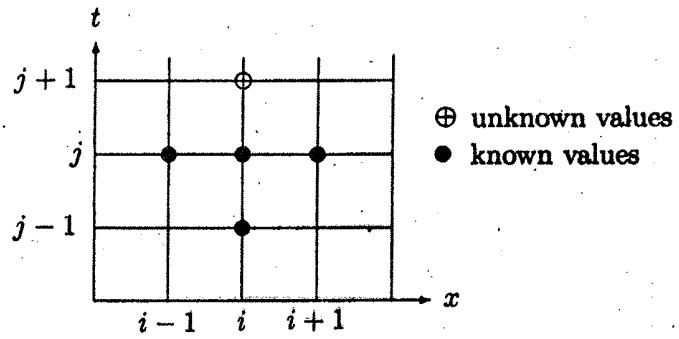


Figure 32.9: Known and unknown meshes for hyperbolic equations.

The value of $u_{i,j+1}$ depends on the values of u at three time-levels $(j-1), j$ and $(j+1)$. The known and unknown values of u are shown in Figure 32.9.

On substituting $j = 0$, the equation (32.79) yields

$$\begin{aligned} u_{i,1} &= r^2 u_{i-1,0} + 2(1-r^2)u_{i,0} + r^2 u_{i+1,0} - u_{i,-1} \\ &= r^2 f_{i-1} + 2(1-r^2)f_i + r^2 f_{i+1} - u_{i,-1}, \text{ where } f_i = f(x_i). \end{aligned}$$

Again, the central difference approximation to the initial derivative condition gives

$$\frac{1}{2k}(u_{i,1} - u_{i,-1}) = g_i.$$

Eliminating $u_{i,-1}$ between above two relations and we obtain the expression for u along $t = k$, i.e., for $j = 1$ as

$$u_{i,1} = \frac{1}{2}\left[r^2 f_{i-1} + 2(1-r^2)f_i + r^2 f_{i+1} + 2kg_i\right]. \tag{32.80}$$

The truncation error of this method is $O(h^2 + k^2)$ and the formula (32.79) is convergent for $0 < r \leq 1$.

Example 32.12.1 Solve the second-order wave equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

with boundary conditions $u = 0$ at $x = 0$ and 1, $t > 0$ and the initial conditions $u = \frac{1}{2}\sin \pi x$, $\frac{\partial u}{\partial t} = 0$, when $t = 0, 0 \leq x \leq 1$, for $x = 0, 0.2, 0.4, \ldots, 1.0$ and $t = 0, 0.1, 0.2, \ldots, 0.5$.

Solution. The explicit formula is

$$u_{i,j+1} = r^2 u_{i-1,j} + 2(1-r^2)u_{i,j} + r^2 u_{i+1,j} - u_{i,j-1}.$$

Let $h = 0.2$ and $k = 0.1$, so $r = k/h = 0.5 < 1$.

The boundary conditions become $u_{0,j} = 0, u_{5,j} = 0$ and initial conditions reduce to $u_{i,0} = \frac{1}{2}\sin\pi(ih)$, $i = 1, 2, 3, 4, 5$ and $\frac{u_{i,1} - u_{i,-1}}{2k} = 0$, so that $u_{i,-1} = u_{i,1}$.

For $r = 0.5$, the difference scheme is then

$$u_{i,j+1} = 0.25u_{i-1,j} + 1.5u_{i,j} + 0.25u_{i+1,j} - u_{i,j-1}. \tag{32.81}$$

For $j = 0$, this relation becomes

$$\begin{aligned} u_{i,1} &= 0.25u_{i-1,0} + 1.5u_{i,0} + 0.25u_{i+1,0} - u_{i,-1} \\ \text{i.e., } u_{i,1} &= 0.125u_{i-1,0} + 0.75u_{i,0} + 0.125u_{i+1,0}, \text{ [using } u_{i,-1} = u_{i,1}] \\ &= 0.125(u_{i-1,0} + u_{i+1,0}) + 0.75u_{i,0}. \end{aligned}$$

The above formula gives the values of u for $j = 1$. For $j = 2, 3, \ldots$ the values are obtained from the formula (32.81).

Hence,

$$\begin{aligned} u_{1,1} &= 0.125(u_{0,0} + u_{2,0}) + 0.75u_{1,0} = 0.27986 \\ u_{2,1} &= 0.125(u_{1,0} + u_{3,0}) + 0.75u_{2,0} = 0.45282 \\ u_{3,1} &= 0.125(u_{2,0} + u_{4,0}) + 0.75u_{3,0} = 0.45282 \\ u_{4,1} &= 0.125(u_{3,0} + u_{5,0}) + 0.75u_{4,0} = 0.27986. \end{aligned}$$

All values are shown in the following table.

	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
	$x = 0$	$x = 0.2$	$x = 0.4$	$x = 0.6$	$x = 0.8$	$x = 1.0$
$j = 0, t = 0.0$	0	0.29389	0.47553	0.47553	0.29389	0
$j = 1, t = 0.1$	0	0.27986	0.45282	0.45282	0.27986	0
$j = 2, t = 0.2$	0	0.23910	0.38688	0.38688	0.23910	0
$j = 3, t = 0.3$	0	0.17552	0.28399	0.28399	0.17552	0
$j = 4, t = 0.4$	0	0.09517	0.15398	0.15398	0.09517	0
$j = 5, t = 0.5$	0	0.00573	0.00927	0.00927	0.00573	0

The exact solution of this equation is

$$u(x, t) = \frac{1}{2}\sin\pi x \cos\pi t.$$

32.12.1 Implicit difference methods

The implicit methods generate a tri-diagonal system of algebraic equations. So, it is suggested that the implicit methods should not be used without simplifying assumption to solve pure IVPs because they generate an infinite number of system of equations. But, these methods may be used for initial-boundary value problems. Two such implicit methods are presented below.

(i) At the mesh point (ih, jk) the scheme is

$$\left(\frac{\partial^2 u}{\partial t^2}\right)_{i,j} = \frac{c^2}{2}\left[\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j+1} + \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j-1}\right].$$

That is,

$$\begin{aligned} &\frac{1}{k^2}[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] \\ &= \frac{c^2}{2h^2}[(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}) + (u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1})]. \end{aligned} \tag{32.82}$$

(ii) $\left(\frac{\partial^2 u}{\partial t^2}\right)_{i,j} = \frac{c^2}{4}\left[\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j+1} + 2\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} + \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j-1}\right]$

Using the finite difference scheme this equation reduces to

$$\begin{aligned} &\frac{1}{k^2}[u_{i,j+1} - 2u_{i,j} + u_{i,j-1}] \\ &= \frac{c^2}{4h^2}[(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}) \\ &\quad +2(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + (u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1})]. \end{aligned} \tag{32.83}$$

Both the formulae hold good for all values of $r = ck/h > 0$.

32.13 Module Summary

In this module we have discussed two methods – Runge-Kutta and Milne's predictor-corrector methods to find numerical solution of ordinary differential equations. The stability of the solution of Runge-Kutta methods is also discussed here. The finite element method to solve an ordinary differential equation is introduced. The finite difference scheme for partial derivative is given. The numerical solution method to solve two important types of partial differential equations, viz., heat equation and wave equation is discussed in this module.

32.14 Self Assessment Questions

1. Deduce second and fourth order Runge-Kutta methods to solve the initial value problem $y' = f(x, y)$ with $y(x_0) = y_0$.

2. Explain the significance of the numbers k_1, k_2, k_3, k_4 used in fourth order Runge-Kutta methods.

3. Analyze the stability of second and fourth order Runge-Kutta methods.

4. Use second order Runge-Kutta method to solve the initial value problem

$$5\frac{dy}{dx} = x^2 + y^2, \qquad y(0) = 1$$

and find y in the interval $0 \leq x \leq 0.4$, taking $h = 0.1$.

5. Using Runge-Kutta method of fourth order, solve

$$\frac{dy}{dx} = xy + y^2,$$

given that $y(0) = 1$. Take $h = 0.2$ and find y at $x = 0.2, 0.4, 0.6$.

6. Use Runge-Kutta method of fourth order to compute the solution of the following problem in the interval [0,0.1] with $h = 0.02$.

$$y' = x + y, \qquad y(0) = 1.$$

7. Use fourth order Runge-Kutta method to solve the second order initial value problem $2y''(x) - 6y'(x) + 2y(x) = 4e^x$ with $y(0) = 1$ and $y'(0) = 1$, at $x = 0.1, 0.2$.

8. Use fourth order Runge-Kutta method to solve $y'' = xy' + y$ with initial conditions $y(0) = 1, y'(0) = 2$. Take $h = 0.2$ and find y and y' at $x = 0.2$.

9. Discuss Milne's predictor-corrector formula to find the solution of $y' = f(x, y)$, $y(x_0) = y_0$.

10. Use Milne's predictor-corrector formula to find the solutions at $x = 0.4, 0.5, 0.6$ of the differential equation

$$\frac{dy}{dx} = x^3 + y^2, \qquad y(0) = 1.$$

11. Using Milne's predictor-corrector method, find the solution of $y' = xy + y^2$, $y(0) = 1$ at $x = 0.4$ given that $y(0.1) = 1.11689, y(0.2) = 1.27739$ and $y(0.3) = 1.50412$.

12. Consider the boundary value problem $y'' - y = 1, \quad 0 < x < 1$ with the boundary conditions $y(0) = 0, y(1) = e - 1$.

    Use finite element method to find the solution of this problem.

13. Solve the boundary value problem $y''+y = x^2$, $0 < x < 1$ with boundary condition $y(0) = 3, y(1) = 1$ using finite element method for two and three elements of equal lengths.

14. Solve the heat equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

subject to the conditions $u(x,0) = 0, u(0,t) = 0$ and $u(1,t) = 2t$; taking $h = 1/2, k = 1/16$.

15. Given the differential equation $\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}$ and the boundary condition $u(0,t) = u(5,t) = 0$ and $u(x,0) = x^2(30 - x^2)$. Use the explicit method to obtain the solution for $x_i = ih, y_j = jk$; $i = 0, 1, \ldots, 5$ and $j = 0, 1, 2, \ldots, 6$.

16. Solve the differential equation $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, $0 \leq x \leq 1/2$, given that $u = 0$ when $t = 0, 0 \leq x \leq 1/2$ and with boundary conditions $\frac{\partial u}{\partial x} = 0$ at $x = 0$ and $\frac{\partial u}{\partial x} = 1$ at $x = 1/2$ for $t > 0$, taking $h = 0.1, k = 0.001$.

17. Solve the following initial value problem $f_t = f_{xx}$, $0 \leq x \leq 1$ subject to the initial condition $f(x,0) = \cos\frac{\pi x}{2}$ and the boundary conditions $f(0,t) = 1, f(1,t) = 0$ for $t > 0$, taking $h = 1/3, k = 1/3$.

18. Solve the parabolic differential equation $u_t = u_{xx}, 0 \leq x \leq 1$ subject to the boundary conditions $u = 0$ at $x = 0$ and $x = 1$ for $t > 0$ and the initial conditions

$$u(x,0) = \begin{cases} 2x & \text{for } 0 \leq x \leq 1/2 \\ 2(1-x) & \text{for } 1/2 \leq x \leq 1, \end{cases}$$

using explicit method.

19. The differential equation $u_{tt} = u_{xx}, 0 \leq x \leq 1$ satisfies the boundary conditions $u = 0$ at $x = 0$ and $x = 1$ for $t > 0$, and the initial conditions $u(x,0) = \sin\frac{\pi x}{4}$, $\left(\frac{\partial u}{\partial t}\right)_{(x,0)} = 0$. Compute the values of u for $x = 0, 0.1, 0.2, \ldots, 0.5$ and $t = 0, 0.1, 0.2, \ldots, 0.5$.

32.15 References

1. M.Pal, Numerical Analysis for Scientists and Engineers: Theory and C Programs, Narosa, 2007.

2. M.K. Jain, S.R.K. Iyengar and R.K. Jain, Numerical Methods for Scientific and Engineering Computation, New Age International (P) Limited, New Delhi, 1984.

3. J.H. Mathews, Numerical Methods for Mathematics, Science, and Engineering, 2nd ed., Prentice-Hall, Inc., N.J., U.S.A., 1992.

4. S.S.Sastry, Introductory Method of Numerical Analysis, PHI, 2006.

VIDYASAGAR UNIVERSITY

DIRECTORATE OF DISTANCE EDUCATION
MIDNAPORE - 721 102

M.Sc. in Applied Mathematics with Oceanology and Computer Programming

Part-I
Group-B Paper-III
Module No. - 33

INTRODUCING TO COMPUTING-I

CONTENTS

1.0 Introduction

A digital computer is a machine, which accepts a stream of symbols, stores them, processes them according to precise rules, and produces a stream of symbols at its output. Regardless of the complexity of processing, there are some basic features, which are common to all digital processing of information, which enables us to treat the subject in a unified manner. We introduce algebra, which is useful in designing logic circuit of processor. The representation of data is very important in digital systems.

2.0 Objectives

In this module, the main objectives are to study the followings

- The number system and addition and subtraction of binary numbers
- The concept of Boolean algebra
- The algorithm and flow-chart
- The computer fundamental
- The data representation and binary coded decimal numbers etc.

3.0 Keywords and study guides

Binary, Octal, Hexadecimal, Truth table, Flowchart, Bit, Byte, Nibble, Word, ALU, CU, Peripheral Devices, data representation, Binary Coded Decimal number, Weighted Code, Self-complementing code, Cyclic code, Error-detecting code, Error correcting code, Hamming code, Alphanumeric code.

4.0 Main Discussion

4.1 Different Number System:

A number system defines a set of values used to represent quantity. Quantifying values and items in relation to each other is helpful for us to make sense of our environment. The study of number systems is not just limited to computers. We apply numbers every day, and knowing how numbers work will give us an insight into how a computer manipulates and stores numbers.

Number System:

Number systems are basically of two types: (i) non-positional and (ii) positional.

Non-positional: In early days, human beings counted on fingers. When ten fingers were not adequate, stones, pebbles, or sticks were used to indicate values. This method of counting uses an additive approach or the non-positional number system. In this system, we have symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5,

etc. Each symbol represents the same value regardless of its position in the number and the symbols are simply added to find out the value of a particular number.

Positional: In a positional number system, there are only a few symbols called digits, and these symbols represent different values depending on the position they occupy ion the number. The value of each digit in such a number is determined by three considerations:

- the digit itself
- the position of the digit in the number
- the base of the number system, where the base is defined as the total number of digit available in the number system

Any number can be represented by using the available digits and arranging them in various positions. There are two characteristics of all number systems that are suggested by the value of the base. In all number systems, the value of the base determines the total number of different symbols or digits available in the number system. The first of these choices is always zero. The second characteristic is that the maximum value of a single digit is alway equal to one less than the value of the base.

There are different positional number systems commonly used in computer, which are as follows:

- Decimal number system
- Binary number system
- Octal number system
- Hexadecimal number system

Positional number systems have a radix or a base, which is the number of different digits in system. A number system with radix *r* will have *r* symbols. A number in a radix *r* sysetm would be written in general as:

$$a_n a_{n-1} a_{n-2} \ldots a_0 a_{-1} a_{-2} \ldots a_{-m}$$

and would be interpreted to mean the following

$$a_n r^n + a_{n-1} r^{n-1} + \ldots + a_0 r^0 + a_{-1} r^{-1} + a_{-2} r^{-2} + \ldots + a_{-m} r^{-m}$$

The symbols $a_{n2}, a_{n-1} \ldots a_{-m}$ used in the above representation should be one of the *r* symbols allowed in the system. In the above representation a_n is called the **most significant digits** of the number and a_{-m} (the last digit) is called the **least significant digit.**

Decimal Number System:

This number system used ten different symbols to represent values. The set values used in decimal are 0 1 2 3 4 5 6 7 8 9 with 0 having the least value and nine having the greatest value. The digit or column on the left has the greatest value whilst the digit on the right has the least value. When doing a calculation, if the highest digit (9) is exceeded, a carry occurs which is transferred to the next column (to the left). The successive positions to the left of the decimal point represent units, tens, hundreds, thousands, etc.

Example: The decimal number 2586 written as $(2586)_{10}$ consists of the digits 6 in the units position, 8 in the ten position, 5 in the hundred position and 2 in the thousand position and its value can be written as:

$$(2586)_{10} = 2\times1000+5\times100+8\times10+6\times1$$

$$= 2\times10^3+5\times10^2+8\times10^1+6\times10^0$$

$$= 2000 + 500 + 80 + 6$$

Binary number system: The base of this system is 2. The digits are 0 and 1 only. Each position in a binary number system represents a power of the base 2. In this system the rightmost position is the units (2^0) position, the second position from the right is the 2's (2^1) position and proceeding in this way we have 4's (2^2) position and so on.

Example: The decimal equivalent of the binary number $(10101)_2$ is as follows:

$$(10101)_2 = 1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0$$

$$= (16 + 0 + 4 + 0 + 1)_{10}$$

$$= (21)_{10}$$

Octal number system: In the octal number system the base is 8. So in this system there are only eight symbols or digits: 0, 1, 2, 3, 4, 5, 6, 7. Here also the largest single digit is 7 (one less than the base). Again each position in an octal number represents a power of the base (8).

Example: The decimal equivalent of the octal number $(2057)_8$ is:

$$(2057)_8 = 2\times8^3+0\times8^2+5\times8^1+7\times8^0$$

$$= (1024+0+40+7)_{10}$$

$$= (1071)_{10}$$

We observe that since there are only 8 digits in the octal number system, so 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary.

Hexadecimal number system: The hexadecimal number system is one with a base of 16. The base of 16 suggests choices of 16 single character digits or symbols. The first 10 digits are the digits of a decimal system 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The remaining six digits are denoted by A, B, C, D, E, F representing the decimal values 10, 11, 12, 13, 14, 15 respectively. In the hexadecimal number system, therefore, the letters A through F are number digits. The number A has a decimal equivalent value of 10 and the hexadecimal F has a decimal equivalent value of 15. Therefore, the largest single digit is F or 15 (one less than the base). Again each position in a hexadecimal system represents a power of the base (16).

Example: The decimal equivalent of the hexadecimal number $(1AF)_{10}$ is as follows:

$$(1AF)_{10} = 1\times16^2 + A\times16^1 + F\times16^0$$

$$= (1\times256+10\times16+15\times1)_{10}$$

$$= (431)_{10}$$

Converting from one number system to another: Numbers expressed in decimal are much more meaningful to us than are values expressed in any other number system. This is mainly because of the fact that we have been using decimal numbers in our day-to-day life from childhood. However, any number value in one number system can be represented in any other number system. Because the input and the final output values are to be in decimal, computer professionals are often required to convert numbers in other number systems to decimal and vice-versa.

Conversion from any other system to decimal: The following three steps are used to convert to a base 10 value from any other number system:

Step-I: Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step-2: Multiply the obtained column values (in Step-I) by the digits in the corresponding columns.

Step-3: Sum the products calculated in Step-2. The total is the equivalent value in decimal.

Example: Now we consider the binary number:

1011 =

----- $1 * 2^0 = 1$

------ $1 * 2^1 = 2$

------- $0 * 2^2 = 0$

-------- $1 * 2^3 = 8$

----

11 (in dcimal)

Example: Convert 176 in octal to decimal.

Each column represents a power of 8

----- $6 * 8^0 = 6$

------ $7 * 8^1 = 56$

------- $1 * 8^2 = 64$

----

126 (in dcimal)

Conversion from decimal to another: The following four steps are used to convert a number from decimal to a system with base *r*:

Step-1: Divide the decimal number to be converted by the value of the base *r*.

Step-2: Record the remainder from Step-1 as the rightmost digit (least significant digit) of the new base number).

Step-3: Divide the quotient of the previous divide by the base *r*.

Step-4: Record the remainder from Step-3 as the next digit (to the left) of the base *r*.

Repat Steps-3 and -4, recording remainders from right to left, until the quotient becomes zero in Step-3. The last remainder thus obtained will be the most significant digit of the *r* - base number.

Converting Decimal to Binary: $(254)_{10} = (?)_2$

The base of Binary number system is 2. So in this case $r = 2$ and the process is as follows:

254/2 giving 127 with a remainder of 0

127/2 giving 63 with a remainder of 1

63/2 giving 31 with a remainder of 1

31/2 giving 15 with a remainder of 1

15/2 giving 7 with a remainder of 1

7/2 giving 3 with a remainder of 1

1/2 giving 0 with a remainder of 1

Thus the binary equivalent is 11111110.

Converting Decimal to Hexadecimal: Convert 232 decimal to hexadecimal. Use the same method mentioned earlier to divide decimal to binary, but divide by 16.

232/16 = 14 with a remainder of 8

14/16 = 0 with a remainder of B (14 decimal = E)

Therefore, $(232)_{10} = (E8)_{10}$

Converting from a base other than 10 to a base other 10: The following two steps are used to convert a number from a base other than 10 to a base other 10:

Step-1: Convert the original number to a decimal number (base 10).

Step-2: Convert the decimal number so obtained to the new base.

Example: $(101110)_2 = (?)_8$.

Step - 1 : Convert $(101110)_2 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$

$= (32+0+8+4+2+0)_{10}$

$= (46)_{10}$

Step - 2 : Convert $(46)_{10}$ to base 8.

8	46	Remainders
	5	6
	0	5

Hence $(46)_{10} = (56)_8$

Therefore, $(101110)_2 = (56)_8$.

4.1.1 Addition and subtraction of binary numbers:

Addition: Binary addition is performed in the same manner as decimal addition. However, since binary system has only two digits, the addition table for binary arithmetic is very simple consisting of only four entries. The complete table for binary addition is as follows:

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1.+ 1 = 0 *plus a carry of* 1 *to next higher column.*

Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried over. By repeated use of the above rules, any two binary numbers can be added together by adding two bits at a time. The procedure is illustrated with the following example.

Example-1: Add the binary numbers 10011 and 1001 in binary form.

	Binary		Decimal
Carry	11	Carry	1
	10011		19
	+1001		+9
	11100		28

Note that while adding the first and the second column of the binary example, a carry is generated.

Example-2: Add the binary number 100111 and 11011 in binary form.

	Binary		Decimal
Carry	11111	Carry	1
	100111		39
	+11011		+27
	1000010		66

In this example, we face a new situation (1+1+1) brought about by the carry-over of 1 in the second column. This can be handled using the same four rules for binary addition. The addition of three 1's can be broken up into two steps. First we add only two 1's giving (1+1=10). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). So we conclude that 1+1+1=1 plus a carry of 1 to the next higher column.

Subtraction: The principle of subtraction consists of two steps, which are repeated for each column of the numbers.

Step-1: The first step is to determine if it is necessary to borrow. If the subtracted (the lower digit) is larger than the minuend (the upper digit), it is necessary to borrow from the column to the left. It is important to note here that the value borrowed depends upon the base of the number and is always the decimal equivalent of the base. Therefore, in decimal, binary, octal and hexadecimal 10, 2, 8 and 16 are borrowed respectively.

Step-2: The second step is simply to subtract the lower value from the upper value.

The complete table for binary subtraction is as follows:

0-0=0

1-0=1

1-1=0

0-1=1 with a borrow from the next column.

Observe that the only case in which it is necessary to borrow is when 1 is subtracted from 0.

Example: Subtract the number $(01110)_2$ from number $(10101)_2$ in binary form.

```
Borrow { 12
       { 0202
         10101
        -01110
        -------
         00111
```

In the first column, 0 is subtracted from 1. No borrow is required in this case and the result is 1. In the second column, we have to subtract 1 from 0. From the binary subtraction process, we know that a borrow is necessary to perform this subtraction. So a 1 is borrowed from the third column which becomes 2 in the second column because the base is 2. A 1 in the 4s column is equal to 2 in the 2s column. Now, in the second column, we subtract 1 from 2 giving a result of 1. The borrow performed in the second column reduces the 1 in the third column to 0. So in the third column, once again we have to subtract 1 from 0 for which borrow is required. The fourth column contains a 0 and thus has nothing to borrow. Therefore, we have to borrow from the fifth column. Borrowing 1 from the fifth column gives 2 in the fourth column. A 1 in the 16s column equals 2 in the 8s column. Now the fourth column has something to borrow. When 1 of the 2 in the fourth column is borrowed, it becomes 2 in the third column. Now, in the third column, we subtract 1 from 2 giving a result of 1. The borrow performed in the third column reduces the 1 in the fifth column to 0 and the 2 in the fourth column to 1. Hence subtraction of the fourth column is now 1 from 1, giving 0 and in the fifth column, subtraction is 0 from 0, giving 0. Thus the final result of subtraction is $(00111)_2$.

4.2 Boolean Algebra:

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called **Boolean algebra.** In 1938 C.E. Shannon introduced a two-valued Boolean algebra called switching algebra, in which he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra. Boolean algebra is an algebraic structure defined on a set of elements B together with two binary operations, + *and*, provided the following (Huntington) postulates are satisfied.

(i) (a) Closure with respect to the operator +.

(b) Closure with respect to the operator..

(ii) (a) An identity element with respect to +, designated by 0;

$x+0=0+x=x.x.\in B$

(b) An identity element with respect to., designated by 1:

$x.1=1.x=x,x,\in B$

(iii) (a) Commutative with respect to + : $x+y=y+x,x,y\in B$

(b) Commulative with respect to : $x.y=y.x,x,y\in B$

(iv) (a) . is distributive over + : $x.(y+z)=(x.y)+(x.z),x,y,z\in B$

(b) + is distributive over . : $x+(y.z)=(x+y).(x+z),x,y,z\in B$

(v) For every element $x\in B$, there exists an element $x'\in B$ (called the complement of x) such that : (a) $x+x'=1$ and (b) $x.x'=1$.

(vi) There exists at least two elements $x,y\in B$ such that $x\neq y$.

Differences between Boolean algebra and ordinary algebra: Comparing Boolean algebra with arithmetic and ordinary algebra (the field of real numbers), we note the following differences:

(i) Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.

(ii) The distributive law of + over., i.e., $x+(y.z)=(x+y).(x+z)$, is valid for Boolean algebra, but not for ordinary algebra.

(iii) Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtractionor division operations.

(iv) Postulate (v) defines an operator called **complement** which is not available in ordinary algebra.

(v) Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements B, but in the two-valued Boolean algebra defined below, B is defined as a set with only two elements 0 and 1.

Two-valued Boolean algebra: One can formulate many Boolean algebras; depending on the choice of elements of B and the rules of operation. Now we deal only with a two-valued Boolean algebra. A two-valued Boolean algebra is defined on a set of two elements, $B = \{0,1\}$, with rules for the two binary operators + and . defined as below:

x y	$x.y$
0 0	0
0 1	0
1 0	0
1 1	1

x y	$x+y$
0 0	0
0 1	1
1 0	1
1 1	1

x	x'
0	1
1	0

These rules are exactly the same as the *AND*, *OR* and *NOT* operations respectively. Now we show that the Huntington postulates are valid for the set $B = \{0,1\}$ and the two binary operators defined above.

(i) Closure is obvious from the tables since the result of each operation is either 1 or 0 where $1,0 \in B$.

(ii) From the tables we see that (a) $0 + 0 = 0,\ 0 + 1 = 1 + 0 = 1$

(b) $1.1 = 1,\ 1.0 = 0.1 = 0$

Which establishes the two identity elements 0 for '+' and 1 for '.' as defined by postulates (ii).

(iii) The commutative laws are obvious from the symmetry of the binary operator tables.

(iv) (a) The distributive law $x.(y+z) = (x.y)+(x.z)$ holds true by the following truth table:

xyz	y+z	x.(y+z)	x.y	x.z	(x.y)+(x.z)
000	0	0	0	0	0
001	1	0	0	0	0
010	1	0	0	0	0
011	1	0	0	0	0
100	0	0	0	0	0

101	1	1	0	1	1
110	1	1	1	0	1
111	1	1	1	1	1

(b) distributive law of '+' over '.' can be shown to hold true similarly by truth table.

(v) From the complement table it is easily shown that

(a) $x+x'=1$, since $0+0'=0+1=1$ and $1+1'=1+0=1$

(b) $x.x'=0$, since $0.0'=0.1=1$ and $1.1'=1.0=1$ which verifies the postulate (v).

(vi) Postulate (vi) is satisfied because the two-valued Boolean algebra has two distinct elements 1 and 0 with $1 \neq 0$.

4.2.1 Basic Theorems: There is one principle and six basic theorems in Boolean algebra, which are discussed as follows:

Principle: Duality

In states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged. In a two-valued Boolean algebra, the identity elements and the elements of the set B are the same: 1 and 0. The duality principle has many applications. If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

Theorem-1(a): $x+x=x$

Proof:

$x+x=(x+x).1$ *by postulate* (iib)

$=(x+x)(x+x')$ *by postulate*(va)

$=x+xx'$ *by postulate*(ivb)

$=x.1$ *by postulate* (va)

$=x$ *by postulate* (iib)

Theorem-1(b): $x.x = x$

Proof:

$x.x = x.x + 0$ *by postulate* (iia)

$= x.x + x.x'$ *by postulate* (vb)

$= x(x + x')$ *by postulate* (iva)

$= x.1$ *by postulate* (va)

$= x$ *by postulate* (iib)

Theorem-2(a): $x + 1 = 1$

Proof:

$x + 1 = 1.(x + 1)$ *by postulate* (iib)

$= (x + x')(x + 1)$ *by postulate* (va)

$= x + x'.1$ *by postulate* (ivb)

$= x + x'$ *by postulate* (iib)

$= 1$ *by postulate* (va)

Theorem-2(b): $x.0 = 0$

Proof: It can be proved by the duality principle easily.

Theorem-3: $(x')' = x$

Proof: From postulate (v), we have $x + x' = 1$ and $x.x' = 0$, which defines the complement of x. The complement of x' is x and is also $(x')'$. Therefore, since the complement is unique, we have that $(x')' = x$.

Theorem-4(a): $x + (y + z) = (x + y) + z$

Proof: it can be proved easily by truth table.

Theorem-4(b): $x.(y.z) = (x.y).z$

Proof: It can be proved easily by truth table.

x y	$(x + y)$	$(x + y)'$	x'	y'	$x'y'$
0 0	0	1	1	1	1
0 1	1	0	1	0	0

1 0	1	0	0	1	0
1 1	1	0	0	0	0

Theorem-5(b): $(x.y)' = x' + y'$

Proof: It can be proved easily by truth table.

Theorem-6(a): $x + xy = x$

$x + xy = x.1 + xy$ *by postulate* (iib)

$= x(1 + y)$ *by postulate* (iva)

$= x(y + 1)$ *by postulate* $(iiia)$

$= x.1$ *by postulate* (iia)

$= x$ *by postulate* (iib)

Theorem-6(b): $x(x + y) = x$

Proof: It can be proved by the principle of duality.

4.2.2 Boolean Function and Truth Table:

A binary variable can take the value of 0 or 1. A Boolean function is an expression formed with binary variables, the two binaryoperators OR and AND, the unary operator NOT, parentheses, and equal sign. For a given value of the variables, the function can either 0 or 1.

Example: We consider the Boolean function: $F_1 = xyz'$. The function F_1 is equal to 1 if $x = 1$ and $y = 1$ and $z' = 1$; otherwise $F_1 = 0$.

Any Boolean function can also be represented in a truth table. To represent a function in a truth table, we need a list of the 2^n combinations of 1's and 0's of the n binary variables, and a column showing the combinations for which the function is equal to 1 or 0. So the number of rows in the table is 2^n, where n is the number of binary variables in the function. The 1's and 0's combinatinos for each row is easily obtained from the binary numbers by counting from 0 to $2^n - 1$.

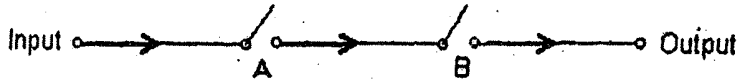
Example: For the above example, there are eight possible distinct combinations for assigning bits to three variables. The column labeled F_1 contains either a 0 or a 1 for each of these combinations. The table shows that the function F_1 is equal to 1 only when $x = 1, y = 1$ and $z = 0$. It is equal to 0 otherwise.

x	y	z	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

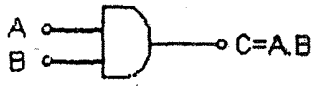
Logic Gates: All operations within a computer are carried out by means of combinations of signals passing through standard blocks of built-in circuits that are known as logic gates. In other words, a logic gate is simply an electronic circuit which operates on one or more input signals to produce standard output signals. These logic gates are building blocks of all the circuits in a computer.

Computer circuits are built up using combinations of different types of logic gates to perform the necessary operation. There are several types of gates, but since Boolean functions are expressed in terms of AND, OR and NOT operations, it is easier to implement a Boolean function with these types of gates. These gates are discussed below:

ANG Aate: An AND gate is the physical realization of the logical multiplication (AND) operation. That is, it is an electronic circuit that generates an output signal of 1 only if all input signals are also 1. To have a conceptual idea, we consider the following figure. Here two switches A and B are connected in series. It is obvious that the input current will reach the output point only when both the switches are in the on (1) state. There will be no output (output=0) if either one or both the switches are in the off (0) state. So, two or more switches connected in series behave as an AND gate.

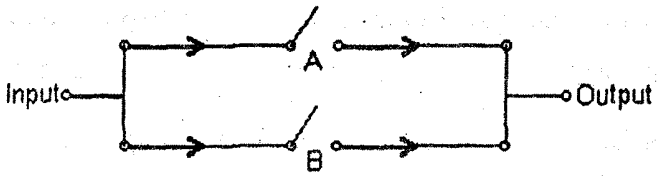


The behaviour of a logic gate, that is the state of its output signal depending on the various combinations of input signals, is conveniently described by means of a truth table. The truth table and the block diagram symbol for an AND gate for two input signals are shown below. Since there are only two inputs (A and B), so only four (2^2) combinations of inputs are possible.

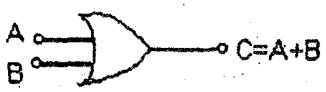


Inputs		Output
A	B	C=A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate: An OR gate is the physical realization of the logical addition (OR) operation. That is, it is an electronic circuit that generates an output signal of 1 if any of the input signals is also 1. Two or more switches connected in parallel behave as an OR gate. From the following figure, it can be seen that the input current will reach the output when any one of the two switches are in the on (1) state. There will be no output only when both the switches (A and B) are in the off (0) state.

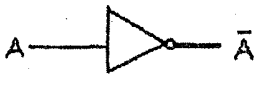


The truth table and the block diagram for an OR gate for two input signals are given as follows. Observe that an output of 1 is obtained when any of the input signals is 1. Output is 0 only when both the inputs are 0.



Inputs		Output
A	B	C=A+B
0	0	0
0	1	1
1	0	1
1	1	1

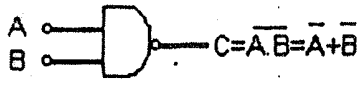
NOT Gate: A NOT gate is the physical eralization of the complementation operation. That is, it is an electronic circuit that generates an output signal which is the reverse of the input signal. A NOT gate is also known as an inverter because it inverts the input. The truth table and the block diagram for an NOT gate are shown in the following figure. It is a unary operation which is defined on a single variable. Hence a NOT gate always has a single input.



Input	Output
A	$\bar{A}$
0	1
1	0

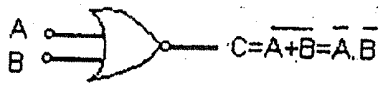
The Universal Gates: AND, OR and NOT gates are logically complete gates in the sense that any Boolean function may be implemented using these three gates. But there is also another gate by which any Boolean function may be realized by alone. In this sense this gate is called Universal gate. The two universal gates are NAND and NOR.

NAND gate: A NAND gate is a complemented AND gate. That is, the output of NAND gate will be a 1 if any one of the inputs is a 0 and will be a 0 only when all the inputs are 1. The truth table and the block diagram for a NAND gate are given below.



Inputs		Output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate: A NOR gate is a complemented OR gate. That is, the output of a NOR gate will be a 1 only when all inputs are 0 and it will be a 0 if any input represent a 1. The truth table and the block diagram are shown below:



Inputs		Output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Algebraic Manipulation: A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR, and NOT gates. A literal is a primed or unprimed variable. When a Boolean function is implemented with logic gates, each literal in the function designates an input to a gate, and each term is implemented with a gate. *The minimization of the number of literals and the number of terms results in a circuit with less equipment.* It is not always possible to minimize both simultaneously; usually, further criteria must be available. The number of literals in a Boolean function can be minimized by algebraic manipulations. Unfortunately, there are no specific rules to follow that will guarantee the final answer. The only method available is a cut-and-try procedure employing the postulates, the basic theorems.

Example: Simplify the Boolean function $xy + x'z + yz$ to a minimum number of literals.

$$xy + x'z + yz = xy + x'z + yz(x + x')$$

$$= xy + x'z + xyz + x'yz$$

$$= xy(1 + z) = x'z(1 + y)$$

$$= xy + x'z$$

Complement of a Function: The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through De Morgan's theorem. De Morgan's theorems can be extended to three or more variables. A simple procedure for deriving the complement of a function is to the dual of the function and complement of each literal. Remember that the dual of a function is obtained from the interchange of AND and OR operators and 1's and 0's.

Example: Find the complement of the function $F = x'yz' + x'y'z$.

$$F = x'yz' + x'y'z$$

The dual of F is $(x'+y+z')(x'+y'+z)$

The complement of each literal: $(x+y'+.z)(x+y+z') = F'$

4.2.3 Canonical Form of Boolean algebra:

Minterms: A binary variable may appear either in its normal form (x) or in its complement form (x'). Now we consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: $x'y', x'y, xy'$ and xy. Each of these four AND terms is called a **minterm** or a **standard product**. In similar manner, n variables can be combined to form 2^n minterms. The 2^n different minterms may be determined by a method as follows:

The binary numbers of 0 to $2^n - 1$ are listed under the n variables. Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1. A symbol for each minterm is also represented by m_j, where j denotes the decimal equivalent of the binary number of the minterm designated.

Example:

x	y	z	Minterms	
			Term	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

Maxterm: In a similar way, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called **maxterms** or **standard sums**. Any 2^n maxterms for n variables may be determined similarly. Each maxterm is obtained from an OR term of the n variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1. Each maxterm is presented by the symbol M_j, where j is

the decimal equivalent of the binary number of the maxterm. The eight maxterms for three variables, together with their symbolic designation, are listed in the following table.

Example:

x	y	z	Minterms	
			Term	Designation
0	0	0	$x+y+z$	M_0
0	0	1	$x+y+z'$	M_1
0	1	0	$x+y'+z$	M_2
0	1	1	$x+y'+z'$	M_3
1	0	0	$x'+y+z$	M_4
1	0	1	$x'+y+z'$	M_5
1	1	0	$x'+y'+z$	M_6
1	1	1	$x'+y'+z'$	M_7

Formation of a Boolean function from a truth table: A Boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables which produces a 1 in the function, and then taking the OR of all those terms.

Example: The function F in the following table is determined by expressing the combinations 001, 100, and 111 as $x'y'z, xy'z'$ and xyz respectively. Since each one of these minterms results in $F = 1$, we shuold have

$$F = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

x	y	z	Function F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

This example demonstrates an important property of Boolean algebra: *Any Boolean function can be expressed as a sum of minterms (by "sum" is meant the ORing of terms).*

Complement of a Boolean Function: Now we consider the complement of a Boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms.

Example: The component of F in the above example is read as:

$$F' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of F', we obtain the function F as follows:

$$F = (x+y+z)+(x+y'+z).(x+y'+z').(x'+y+z').(x'+y'+z)$$

$$= M_0M_2M_3M_5M_6$$

This example demonstrates a second important property of Boolean algebra: *Any Boolean function can be expressed as a product of maxterms (by "product" is meant the ANDing of terms).* The procedure for obtaining the product of maxterms directly from the truth table is as follows. Form a maxterm for each combination of the variables which produces a 0 in the function, and then form the AND of all those maxterms.

Canonical form: Boolean functions expressed as a sum of minterms of product of maxterms are said to be in **canonical form.**

Example-1: Express the Boolean function $F = A + B'C$ in a sum of minterms.

The function has three variables A, B and C. The first term C is missing two variables; therefore we have

$$A = A(B+B') = AB + AB'$$

This is still missing one variable:

$$A = AB(C+C')+AB'(C+C')$$

$$= ABC + ABC' + AB'C + AB'C'$$

The second term $B'C$ is missing one variable:

$$B'C = B'C(A+A') = AB'C + A'B'C$$

Combining all terms, we have

$$F = A + B'C$$

$$= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C$$

But $AB'C$ appears twice, and according to theorem $(x + x = x)$, it is possible to remove one of them. Rearrange the minterms in ascending order, we finally obtain:

$$F = A + B'C$$

$$= A'B'C + AB'C' + AB'C + AB'C + ABC$$

$$= m_1 + m_4 + m_5 + m_6 + m_7$$

$$= \sum(1,4,5,6,7)$$

Example-2: Express the Boolean function $F = xy + x'z$ in a product of maxterm form.

First convert the function into OR terms using the distributive law:

$$F = xy + x'z = (xy + x')(xy + z)$$

$$= (x + x')(y + x')(x + z)(y + z)$$

$$= (x' + y)(x + z)(y + z)$$

The function has three variables: x, y and z. Each OR term is missing one variable; therefore;

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$

$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$

$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all the terms and removing those that appear more than once, we finally obtain:

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$

$$= M_0 M_2 M_4 M_5$$

$$= \therefore F = \prod(0,2,4,5)$$

Conversion between Canonical forms: The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, while its complement is a 1 for those minterms that the function is a 0.

Example: We consider the function $F(A,B,C) = \sum(1,4,5,6,7)$. This has a complement that can be expressed as

$$F'(A,B,C) = \sum(0,2,3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by De Morgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' m_2' m_3' = m_0 m_2 m_3 = \prod(0,2,3)$$

4.3 Algorithm and Flow-Chart:

Algorithm:

An algorithm is a finite set of instructions which, if followed, accomplish a particular task. In addition every algorithm must satisfy the following criteria:

(i) input: there are zero or more quantities which are externally supplied;

(ii) output: at least one quantity is produced;

(iii) definiteness: each instruction must be clear and unambiguous;

(iv) finiteness: if we trace out the instructions of an algorithm, then for all cases the algorithm with terminate after a finite number of steps;

(v) effectiveness: every instruction must be sufficiently basic that it can in principle be carried out a person using only pencil and paper. It is not enough that each operation be definite as in (iii), but it must also be feasible.

In formal computer science, one distinguishes between an algorithm and a program. A program does not necessarily satisfy the condition (iv). One important example of such a program for a computer is its operating system which never terminates (except for a system crashes) but continues in a wait loop until more jobs are entered. An algorithm can be described in many ways. A natural language such as English can be used but we must be very careful that the resulting instructions are definite (condition (iii)).

Flowchart:

A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. Solid lines having arrow mark to indicate the flow of operation, that is the exact sequence in which the instructions are to be executed, connect these boxes. Therefore, a flowchart is a picture of the logic to be included in the computer program. It is simply a method of assisting the programmer to layout, in a visual, two-dimensional format, ideas on how to organize a sequence of steps necessary to solve a problem by a computer. It is basically the plan to be followed when the program is written.

4. Computer fundamental:

4.4.1 Bit, Byte, Nibble and Word:

the Bit: The smallest "unit" of data on a binary computer is a single bit. Since a single bit is capable of representing only two different values (typically zero or one) you may get the impression that there are a very small number of items you can represent with a single bit. Not true! There are an infinite number of items you can represent with a single bit. With a single bit, you can represent any two distinct items. Examples include zero or one, true or false, on or off, male or female, and right or wrong. However, you are not limited to representing binary data types (that is, those objects which have only two distinct values).

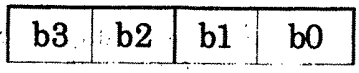
To confuse things even more, different bits can represent different things. For example, one bit might be used to represent the values zero and one, while an adjacent bit might be used to represent the values true and false. How can you tell by looking at the bits? The answer, of course, is that you can't. But this illustrates the whole idea behind computer data structures: data is what you define it to be.

If you use a bit to represent a boolean (true/false) value then that bit (by your definition) represents true or false. For the bit to have any true meaning, you must be consistent. That is, if you're using a bit to represent true or false at one point in your program, you shouldn't use the true/false value stores in that bit to represnt red or blue later.

Since most items you will be trying to model require more than two different values, single bit values aren't the most popular data type. However, since everything else consists of groups of bits, bits will play an important role in your programs. Of course, there are several data types that require two distinct values, so it would seem that bits are important by themselves. However, you will soon see that individual bits are difficult to manipulate, so we'll often use other data types to represent boolean values.

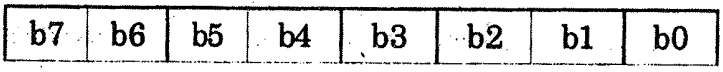
The Nibble: A nibble is a collection of bits on a 4-bit boundary. It wouldn't be a particularly interesting data structure except for two items: BCD (binary coded decimal) numbers and hexadecimal (base 16) numbers. It takes four bits to represent a single BCD or hexadecimal digit. With a nibble, we can represent up to 16 distinct values. In the case of hexadecimal numbers, the values 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F are represented with four bits. BCD uses ten different digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) and requires four bits. In fact,

any sixteen distinct values can be represented with a nibble, but hexadecimal and BCD digits are the primary items we can represent with a single nibble.



The Byte: Without question, the most important data structure used by the 80 x 86 microprocessor is the byte. This is true since the ASCII code is a 7-bit non-weighted binary code that is used on the byte boundary in most computers. A byte consists of eight bits and is the smallest addressable datum (data item) in the microprocessor.

Main memory and I/O addresses in teh PC are all byte addresses. This means that the smallest item that can be individually accessed by an 80 x 86 program is an 8-bit value. To access anything smaller requires that you read the byte containing the data and mask out the unwanted bits. The bits in a byte are numbered from bit zero (b0) through seven (b7) as follows:



Bit 0 is the low order bit or least significant bit; bit 7 is the high order bit or most significant bit of the byte. We'll refer to all other bits by their number.

A byte also contains exactly two nibbles. Bits b0 through b3 comprise the low order nibble, and bits b4 through b7 form the high order nibble. Since a byte contains exactly two nibbles, byte values require two hexadecimal digits.

Since a byte contains eight bits, it can represent 2^8, or 256, different values. Generally, we'll use a byte to represent.

Unsigned numeric values in the range: *From* 0 *to* 255

Signed numbers in the range: *From* –128 *to* 127

ASCII character codes

Other special data types requiring no more than 256 different values. Many data types have fewer than 256 items so eight bits is usually sufficient.

Since the PC is a byte addressable machine, it turns out to be more efficient to manipulate a whole byte than an individual bit or nibble. For this reason, most programmers use a whole byte to represent data types that require no

more than 256 items, even if fewer than eight bits would suffice. For example, we'll often reprsent the boolean values true and false by 00000001 and 00000000 (respectively).

Probably the most important use for a byte is holding a character code. Characters typed at the keyboard, displayed on the screen, and printed on the printer all have numeric values. To allow it to communicate with the rest of the world, the IBM PC uses a variant of the ASCII character set. There are 128 defined codes in the ASCII character set. IBM uses the remaining 128 possible values for extended character codes including European characters, graphic symbols, Greek letters, and math symbols.

The Word: For the 8085 and 8086, a word is a group of 16 bits. We will number the bits in a word starting from bit zero (b0) through fifteen (b15) as follows:

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0

Like the byte, bit 0 is the LSB and bit 15 is the MSB. When referencing the other bits in a word use their bit position number.

Notice that a word contains exactly two bytes. Bits b0 through b7 form the low order byte, bits 8 through 15 form the high order byte. Naturally, a word may be further broken down into four nibbles. Nibble zero is the low order nibble in the word and nibble three is the high order nibble of the word. The other two nibbles are "nibble one" or "nibble two".

With 16 bits, you can represent 2^16 (65,536) different values. These could be the unsigned numeric values in the range of 0 => 65,535, signed numeric values in the range of –32,768 => + 32,767, or any other data type with no more than 65,536 values. The three major uses for words are

16-bit integer data values

16-bit memory addresses

Any number system requiring 16 bits or less

The Double Word: A double word is exactly what its name implies, two words. Therefore, a double word quantity is 32 bits. Naturally, this double word can be divided into a high order word and a low order word, four bytes, or eight nibbles. Double words can represent all kinds of different data. It may be

An undersigned double word in the range of: 0 to 4,294,967,295

A signed double word in the range -2, 147, 483, 648 => 2,147,483,647,

A 32-bit floating point value: -2, 147, 483, 648 to 2,147,483,647

Any data that requires 32 bits or less

Note: The boundary for a Word is defined as either 16-bits or the size of the data bus for the processor, and a Double Word is Two Words. Therefore, a Word and a Double Word is not a fixed size but varies from system to system depending on teh processor. However, for one discussion, we will define a word as two bytes.

4.4.2 Basic Structure of Computer:

4.4.2.1 I/O Units:

Data and instructions must enter the computer system before any computation can be performed on the supplied data. This task is performed by the input unit. For example, data is entered from a keyboard. The following functions are performed by an input unit:

- It accepts the instructions and data from the outside of the computer.
- It converts these instructions and data in computer acceptable form.
- It supplies the converted instructions and data to computer system for further processing.

The function of an output unit is just the reverse of that of an input unit. It supplies information and results of computation from the computer system to the outside world. The following functions are performed by this unit.

- It accepts the results produced by the computer which are in coded form and hence cannot be easily understood by us.
- It converts these coded results to human acceptable form.
- It supplies the converted results to the outside of the computer.

4.4.2.2 ALU:

The arithmetic logic unit (ALU) of a computer system is the device where the actual execution of the instructions takes place during the processing operation. All calculations are performed and all comparisons (decisions) are

made in the ALU. The type and number of arithmetic and logic operations that a computer can perform is determined by the engineering design of the ALU.

4.4.2.3 CU:

The control unit is able to maintain order and direct the operation of the entire system. Although, it does not perform any actual processing on the data, the control unit acts as central nervous system for the other components of the computer. It manages and coordinates the entire computer system. It obtains instructions from the program stored in the main memory, interprets the instructions, and issues signals that cause other units of the system 'to execute them.

4.4.2.4 Memory Unit:

The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. Moreover, the intermediate results produced by the computer must also be preserved for ongoing processing. The memory unit is designed to perform all these needs. It provides space foe storing data and instructions, space for intermediate results, and also space for the final results.

4.4.2.5 Peripheral Devices:

A computer device, such as a CD-ROM or printer, that is not part of the essential computer, i.e., the memory and microprocessor. Peripheral devices can be external — such as a mouse, keyboard, printer, monitor, external Zip drive or scanner — or internal, such as a CD-ROM drive, CD-R drive or internal modem. Internal peripheral devices are often referred to as integrated peripherals.

4.4.3 Different types of I/O Units:

Line Printer:

A high-speed printer capable of printing an entire line at one time. A fast line printer can print as many as 3,000 lines per minute. The disadvantages of line printers are that they cannot print graphics, the print quality is low, and they are very noisy.

Dot-matrix printer:

A type of printer that produces characters and illustrations by striking pins against an ink ribbon to print closely spaced dots in the appropriate shape. Dot-matrix printers are relatively expensive and do not produce high-quality output. However, they can print to multi-page forms (that is, carbon copies), something laser and ink-jet printers cannot do.

Dot-matrix printers vary in two important characteristics:

- **speed**: Given in *characters per set-mid (cps)*, the speed can vary from about 50 to over 500 cps. Most dot-matrix printers offer different speeds depending on the quality of print desired.
- **print quality**: Determined by the number of pins (the mechanisms that print the dots), it can vary from 9 to 24. The best dot-matrix printers (24 pin) can produce near letter-quality type, although you can still see a difference if you look closely.

Laser Printer:

A type of printer that utilizes a laser beam to produce an image on a drum. The light of the laser alters the electrical charge on the drum wherever it hits. The drum is then rolled through a reservoir of toner, which is picked up by the charged portions of the drum. Finally, the toner is transferred to the paper through a combination of heat and pressure. This is also the way copy machines work.

Because an entire page is transmitted to a drum before the toner is applied, laser printers are sometimes called page printers. There are two other types of page printers that fall under the category of *laser printers* even though they do not use lasers at all. One uses an array of *LEDs* to expose the drum, and the other uses LCDs. Once the drum is charged, however, they both operate like a real laser printer.

One of the chief characteristics of laser printers is their resolution — how many dots (dpi) they lay down. The available resolutions range from 300 dpi at the low end to 1,200 dpi at the high end. By comparison, offset printing usually prints at 1,200 or 2,400 dpi. Some laser printers achieve higher resolutions with special techniques known generally as resolution enhancement.

In addition to the standard monochrome laser printer, which uses a single toner, there also exist color laser printers that use four toners to print in full color. Color laser printers tend to be about five to ten times as expensive as their monochrome siblings.

Laser printers produce very high-quality print and are capable of printing an almost unlimited variety of fonts. Most laser printers come with a basic set of fonts, called internal or resident fonts, but you can add additional fonts in one of two ways:

- font cartridges : Laser printers have slots in which you can insert font cartridges, ROM boards on which fonts have been recorded The advantage of font cartridges is that they use none of the printer's memory.
- soft fonts : All laser printers come with a certain amount of RAM memory, and you can usually increase the amount of memory by adding memory boards in the printer's expansion slots. You can then copy fonts from a disk to the printer's RAM. This is called downloading fonts. A font that has been downloading is often referred to as a soft font, to distinguish it from the hard fonts available on font cartridges. The more RAM a printer has, the more fonts that can be downloaded at one time.

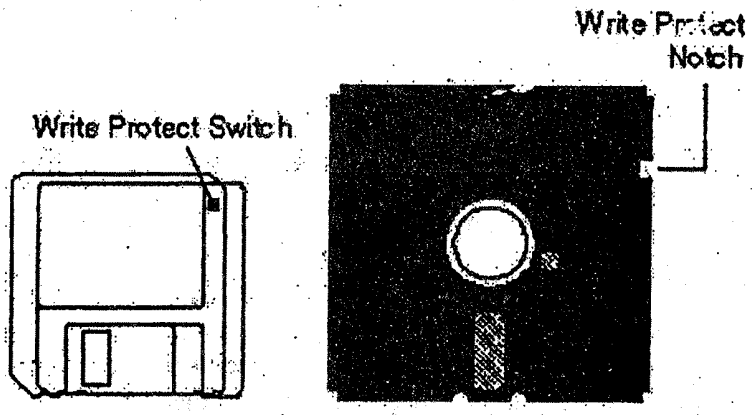
In addition to text, laser printers are very adept at printing graphics. However, you need significant amounts of memory in the printer to print high-resolution graphics. To print a full-page graphic at 300 dpi, for example, you need at least I MB (megabytes) of printer RAM. For a 600-dpi graphic, you need at least 4 MB RAM.

Because laser printers are *nonimpact* printers, they are much quieter than dot matrix or daisy wheel printers. They are also relatively fast, although not as fast as some dot-matrix printers. The speed of laser printers ranges from about 4 to 20 pages of text per minute (ppm). A typical rate of 6 ppm is equivalent to about 40 characters per second (cps).

Floppy disk:

A soft magnetic disk. It is called *floppy* because it flops if you wave it (at least, the 5 1/4-inch variety does). Unlike most hard disks, floppy disks (often called *floppies* or *diskettes*) are portable, because you can remove them from a disk drive. Disk drives for floppy disks are called *floppy drives*. Floppy disks are slower to access than hard disks and have less storage capacity, but they are much less expensive. And most importantly, they are portable.

Write Protect Notch



Floppies come in three basic sizes:

- **8-inch**: The first floppy disk design, invented by IBM in the late 1960s and used in the early 1970s as first a read-only format and then as a read-write format. The typical desktop/laptop computer does not use the 8-inch floppy disk.
- **5¼-inch**: The common size for PCs made before 1987 and the predecessor to the 8-inch floppy disk. This type of floppy is generally capable of storing between 100K and 1.2MB (megabytes) of data. The most common sizes are 360K and 1.2MB.
- **3½-inch**: *Floppy* is something of a misnomer for these disks, as they are encased in a rigid envelope. Despite their small size, microfloppies have a larger storage capacity than their cousins from 400K to IAMB of data. The most common sizes for PCs are 720K (double-density) and 1.44MB (high-density).

Winchester Disk:

Another term for hard disk drive. The term *Winchester* comes from an early type of disk developed by IBM that had 30MB of fixed storage and 30MB of removable storage; so its inventors called it a Winchester in honor of its 30/30 rifle. Although modern disk drives are faster and hold more data, the basic technology is the same, so *Winchester* has become synonymous with *hard.*

4.6 Data Representation:

In this section we discuss the representation of data in digital system. Two main categories of data are (i) Numbers and (ii) Characters. Internally in a digital system all data are represented by equivalent strings of symbols where each symbol, called a bit, either a 0 or a 1. The primary reasons for choosing to represent all information using only zeros or ones are

(i) Physical devices used for operating on data in digital systems perform most reliably when operated in one out of two distinct states. For example, circuits designated with transistors operate with maximum reliability when used in the "two state", namely, binary mode.

(ii) Most devices which are currently available to store information do so by being in one out of two stable states. For example, magnetic disc store information by being magnetized in a specified direction or in an opposite direction.

4.6.1 Binary Coded Decimal Numbers:

We have already discussed the methods of converting decimal numbers to binary form and vice versa. There is another method of representing decimal numbers using binary digits. This method is called **binary coded decimal representation (BCD).** There are ten symbols in the decimal system, namely, 0,1,...,9. Encoding is the procedure of representing each one of these ten symbols by a unique string consisting of the two symbols of the binary system, namely, 0 and 1. It is further assumed that the same number of bits is used to represent any digit. The number of symbols which could be represented using n bits is 2^n. Thus in order to represent the ten decimal digits we require at least 4 bits as 3 bits will allow only 2^3=8 possible distinct 3-bit groups. The method of encoding decimal numbers in binary is to make up a table of 10 unique 4-bit groups and allocate one 4-bit group to each decimal digit as shown in the following table.

Decimal digit	Binary code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7.	0111
8	1000
9	1001

If we want to represent a decimal number, say, 15, using the code given in the above table we look up the table and get the binary code for I as 0001 and that for 5 as 0101 and code 15 by the binary code 00010101.

Difference between encoding and conversion: Now we distinguish carefully between encoding and conversion. For example, 15 when converted to binary would be 1111. On the other hand when it is encoded each digit must be represented by a 4-bit code and an encoding is 00010101. It should be observed that encoding requires more bits compared to conversion. On the average $\log_2 10 = 3.32$ bits are required when decimal numbers are converted to binary; as compared with 4 bits per digit are needed in encoding. The ratio (4/3.3)=1.2 is a measure of extra bits (and consequently extra storage) required if an encoding is used. On the other hand conversion of decimal to binary is slower compared to encoding. This is due to the fact that an algorithm involving successive division is needed for conversion whereas encoding is by straight forward table look-up. The slowness of conversion is not a serious problem in computations in which the volume of input/output is small. In business computers, where input/ output dominates, it is necessary to examine BCD representation. In smaller digital systems such as desk calculators, digital clocks, etc., it is uneconomical to incorporate complex electronic circuits to convert decimal to binary and vice versa. Therefore BCD representation should be considered.

So we need at least 4 bits to represent a decimal digit. There are, however, 16 four-bit groups. We need only 10 of these 16 for encoding decimal digits. There are 30 billion ways we can pick an ordered sequence of 10 out of 16 items. Therefore many codes can be constructed. Fortunately all these 3×10^{10} possible codes are not useful.

Only a small number of these are used in practice and they are chosen from the view point of (i) ease in arithmetic, (ii) some error detection property, (iii) ease in coding, and (iv) any other property useful in a given application. The useful codes may be divided broadly into 4 classes which are:

(i) Weighted codes

(ii) Self complementing codes

(iii) Cyclic, Reflected or Gray codes. and

(iv) Error detecting and correcting codes.

4.6.2 Weighted codes:

In a weighted code the decimal value of a code is the algebraic sum of the weights of those columns in which a 1 appears. In other words $d = \sum w_i b_i$ where w_i s is the weights and b_i s are either 0 or 1. Three examples of weighted codes are given below.

Decimal digit	Weight 8421	Weight* 8421	Weight 2421
0	0000	0000	0000
1	0001	0111	0001
2	0010	0110	0010
3	0011	0101	0011
4	0100	0100	0100
5	0101	1011	1011
6	0110	1010	1100
7	0111	1001	1101
8	1000	1000	1110
9	1001	1111	1111

* An over-bar is used to indicate a negative weight (2 = —2).

In a weighted code we may have negative weights. Further the same weight may be repeated as in 2, 4, 2, 1 code. **The criterion in choosing weights is that in using these weights we must be able to represent all the decimal digits from 0 through 9.** The 8, 4, 2, 1 code uses the natural weights used in binary number representation.

4.6.3 Self-complementing codes:

If a code is constructed such that when we replace a 1 by a 0 and a 0 by a 1 in the 4-bit code representation of a digit d we obtain the code for (9-d), it is called a self-complementing code. For example, the 2, 4, 2, 1 and the 8,4,2j codes are self-complementing. **A necessary condition for a self-complementing weighted code is that the sum of its weights be 9.**The following table gives a self-complementing code.

d	Code for d	Code for 9-d	9-d
	2421	2421	
0	0000	1111	9
1	0001	1110	8
2	0010	1111	7
3	0011	1100	6
4	0100	1011	5
5	1011	1100	4
6	1100	0011	3
7	1101	0010	2
8	1110	0001	1
9	1111	0000	0

A self-complementing code need not necessarily be weighted. An important un-weighted self-complementing code is the **Excess-3 code**. This code is derived from 8, 4, 2, 1 code by adding 0 0 1 1 (a three) to all code groups. The Excess-3 code is given below.

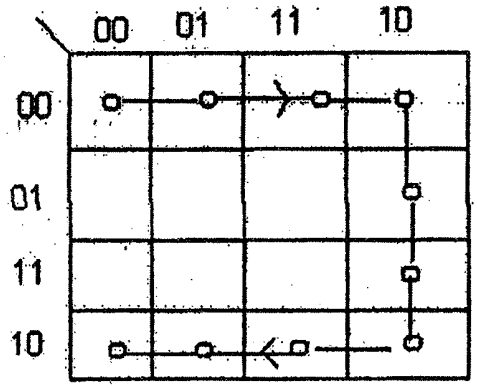
d	Excess-3 code
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

4.6.4 Cyclic codes:

Digital systems can be designed to process data in discrete form only. Many physical systems supply continuous output data. These data must be converted into digital or discrete form before they are applied to a digital system.

Continuous or analog information is converted into digital form by means of an analog-to-digital converter. It is sometimes convenient to use the cyclic code (Gray code/reflected code) to represent the digital data converted from the analog data. In a cyclic code each code group does not differ from its neighbor in more than one bit. To formalize this concept we will define what is known as the Hamming distance after its inventor R. W. Hamming. The Hamming distance between two equal length binary sequences of 1s and 0s is the number of positions in which they differ. For example, if A=0110 and B=1010, the Hamming distance between A and B is two as they differ in their first and second positions counting from the left.

Hamming distance between two successive code groups in a cyclic code is unity. In other words, each code is adjacent to the next in sequence. Consider the following map. Each square in this map represents a 4-bit code. For example, the square in the second row, third column represents the code 0111. This map is coded to ensure that each square is at a unit distance from its adjacent square. Corresponding squares in the top row are adjacent to those in the bottom row. For example, the fourth square in the top row is 0010 and the fourth square in the bottom row is 1010 and the Hamming distance between them is unity.



4.6.5 Error detecting codes:

Binary information may be transmitted through some form of communication medium such as wires or radio waves. Any external noise introduced into a physical communication medium changes bit values from 0 to 1 or vice-versa. An error-detection code can be used to detect errors during transmission. The detected error can not be corrected, but its presence is indicated.

The main principle used in constructing error detecting codes is to use redundant bits in codes with the specific purpose of detecting errors. The method used is to ensure that the Hamming distance between any two codes in

the set of codes is a pre-assigned minimum. If the minimum Hamming distance between any two codes is two, then if a single error occurs in one of the codes this can be detected because the corrupted code will be different from any of the codes allowed in the group.

One common method of constructing such codes is the introduction of an extra bit in the code. For example, we suppose that 0011 is the code for 8 in a coding scheme. We may introduce a fifth bit such that the total number of ones in this five bit group is odd. This bit is called the parity bit. If in a code the total number of ones is not odd then we can conclude that it has a single error. We can not detect two errors as the odd parity will be satisfied with two or any even 1 umber of errors. We can however detect any odd number of errors. An 8, 4, 2, 1 code with an added odd p9rity bit in the following table is shown. We may also introduce an extra bit in the code to make the total cumber of ones in the code-even. This is called an **even parity bit.**

Digit	8421 Odd parity bits	Non-Weighted code Even parity bits
0	00001	11000
1	00010	00011
2	00100	00101
3	00111	00110
4	01000	01001
5	01011	01010
6	01101	01100
7	01110	10010
8	10000	10010
9	10011	10100

4.6.6 Error correcting codes:

Coding schemes may be devised which not only detect errors but also automatically correct them. We consider, for example, 36 bits recorded on a magnetic tape in 6 tracks with 6 bits along each track. We suppose an extra code group is recorded on a seventh track and it is devised to give an odd parity on the columns. Further, we suppose that each code word has a seventh bit added as an odd parity bit. If any bit in the group is erroneously transmitted, it will cause simultaneous failure of parity on a row and a column. The row and the column automatically fix the position of the erroneous bit and it can be corrected.

4.6.7 Hamming Codes for error Correction:

The error detection is possible as the addition of the parity bits creates a minimum Hamming distance of two between any two codes. Hamming showed that by systematically introducing more parity bits in the code it is not only possible to detect an error but also find out where the error occurred and correct it. More than one error can also be detected and corrected by increasing the code length with more parity bits and thereby increasing the minimum Hamming distance between codes.

Now we examine how a Hamming code to detect and correct one error can be constructed. We suppose that we want to add parity bits to 8, 4, 2, 1 code to make it a single error correction code. In order to correct a single error we should know where the error occurred in the composite code including the parity bits. With four information bits we need at least 3 parity bits so that the parity bits can be used to find out the error position in the 7-bit code. The code is constructed as follows:

The individual bits in the 7-bit code are numbered from 1 to 7. Bit positions 1, 2 and 4 are used as parity check bits and bits 3, 5, 6, 7 as information bits. The bit at position 1 is set so that it satisfies an even parity for bits 1, 3, 5, 7. Bit 2 is set to satisfy an even parity on bits 2, 3, 6, 7. Bit 4 is set so that it satisfies an even parity on bits 4, 5, 6, 7.

When a code is received the following procedure is used to detect and correct the error.

(i) Check even parity on positions 1, 3, 5, 7. If it passes then $C_1 = 0$, else $C_1 = 1$.

(ii) Check even parity on positions 2, 3, 6, 7. If it passes then $C_2 = 0$, else $C_2 = 1$

(iii) Check even parity on positions 4, 5, 6, 7. If it passes then $C_4 = 0$, else $C_4 = 1$.

(iv) The decimal equivalent of $C_4C_2C_1$ gives the position of the incorrect bit and that bit is corrected. If $C_4C_2C_1 = 0$ then there is no error in the code.

Example:

We suppose that the following code is received.

0110110 ←—— Received Code

1234567 ←—— Bit positions

Even parity in positions 1, 3, 5, 7 passes. Thus $C_1 = 0$.

Even parity in positions 2, 3, 6, 7 fails. Thus $C_2 = 1$.

Even parity in positions 4, 5, 6, 7 passes. Thus $C_4 = 0$.

The position of the error is thus $C_4C_2C_1 = 010 = 2$.

The correct code is thus: 0 0 1 0 1 1 0.

4.6.8 Alphanumeric codes:

All digital computers require the handling of data that consist not only of numbers, but also of letters and special characters. These are normally the 26 English letters of the alphabet, the 10 decimal digits and several special characters such +, —, X, /, $, etc. In order to code these with binary numbers one needs a string of binary digits. In recent machines the number of characters has increased. Besides capital letters of the alphabet, the lower case letters are also used and several mathematical symbols such as $>, \geq, \leq, \leq, =, \neq$, etc., have been introduced. The need to represent more than 64 characters gave rise to 7 or 8 bits to code a character. With / bits we can code 128 characters which are quite adequate. In order to ensure uniformity in coding characters, two standards have been involved. These are the ASCII (American Standard Code for Information Interchange) which is a 7-bit code, and EBCDIC (Extended BCD Interchange Code) which is 8-bits code.

The ASCII code consists of seven bits for all practical. It is used to code two types of information. One type is the printable characters such as letters, numeric and special characters. The other set is known as Control Characters which represent coded information to control the operation of digital computers and are not printed. A parity bit may be added to the 7-bit ASCII character code to yield an eight-bit code. A group of eight bits is known as a byte. A byte would be sufficient to represent a character.

5. Unit Summary

In this module, number system, addition and subtraction of binary numbers, the concept of Boolean algebra, its basic theorems, canonical Form of Boolean algebra, algorithm and flow-Chart, computer fundamental, different types of 1/O Units, data Representation, binary coded decimal numbers, self-complementing codes, cyclic codes, error-detecting codes, error-correcting codes, hamming codes for error correction and alphanumeric codes are discussed.

6. Self Assessment Questions

1. Add and multiply the following binary numbers without converting to decimal: (i) 101101 and 100101, (ii) 1101 and 1001
2. Convert the following binary numbers to decimal: 10.10001, 101110.0101.
3. Convert the following decimal numbers to binary: 12.0624, 673.23, and 1998.
4. Subtract the following binary numbers: 1101-1010, 101101-10001.
5. Find the complement of the following Boolean function

   $AB' + C'D'$
6. Obtain the truth table of the function:

   $F = xy + xy' + y'z$
7. What is the difference between canonical form and standard form?

7. Suggested further Readings

1. M. Moris Mano, Digital Logic and Computer Design, Prentice-Hall of India, 2003.
2. V. Rajaraman and T. Radhakrishnan, An Introduction to Digital Computer Design, Prentice-Hall of India, 1987.

VIDYASAGAR UNIVERSITY

DIRECTORATE OF DISTANCE EDUCATION
MIDNAPORE - 721 102

M.Sc. in Applied Mathematics with Oceanology and Computer Programming

Part-I
Group-C Paper-III
Module No. - 34

INTRODUCTION TO COMPUTING-II

CONTENTS

Module 34: Introducing to Computing

1.0 Introduction

Processors of computers perform arithmetic operations only on binary numbers. We should thus know how to do binary arithmetic in order to understand the working of processors. We use the Boolean algebra to design simple logic circuits for logical and arithmetic operations performed by processors.

2.0 Objectives

In this module, the main objectives are to study the followings

- Arithmetic Operations, floating point representation of numbers, etc.
- Simplifying Boolean expressions by Veitch-Karnaugh Map Method.
- Combinatorial Circuit Design Procedure

3.0 Key Words and Study guides

One's complement, two's complement, Binary addition; Binary Subtraction, Binary multiplication, Binary division, Floating point representation, Veitch-Karnaugh Map, Combinatorial circuit.

4.1 Arithmetic Operation:

4.1.1 Complements Representation of Numbers:

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulations. There are two types of complements for each base-r system: (1) the r's complement and (2) the (r-1)'s complement for binary numbes, or 10's and 9's complement for decimal numbers.

The r's complement:

Given a positive number N in base r with an integer part of n digit, the r's complement of N is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$.

Example:

The 10's complement of $(52520)_{10}$ is $10^5 - 52520 = 47480$.

The number of digits in the number is $n = 5$.

The 10's complement of $(0.3267)_{10}$ is $1 - 0.3267 = 0.6733$.

No integer part, so $10^n = 10^0 = 1$

The 10's complement of $(25.639)_{10} = 10^2 - 25.639 = 74.361$.

The 2's complement of $(101100)_2$ is $(2^6)_{10} - (101100)_2$

$= (1000000 - 101100)_2 = 010100$.

The 2's complement of $(0.0110)_2$ is $(1 - 0.0110)_2 = 0.1010$.

From the definition and the examples, it is clear that the **10's complement** of a decimal number can be formed by leaving all least significant zeros unchanged, subtracting the first non-zero least significant digit from 10, and then subtracting all other higher significant digits from 0. **The 2's complement** can be formed by leaving all least significant zeros and the first non-zero digit unchanged, and when replacing 1's by 0'2 and 0's by 1's in all other higher significant digits.

The (r–1) complement:

Given a positive number N in the base r with an integer part of n digits and a fraction part of a m digits, the (r-1)'s complement of N is defined as $r^n - r^{-m} = N$.

Example:

The 9's complement of $(52520)_{10}$ is $(10^5 - 1 - 52520) = 99999 - 52520 = 47479$.

No fraction part, so $10^{-m} = 10^0 = 1$.

The 9's complement of $(0.3267)_{10}$ is $(1 - 10^{-4} - 0.3267) = 0.9999 - 0.3267 = 0.6732$.

No integer part, so $10^n = 10^0 = 1$.

The 9's complement of $(25.639)_{10}$ is $(10^2 - 10^{-3} - 25.639) = 99.999 - 25.639 = 74.360$.

The 1's complement of $(101100)_2$ is $(2^6 - 1) - (101100) = (111111 - 101100)_2 = 010011$.

The 1's complement of $(0.0110)_2$ is $(1 - 2^{-2})_{10} - (0.0110)_2$

$= (0.1111 - 0.0110)_2 = 0.1001$.

From these examples, we say that the 9's complement of a decimal number is formed simply by subtracting every digit from 9. The 1's complement of a binary number is even simpler to form: the 1's are changed to 0's and the 0's to 1's. Since the (r–1)'s complement is very easily obtained, it is sometimes convenient to use it when the r's complement is desired. It follows that the r's complement can be obtained from the (r–1)'s complement after the addition of r^{-m} to the least significant digit. For example, the 2's complement of 10110100 is obtained from the 1's complement 01001011 by adding 1 to give 01001100.

4.1.2 Addition/Subtraction in One's and two's Complement:

Subtraction with *r*'s complement:

The subtraction of two positive numbers M and N, $(M-N)$, both of base r is done as follows:

(i) Add the minuend M to the r's complement of the subtrahend N.

(ii) Inspect the result obtained in step (i) for an end carry:

(a) If an end carry occurs, discard it.

(b) If an end carry does not occur, take the r's complement of the number obtained in step (i) and place a negative sign in front.

Example: Using 10's complement, subtract 72532-3250.

$M = 72532$ 72532

$N = 03250$ +

10's complement of $N = 96750$

end carry → 1 69282

$\therefore M - N = 69282$

Example: Subtract $(3250 - 72532)_{10}$.

$M = 03250$ 03250

$N = 72532$ +

10's complement of $N = 27468$

no carry 30718

$\therefore M - N = -69282 = -$(10's complement of 30718).

Example: Use 2's complement to perform $(1010100 - 1000100)_2$.

$M = 1010100$ 1010100

$N = 1000100$ +

2's complement of $N = 0111100$

end carry → 1 0010000

$\therefore M - N = 10000$

Example: Use 2's complement to perform $(1000100 - 1010100)_2$.

$M = 1000100$	1000100
$N = 1010100$	+
2's complement of N =	0101100
no carry	1110000

$\therefore M - N = -10000 = -$ (2's complement of 1110000).

Subtraction with (*r*-1)'s complement:

The subtraction of $M - N$, both positive numbers in base r is calculated in the following:

(i) Add the minuend M to the (r-1)'s complement of the subtrahend N.

(ii) Inspect the result obtained in step(i) for an end carry.

(a) If an end carry occurs, add 1 to the least significant digit (end-around carry).

(b) If an end carry does not occur, take the (r-1)'s complement of the number obtained in step-(i)and place a negative sign in front.

Example: Using 9's complement, subtract 72532 - 3250.

$M = 72532$	72532
$N = 03250$	+
9's complement of N =	96749
end carry → 1	69281
	+ 1
	59282

$\therefore M - N = 69282$.

Example: Subtract $(3250 - 72532)_{10}$.

$M = 03250$ 03250

$N = 72532$ +

9's complement of N = 27467

no carry 30717

$\therefore M - N = -69282 = -$(9's complement of 30717).

Example: Use 1's complement to perform $(1010100 - 1000100)_2$

$M = 1010100$ 1010100

$N = 1000100$ +

1's complement of N = 0111011

0010000

end carry → 1 0001111

+ 1

$\therefore M - N = 10000$.

Example: Use 1's complement to perform $(1000100 - 1010100)_2$.

$M = 1000100$ 1000100

$N = 1010100$ +

1's complement of N = 0101011

no carry 1101111

$\therefore M - N = -10000 = -$(1's complement of 1101111).

Comparison between 1's and 2's complements:

(i) The 1's complement has the advantage of being easier to implement by digital components since the only thing that must be done is to change 0's into 1's and 1's into 0's.

The implementation of the 2's complement may be obtained in two ways: (a) by adding 1 to the least significant digit of the 1's complement and (b) by leaving all leading 0's in the least significant positions and the first 1 unchanged, and only then changing all 1's into 0's and all 0's into 1's.

(ii) The 1's complement requires two arithmetic additions when an end-around carry occurs. The 1's complement has the additional disadvantages of possessing two arithmetic zones: one with all 0's and one with all 1's.

During subtraction of two numbers by complements, the 2's complement is advantageous in that only one arithmetic addition operation is required.

4.1.3 Binary Multiplication and Multiplication of signed numbers:

Binary Multiplication: It is nothing but successive addition. The method used to multiply two-signed numbers depends on the method used to represent negative numbers. If negative numbers are represented in the one's or two's complement form then multiplication becomes complicated. The following is the method for multiplication of two numbers in sign magnitude notation:

Step-1: Examine the least significant bit of the multiplier. If it is a 1, copy the multiplicand and call it the first partial product. If the least significant bit is a zero, then enter zero as the first partial product, store the partial product.

Step-1: Examine the bit left of the bit examined last. If it is 1, do Step-3, otherwise do Step-4.

Step-3: Add the multiplicant to the previously stored partial product after shifting the partial product one bit to the right. This sum becomes the new partial product. Go to Step 5.

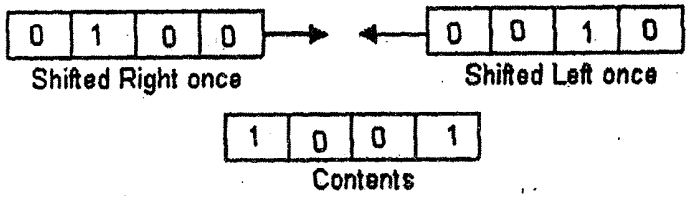
Step-4: Get new partial product by shifting the previous partial product one bit to the right.

Step-5: Repeat Steps 2 to 4 till all bits in the multiplier have been considered. The final value obtained for the partial product is the product of the multiplicant and multiplier.

Example: Multiply 11012 *and* 1011.

Multiplicand	1101
Multiplier	1011
Partial product	1101
	1101
	0000
	1101
Product	10001111

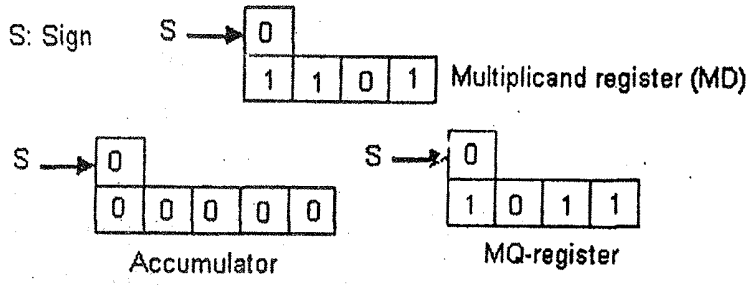
To implement this method, we introduce a device called a register. A register can store or preserve a string of binary digits. The length of the register equals the number of bits it can store. The contents of a register can be shifted left or right. If the content of a register is shifted right by one bit, then the least significant bit is lost. The position occupied by the most significant bit becomes empty and is filled with 0. Similarly if the contents are shifted left, the most significant bit is lost and the least significant bit filled by a 0. This is illustrated as below.



In order to multiply we need three registers. Assuming an 'n bit' multiplier and an 'n bit' multiplicand, we need an 'n bit' register to store the multiplier and a '2n bit' register to store the final product and the intermediate partial products. We may reduce the length of the product register by remembering that, after each bit of the multiplier is used to develop a partial product, it may be discarded and that the length of partial product grows from n to 2n in increments of one bit.

Therefore we implement multiplication using three registers: a multiplicand register which can store n bits, a multiplier-quotient register of n bits to store the multiplier (or the quotient during division), and an accumulator in which partial products are added and stored. This register needs (n+1) bits as the intermediate sum of two n bit numbers could be (n+1) bits long. The accumulator and the MQ register are physically joined so that their contents can be shifted

together either left or right. For example, for n=4 the configuration of the registers and their initial contents for the above example is given below:



Multiplication of Signed Numbers: There are two methods for multiplication of signed numbers which are as follows.

Method 1: For method 1 there are two cases.

Case 1: If one of the numbers, say the multiplicand is negative and ie represented in the 2's complement form then multiplication can be carried out by the same procedure mentioned in above. We illustrate it taking following example:

```
    10011   (-13) Multiplicand
    00 110  (+6) Multiplier 0000000000
111110011
 11110011
  0000000
   000000
1110110010  (-78)
```

We observe that the leading bits of the negative partial products are made 1. This is known as sign bit extension. This is necessary if we remember that the leading zeros of appositive number would become ones when their two's complement is taken. For example, +5=00000101 and -5=11111011.

Case 2: If the multiplier is negative and the multiplicand is positive we can interchange them and carry out the same procedure or we can complement both of them and carry out the procedure. If both are negative then both can be complemented and we can use the procedure used for positive operands.

Method 2: The method of dealing with operands in two's complement form was proposed by Booth. This method has two advantages which are as follows:

- Firstly it deals with both positive and negative multipliers uniformly.
- Secondly it reduces the number of addition operations in the multiplication in cases where the multiplier has along sequences of ones.

We illustrate it using following two examples.

EX—1:

000110 (+6) Multiplicand

001110 (+14) Multiplier

000000000000

00000000110

0000000110

000000110

00000000

0000000

000001010100 (+84) Product

EX-2

010000 (16)

– 000010 (2)

001110 (14)

00000 110 (+6) Multiplicand

0 + 100 –10 (+ 14) Multiplier

0000000 000 00

1111111 101 0

0000000 000

0000000 00

0000011 0

0000000

00000 10 101 00 (+84) Product

We observe that 14 can be expressed as (16-2).

4.1.4 Binary Division:

We discuss a method for dividing integers represented using the sign magnitude notation. The method is called the **restoring method** for division. Another method called the non-restoring method is also popular. The restoring method is given as follows:

Step-1: Let y be the most significant bit of the dividend.

Step-2: Subtract the divisor from y.

Step-3: If a borrow is generated in subtraction then add divisor to the remainder. Quotient bit is 0. Else quotient bit is 1.

Step-4: Append the next significant bit of the dividend to the remainder.

From this procedure, we see that

a) The divisor is to be preserved as it is to be successively subtracted from the dividend.

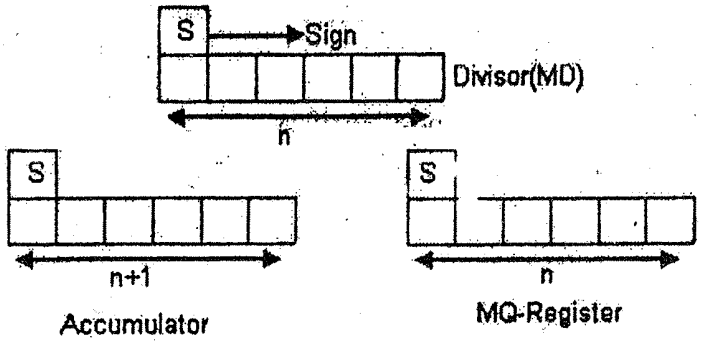
b) The dividend bits are used starting from the most significant bit. Once a bit is used to develop a quotient it is not needed again. The bit to its right is appended to the remainder for developing the next quotient bit.

c) As each quotient bit is developed the corresponding most significant bit of the dividend may be discarded.

Ex. Divide 1011 by 11

Quotient bit	*Divisor*	*Dividend*	*Quotient*
	11	) 1011 (	0011
		11	
0	Borrow →	110	
	Restore	11	
		010	
		11	
0	Borrow	111	
	Restore	11	
		101	
		11	
1	No Borrow	101	
		11	
1	No Borrow	10	← Remainder

Therefore, here we may use three registers. The three registers used are an n bit register to store the divisor, an (n+1) bit accumulator in which the dividend is originally stored and from which the divisor is subtracted in each step and an n bit quotient register as in the following figure. The accumulator and the MQ register may be again physically joined so that their contents can be shifted together.



4.1.5 Arithmetic with BCD numbers:

In this section we discuss how binary coded decimal numbers can be added and subtracted using adders constructed to add binary numbers. This is illustrated using 8, 4, 2, 1 and excess-3 codes.

Arithmetic in 8, 4, 2, 1 code: In this coding scheme we use the first 10 natural sequences of 4 bit groups, from 0000 to 1001. The remaining six 4 bit groups, namely, 1010 to 1111 are illegal in this code. This code is represented on a circle.

We assume that we want to add 6 to 8. If we add the codes for 6 and 8 using binary addition, we obtain

6+8=0110+1000=1110

for the sum. The 4 bit group 1110 is, however, illegal in 8,4,2,1 code. The correct answer, namely, 14, should be represented as 00010100 in this code. We must thus find means of correcting an illegal code obtained on addition. The figure suggests a method itself. Remember that addition is just counting starting with the augends as the "origin", counting succeeding symbols, and terminating when the addend equals the count. After counting, if we find the result within 9 then 'the result needs no correction. If the result is between 10 and 15 we will be in an illegal group. To get the right code we should skip six code groups. This may be achieved by adding a six to the result.

Implementing this scheme with electronic circuitry requires a simple means of detecting legal codes and correcting the illegal codes. Observe that if the answer is less than or equal to 9 and a 6 is added, the binary equivalent of this sum will not exceed four bits in length (as it will be 1111). If the answer is 10 or higher, adding a 6 to the answer will result in a sum of 16 or more and the binary equivalent of this number will be 5 bits long giving an overflow bit. Thus the absence of an overflow when a 6 is added indicates that the result is a legal code and the presence of an

overflow indicates that the answer lies in the illegal code group. When an overflow results, it also indicates that the sum leads to a carry. The least significant 4 bits of the sum resulting after adding a six pushes the answer into the right code group. This is illustrated in the following examples.

Example-1:		Example-2:		Or 6	0110
6	0110	6	0110	2	0010
8	1000	2	0010	Answer	1000
	1110		1000	Add 6	0110
Add 6	0110	Add 6	0110		1110
Overflow ←	1 0100		1110	Add 10	1010
	Right Code		No Overflow	Ignore overflow	11000
Answer 0001	0100			(Equivalent to subtracting 16)	
1	4	1			

We observe that in Example-2 when no overflow is detected on adding six to the sum of the addend and the augend it is not necessary to subtract six from the final result to get back the correct sum. Instead one may add 10 to the final result and ignore the resulting overflow because $-6 = (10 - 16)$.

Negative decimal numbers may be represented using the same principles which were used to represent negative binary numbers. The three methods of representing negative numbers, namely, sign magnitude form, one's complement form and the two's complement form may be extended to BCD numbers. The one's complement is replaced by nine's complement and the two's complement by the 10's complement for decimal numbers. The following example illustrates the three representations:

$-6 = 1,0110$ Sign, magnitude form

$-6 = 1,0011$ 9's complement form $(9 - 6) = 3 = 0011$

$6 = 1,0100$ 10's complement form $(10 - 6) = 4 = 0100$

Subtraction of decimal numbers may be achieved by adding their nine's complements in a way identical to that used in binary subtraction.

Arithmetic in excess-3 code: In the excess-3 code (Xs-3 code), the code for a decimal digit is the binary equivalent of the digit plus 3 (3 = 0011). This code is illustrated on a circle in the above figure. When we add two digits expressed in this code the answer would have an excess of 6 over its binary equivalent. If an overflow is obtained during addition it indicates that the answer is ≥ 16. This implies that the sum of the digits expressed in $Xs-3$ is ≥ 10, i.e., there is carry to the next digit. Notice that the analog with the addition in the 8, 4, 2, 1 code. In that case we added six to the sum of the digits to detect an overflow, the occurrence of which indicated a carry to the next digit. In the present case the addition of six is built-in in the $Xs-3$ coding scheme. An overflow indicates a carry to the next digit. The last 4 bits of the sum would be in 8, 4, 2, 1 code and not in the $Xs-3$ code. Therefore, we have to add (0011) to the sum to restore it to the $Xs-3$ code.

If no overflow is detected on addition of two numbers in $Xs-3$ code then it implies that there is no carry to the next digit. Further there will be an excess of six in the sum and we must subtract 3 from the result to restore it to the $Xs-3$ code. Remembering that $-3 = 13-16$, instead of subtracting 3 from the sum we may add 13 and ignore the overflow. This will ensure that the result is in $Xs-3$ code.

4.1.6 Floating Point Representation of Numbers:

A number with a functional part may be stored and represented in several ways in a digital system. For example, we consider a real number 172.354. It may be written as follows:

(i) 172.354

(ii) $.172354 \times 10^3$

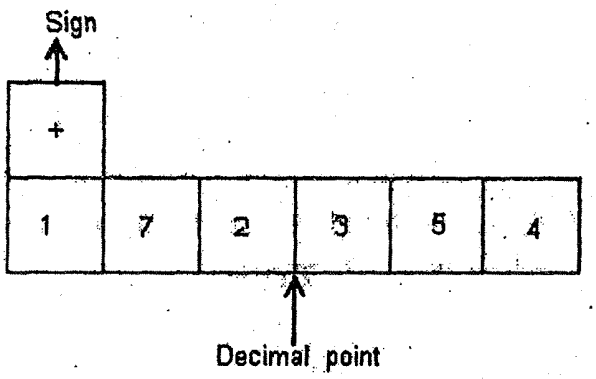
Now we assume that a register is available which can store 6 digits and a sign bit. **One way of storing** the above number in the register is to imagine that the register is split into two parts:

(i) One part containing the integer portion of the number.

(ii) the other part containing the fractional portion of the number.

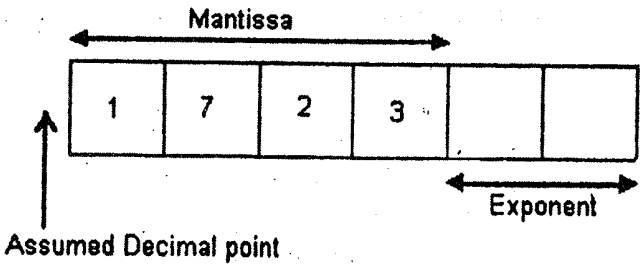
The decimal point is assumed to exist between the two parts of the register. It does not exist physically as a stored bit; it only resides in the mind of the user and should be remembered by him while manipulating the number in the register. The following figure illustrates this representation. Hardware implementation of arithmetic operations is simple if this representation is used. The two parts of the number may be treated independently. After the operation is performed one may transfer any carry or borrow generated in the fractional part to the integer part. The practical

difficulty in using this scheme is the need for the user to keep track of the decimal point location and significant digits, particularly in multiplication and division operations. Further, the range of numbers that could be represented using this notation is limited. The range of the following register configuration is ±999.999.



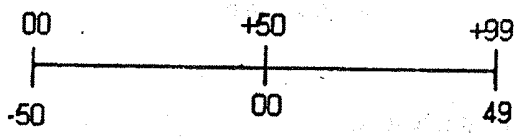
Now we consider another method of storing real numbers using the register in the above figure. This is known as the floating point representation. We consider the second form of 172.354. In this form the number is written as a fraction multiplied by a power of 10. The fractional part is known as the mantissa and the power of 10 multiplying the fraction is known as the exponent. If the other number systems are used, such as, octal, a number would be represented by a fraction in that base multiplied by a power of 8

(e.g., 127 x 84). If a number in this form is to be stored in a register with a capacity of 6 digits and a sign, then we should divide again into two parts; one part to hold the mantissa and one part to hold the exponent. If we arbitrarily allot 2 digits for the exponent and 4 digits for the mantissa we may store the number in the register as in the following figure.

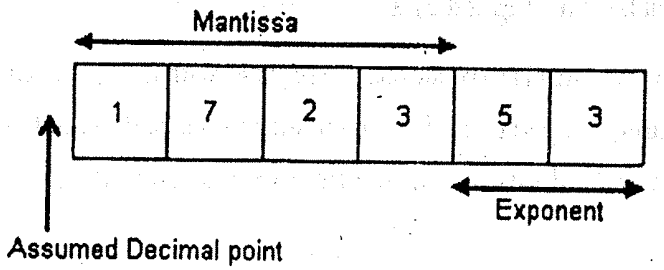


Two problems arise in doing this. Firstly, all the six digits in the number can not be stored in the register as two digits in the register have been taken out to store the-exponent. Secondly, as only one sign bit is available it can be used with mantissa only. We have to devise some other way to indicate the sign of the exponent. As in the above example, the most significant 4 digits of the number are stored in the mantissa part. The last two digits are truncated and are thus lost. **This is the penalty for having a separate exponent.**

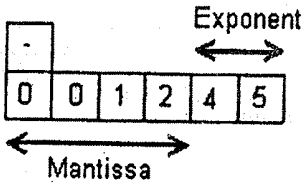
In order to be able to store both positive and negative exponents it would be necessary to split the range of exponent, namely 00 to 99, into two parts. If we shift the origin to 50, we may interpret all exponents > 50 as positive and all exponents < 50 as negative. Therefore, the range of the exponent will be – 50 to + 49 . Exponents expressed in this notation are said to be in the excess 50 form. This is illustrated below.



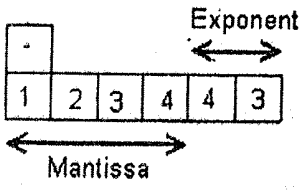
Example: Using this notation .1723 x 10^3 may be stored in the register as shown in the following figure:



Example: Assume that we want to store the number—.001234 x 10^{-5}. If we blindly follow the procedure mentioned above then we may store it as below.



Normalization: A little thought shows that the information conveyed by the leading zeros in the mantissa may be included in the exponent. The number may be written as —.1234 x 10^{-7} and stored as below, thereby preserving all the significant digits in the mantissa. This technique is called normalization. In the normalized floating point representation, the most significant digit of the mantissa is non-zero. **Normalization is universally used in digital systems.**



With the normalized floating point representation of real numbers the range of numbers we can represent in a 6 digit register is

maximum magnitude .9999 X 10^{+49}

minimum magnitude A000 x 10^{-50}

This should be compared with the range representable with a fixed (assumed) decimal point which is +999.999. The smallest number which could be stored with this technique is 000.001. Therefore using a given six digit register with one sign bit, the normalized floating point representation is able to store a much wider range of numbers. Hence, the loss of two significant digits in order to do this is well worth it. Besides the loss of two significant digits, the use of normalized floating point numbers requires that some specific rules be followed when arithmetic operations are performed with such numbers.

How to represent a zero in the normalized floating point representation? If all the digits in the mantissa are zero then one may conclude that the number is zero. In actual computation, however, due to rounding of numbers which arises because of finite number of digits in the mantissa it would be preferable to call a very small number not exactly zero as zero. This suggests that zero may be represented by all zeros for the mantissa and the largest and the largest negative number as exponent. In the excess 50 representation of exponent the largest negative exponent is represented by 0. Thus a zero will have both mantissa and exponent equal to zero.

Binary floating point numbers: Binary floating point numbers may also be represented in similar manner as

mantissa x $2^{exponent}$

where the mantissa would be a binary fraction with a non-zero leading bit. To illustrate it we suppose that a 32 bit register is available. We will now examine the methods which we may use to store real numbers in it. We will reject a fixed point representation, as the range of real numbers representable using this method is limited. With a floating point representation We have to decide the following:

- Number of bits used to represent the mantissa
- Number of bits used to represent the exponent
- Whether to use an excess representation for the exponent
- Whether to use a base other than 2 for the exponent

The number of bits to be used for the mantissa is determined by the number of significant digits required in computation. From the experience in numerical computation it is found necessary to have at least 7 significant digits in most practical problems. The number of bits required to represent 7 significant decimal digits is approximately 23 (1 decimal digit = $\log_2 10$ bits). If one bit is allocated for the sign of the mantissa then 8 bits are left for the exponent. If we decide to allocate one bit of the exponent for its sign then 7 bits are left for the magnitude of the exponent. In the normalized floating point mode the **largest magnitude number** which may be stored is

$0.111.....0\times 2^{1111111} = \left(1-2^{-24}\right)\times 2^{\left(2^7-1\right)}$

23 bits $= \left(1-2^{-24}\right)\times 2^{(127)}$

$\cong 10^{38}$

The minimum magnitude number is

0.1000 $0x2^{-1111111} = 0.5 \times 10^{-38}$

23 bits

If we use excess representation for the exponent the range of the binary exponent would be 0 to 256. The minimum exponent would be – 128 and the maximum 127. We thus do not gain exponent range. The main gain in using this representation would be the representation of zero. A zero will be represented by all 0 bits for mantissa as well as the exponent.

4.1.7 Floating point addition and subtraction:

If two numbers represented in floating point notation are to be added or subtracted the exponents of the two operands must be made equal. In doing this an operand would need shifting. To illustrate this we consider the following examples.

Example:

	Number	Register form
Operand 1 :	$.1234 \times 10^{-3}$	123447
Operand 2 :	$.4568 \times 10^{-3}$	456847
Addition :	$.5802 \times 10^{-3}$	580247

Example:

	Number	Register form
Operand 1 :	$.1234 \times 10^{-3}$	123453
Operand 2 :	$.4568 \times 10^{-2}$	456852
Addition :	Shift operand 2 right by one place and add	
		123453
		045653
$.1690 \times 10^{3}$		169053

Example:

	Number	Register form
Operand 1 :	$.4568 \times 10^{49}$	456899
Operand 2 :	$.8268 \times 10^{49}$	826899
Addition :		
	1.2836×10^{49}	1238(100)
	$.1283 \times 10^{50}$	x

In this example the sum exceeds $.9999 \times 10^{49}$ and is called an overflow condition.

Example:

	Number	Register form
Operand 1:	+.4568 x 10^{-50}	+ 456800
Operand 2:	—.4500 x 10^{-50}	— 450000
Addition :		
	.0068 x 10^{-50}	— 6800(-52)
	.6800 x 10^{-52}	x

In this example the answer is less than .1 X 10-50 and is called an under condition

4.1.8 Floating point multiplication:

Floating point multiplication is fairly simple. In this case the exponents are added and the mantissa multiplied. In the added exponents there would be an excess of 50. This is subtracted. If the exponent exceeds 99, overflow is declared. As both mantissas are normalized to less than 1 the product mantissa can not exceed 1. The mantissa, however, can have leading zeros. Thus normalization after the product is obtained with adjustment of exponent would be necessary.

4.1.9 Floating point division:

In this case the divisor mantissa should be larger than the dividend mantissa. If this is not true a divide stop will occur. To ensure that division does not stop we may shift the dividend right by one digit and add 1 to the exponent before the beginning of division.

The dividend is placed in the accumulator and the divisor in the MR register. Division of the mantissa parts precede as usual. The fractional part of the answer is stored in MQ at the end of the division. The quotient is shifted left till the most significant digit is non-zero. The exponent part of the answer is equal to

(A CC exponent — MR exponent — Number of left shifts + 50)

Fifty is added as the excess 50 in the accumulator and MR exponents would be cancelled in the subtraction operation.

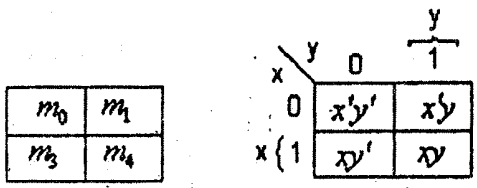
Note: Same procedures can be followed when both exponent and mantissa are represented in binary form.

4.2 Simplifying Boolean expressions by Veitch- Karnaugh Map Method:

The Map Method: The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented. Although the truth table representation of a function is unique, expressed algebraically, it can appear in many different forms. Boolean functions may be simplified by algebraic means. But this process for simplification is very troublesome. The other method for simplifying a Boolean function is Map method which provides a simple straightforward procedure. This method, first proposed by Veitch and then slightly modified by Karnaugh, is also known as the Karnaugh map or Veitch diagram.

The map is a diagram made up of squares. Each square represents one minterm. Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function. In fact, the map presents a visual diagram of all possible ways a function may be expressed in a standard form. We shall assume that the simplest algebraic expression is any one in a sum of products or product of sums that has a minimum number of literals.

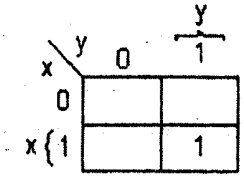
Two-variable map: There are four minterms for two variables. A two-variable map which consists of four squares, one for each minterm, is shown below.



The O's and 1's marked for each row and each column designates the values of variables x and y, respectively. x appears primed in row 0 and unprimed in row 1. Similarly, y appears primed in column 0 and unprimed in column 1. If we mark the squares whose minterms belong to a given function, the two-variable map becomes another useful way to represent any one of the 16 Boolean functions of two variables.

Example: f=xy

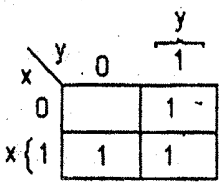
Since xy is equal to m_3, a 1 is placed inside the square that belongs to m_3.



Example: f=x+y

The function x+ is represented in the map by three squares marked with 1's. These squares are found from the minterms of the function:

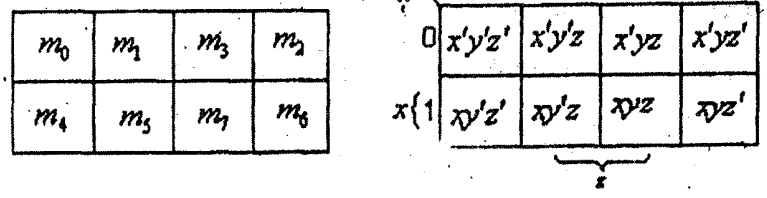
$$f = x + y = x'y + xy' + xy = m_1 + m_2 + m_3$$



The three squares could have also been determined from the intersection of variable x in the second row and variable y in the second column, which encloses the area belonging to x or y.

Three-variable map: A three-variable map is given below. There are eight minters for three binary variables. Therefore, a map consists of eight squares. The minterms are arranged, not in binary sequence, but in a sequence similar to the reflected code. The characteristic of this sequence is that only one bit changes from 1 to 0 or from 0 to 1 in the listing sequence. The map drawn-in the following is marked with numbers in each row and each column to show the relationship between the squares and the three variables. For example, the square assigned m- corresponds to row 1 and column 01. When these two numbers are concatenated, they give the binary number 101, whose decimal equivalent is 5. Another

way of looking at square m_5 = xy'z is to consider it to be in the row marked x and the column belonging to y'z (column 01). We note that there are four squares where each variable is equal to 1 and four where each is equal to 0. The variable appear unprimed in those four squares where it is equal to 1 and primed in those squares where it is equal to 0.



Logic for simplifying Boolean functions: For simplifying Boolean functions, we must remember the basic property possessed by adjacent squares. Any two adjacent squares in the map differ by only one variable which is primed in one square and unprimed in the other. For example, m_5 and m_7 lie in two adjacent squares. Variable y is primed in m_5 and unprimed in m_7 while the other two variables are the same in both squares. From the postulates of Boolean algebra, it follows that the sum of two minterms in adjacent squares can be simplified to a single AND term consisting of only two literals. To clarify this, we consider the sum of two adjacent squares such as m_5 and m_7:

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

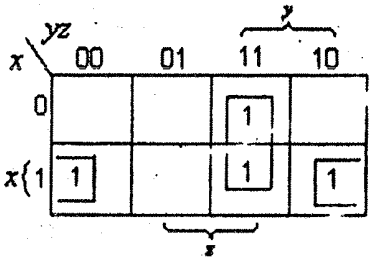
Here the two squares differ by the variable y, which can be removed when the sum of the two minterms is formed. Therefore, any two minterms in adjacent squares that are ORed together will cause a removal of the different variable.

Example-1: Simplify the Boolean function:

$$F(x, y, z) = x'yz + xy'z' + xyz + xyz'$$

There are four squares marked with 1's, one for each minterm of the function. Marking with 1's in each square can be accomplished in two ways: either by converting each minterm to a binary number and then marking a 1 in the corresponding square, or by obtaining the coincidence of the variables in each term. For example, the term $x'yz$ has the corresponding binary number 011 and represents minterm m_3 in square 011. The second way to recognize the square is by the coincidence of variables x', y, and z, which is found in the map by observing that x' belongs to the four squares in the first row, y belongs to the four squares in the two right columns, and z belongs to the four squares in the two middle columns. The area that belongs to all three literals is the single square in the first row and third column. Similarly the other squares belonging to the function F are marked by 1's in the map. Two adjacent

squares are combined in the third column to give a two-literal term yz. The remaining two squares with 1's are also adjacent by the new the new definition and are shown in the diagram enclosed by half rectangles. These two squares, when combined, give the two-literal term xz'. So the simplified function becomes $F = yz + xz'$



Example-2: Considering the following truth table, find the simplified Boolean expression.

Minterms	a	b	c	z
m_0	0	0	0	0
m_1	0	0	1	0
m_2	0	1	0	1
m_3	0	1	1	0
m_4	1	0	0	0
m_5	1	0	1	1
m_6	1	1	0	1
m_7	1	1	1	1

The mapping is straightforward. For each combination of a, b, ç with z=1, a 1 is entered in the Karnaugh map. Thus for abc = 010, 101, 110, and 111 we have I's in the map. The next step is to identify adjacent 1's in the map. Two adjacent I's may be combined to eliminate one literal. In this example, we have

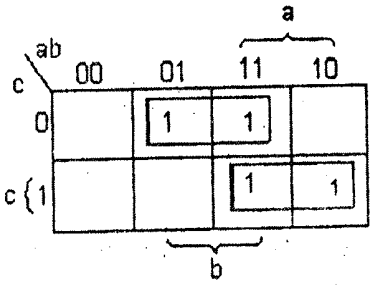
$$z = (a'bc' + abc') + (abc + ab'c)$$

$$= (a' + a)bc + ac(b + b')$$

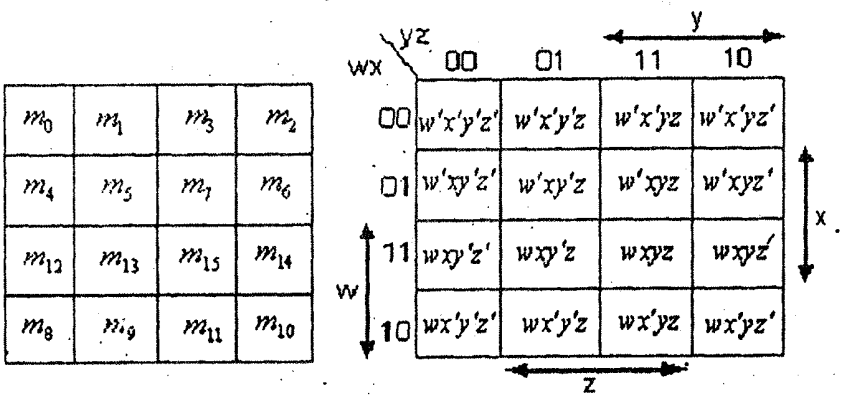
$$= bc' + ac$$

The main merit of the Karnaugh map is the fact that terms which are logically adjacent, that is, minterms which differ in only one variable (such as We, abc) are also physically adjacent in the map. Thus one can, by inspection, pick

terms to be combined to eliminate variables. This is true except for the first and last columns of the map. They are logically adjacent; the terms $a'b'c'$ and $ab'c'$ are adjacent; similarly $a'b'c'$ and $ab'c$ are adjacent. To ensure plysical adjacency one may imagine the map to be wrapped on a cylinder with the first and last columns next to one another.



Four-variable map: The map for Boolean functions of four binary variables is shown below. Here 16 minutes are listed and the squares are assigned to each. The rows and columns are numbered in a reflected-code sequence, with only one digit changing value between two adjacent rows or columns. The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number. For example, the numbers of the third row (11) and the second column (01), when concatenated, give the binary number (1101), the binary equivalent of decimal 13. Therefore, the square in the third row and second column represent minterm m_{13}.



Logic for simplifying Boolean functions: The map minimization of four-variable Boolean functions is similar to the method used to minimize three-variable functions. Adjacent squares are defined to be squares next to each other. In addition, the map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares. For example, m_0 and m_2 form adjacent squares, as do m_3 and m_{11}. The combination of adjacent squares that is useful during the simplification process is easily determined from the inspection of the four-variable map.

- One square represents one minterm, giving a term of four literals.

- Two adjacent squares represent a term of three literals.
- Four adjacent squares represent a term of two literals.
- Eight adjacent square represent a term of one literal.
- Sixteen adjacent squares reprsent the function equal to 1.

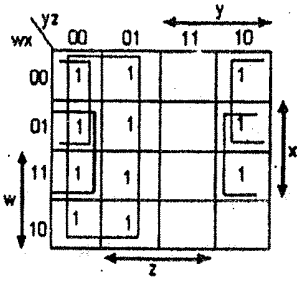
No other combination of squares can simplify the function.

Example - 1: Simplify the Boolean function:

$$F(w,x,y,z) = \sum(0,1,2,4,5,6,8,9,12,13,14)$$

Since the function has four variables, a four-variable map must be used. The minterms listed in the sum are marked by 1's in the map. Eight adjacent squares marked with 1's can be combined to form the one literal y'. The remaining three 1's on the right can not be combined together to give a simplified term. They must be combined as two or four adjacent squares. The larger the number of squares combined, the less the number of literals in the term. In this example, the top two 1's on the right are combined with the top 1's on the left to give the term $w'z'$. **Note that it is permissible to use the same square more than once.** We are now left with a square marked by 1 in the third row and fourth column (square 1110). Instead of taking this square alone (which will give a term of four literals), we combine it with squares already used to form an area of four adjacent squares. These squares comprise the two middle rows and the two end columns, giving the term xz'. The simplified function is $F = y' + w'z' + xz'$.

Example-2: Simplify the Boolean function:

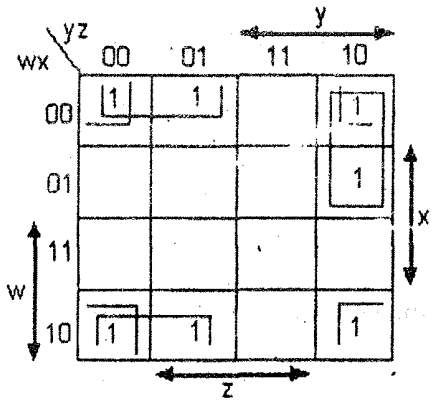


$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

The area in the map covered by this function consists of the squares marked with 1's in the following map. This function has four variables and, as expressed, consists of three terms, each with three literals, and one term of four literals. **Each term of three literals is represented in the map by two squares.** For example, $A'B'C'$ is

represented in squares 0000 and 0001. The function can be simplified in the map by taking the 1's in the four corners to give the term $B'D'$. This is possible because these four squares are adjacent when the map is drawn in a surface squares are adjacent when the map is drawn in a surface with top and bottom or left and right edges touching one another. The two left-hand 1's in the top row are combined with the two 1's in the bottom row to give the term $B'C'$. The remaining 1 may be combined in a two-square are to give the term $A'CD'$. The simplified function is

$$F = B'D' + B'C' + A'CD'$$



Don't Care Conditions: The 1's and 0's in the map signify the combination of variables that makes the function equal to 1 or 0 respectively. The combinations are usually obtained from a truth table that lists the conditions under which the function is a 1. The function is assumed equal to 0 under all other conditions. This assumption is not always true since there are applications where certain combinations of input variables never occur. For example, a four-bit decimal code has six combinations which are not used. Any digital circuit using this code operates under the assumption that these unused combinations will never occur as long as the system is working properly. As a result, we don't care what the function output is to be for these combinations of the variables because they are guaranteed never to occur. These don't-care conditions can be used on a map to provide further simplification of the function. A don't-care combination can not be marked with either a 1 or a 0 on the map. To distinguish the don't-care conditions from 1's and 0's, a X or ϕ will be used.

When choosing adjacent squares to simplify the function in the map, the X's may be assumed to be either 0 or 1, whichever gives the simplest expression. In addition, an X need not be used at all if it does not contribute to covering a larger area. In each case; the choice depends only on the simplification that can be achieved.

Example-1: Simplify the Boolean function: $F(w,x,y,z) = \sum(1,3,7,11,15)$ and the don't-care conditions: $d(w,x,y,z) = \sum(0,2,5)$

The minterms of F are the variable combinations that make the function equal to 1. The minterms of *d* are the don't-care combinations known never to occur. The minimization is shown as follows. The minterms of F are marked by 1's, those of are marked by X's, and remaining squares are filled with 0's.

In (a), the 1's and X's are combined in any convenient manner so as to enclose the maximum number of adjacent squares. It is not necessary to include all or any of e X's, but only those useful for simplifying a term. One combination that gives a minimum function encloses one X and leaves two out. This results in a simplified sum-of-products function: $F = w'z + yz$.

In (b), the O's are combined with any X's convenient to simplify the complement of the function. The best results are obtained if we enclose the two X's as shown. The complement function is simplified to $F' = z' + wy'$. Complementing again, we obtain a simplified product of sums function: $F = (w' + y)$.

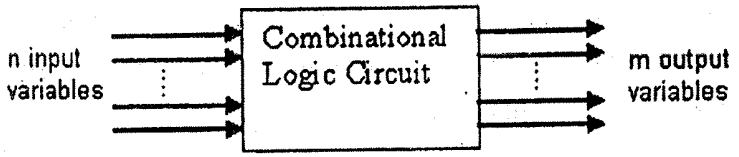
The two expressions obtained give two functions which can be shown to be algebraically equal. This is not always the case when don't-care conditions are involved.

4.3 Combinatorial Circuit Design Procedure and implementation by Binary operators and Logic Gates:

Logic circuits for digital systems may be combinational (also called combinatorial) or sequential. A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs. A combinational circuit performs a specific information-processing operation fully specified logically by a set of Boolean functions. Sequential circuits employ memory elements (binary cells) in addition to logic gates.

Combinational Circuit: It consists of input variables, logic gates, and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms binary information from the given input data to the required output data. Obviously, both input and output data are represented by binary

signals, i.e., they exist in two possible values, one representing logic-1 and the other logic-0. A block diagram of a combinational circuit is shown below.



Here, the n input binary variables come from an external source the m output variables go to an external destination. In many applications, the source and / or destination are storage registers located either in the vicinity of the combinational circuit or in a remote external device.

Design Procedure: The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram, or a set of Boolean functions from which the logic diagram can be easily obtained. The procedure involves the following steps:

(i) The problem is stated.

(ii) The number of available input variables and required output variables is determined.

(iii) The input and output variables are assigned letter symbols.

(iv) The truth table that defines the required relationships between inputs and outputs is derived.

(v) The simplified Boolean function for each output is obtained.

(vi) The logic diagram is drawn.

A truth table for a combinational circuit consists of input columns and output columns. The 1's and 0's in the input columns are obtained from the 2^n binary combinations available for n input variables. The output Boolean functions from the truth table are simplified by any available method, such as algebraic manipulation, the map method, or the tabulation procedure. Usually there will be a variety of simplified expressions from which to choose. A practical design method would have to consider such constraints as

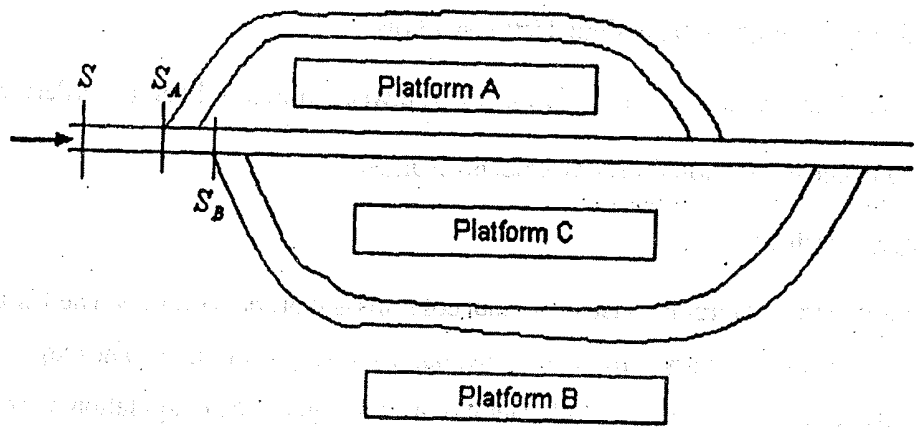
- minimum number of gates,
- minimum number of inputs to a gate,
- minimum propagation time of the signal through the circuit,

- minimum number of interconnections, and
- limitations of driving capabilities of each gate.

Since all these criteria can not be satisfied simultaneously, in most cases the simplification begins by satisfying an elementary objective, such as producing a simplified Boolean function in a standard form, and from that proceeds to meet any other performance criteria. Now we will illustrate the design procedure with a few examples.

Example-1:

Problem statement: A railway station with three platforms A, B and C is shown below in figure-1. A train coming to the station in the direction of the arrow is to be routed to platform A, B or C. Normally the train is to be routed to platform A if that platform is empty. Only if both A and B are occupied the train is to be routed to the platform C. Each platform has a switch which will be turned ON if a train is standing in that platform. A signal light is to be green if a train is allowed to enter the station otherwise it is red. A switching system is to be designed which will set points S_A, S_B to route trains to the appropriate platform and to control the signals.



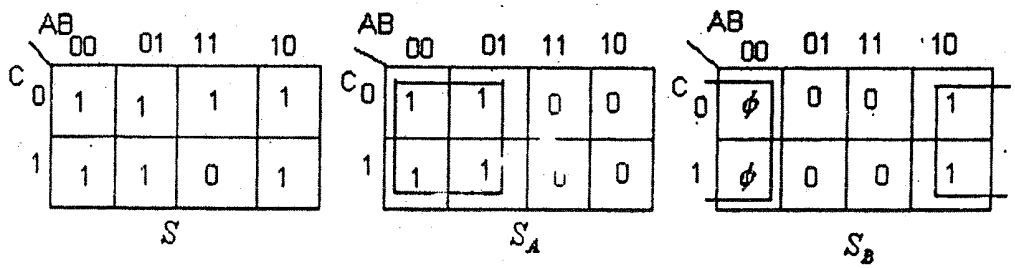
Truth Table for track switching					
A	B	C	S	S_A	S_B
0	0	0	1	1	ϕ
0	0	1	1	1	ϕ
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	0	0	0

Input and output variables: Analyzing this word statement, we see that the inputs to the switching system are the platform occupancy signals which we call A, B and C respectively. The outputs required are the signals to set the points and the green signal to let the train into the station. The switching system required is shown in figure-2.

In this example, all inputs and output variables are inherently Boolean in nature.

Now we do logical assignment to the variables. A '0' is assigned to variables A, B, C to indicate that platforms A, B, C respectively are empty and a '1' is used to indicate that the platform is occupied. **For the output variables** the assignment is a '1' if the particular signal is switched on (green signal) or the point is switched to allowed entry to the appropriate line. The truth table is constructed keeping in view the requirement that the trains are to be routed to platform A if the platform is vacant. Platform C is to be used only when both A and B are occupied.

From the truth table we obtain the Karnaugh maps as follows:



Using the normal reduction procedure, we obtain the following minimum expressions for the outputs:

$$S = \left(\overline{A} + \overline{B} + \overline{C}\right) = \overline{A.B.C}$$

$$S_A = \overline{A}$$

$$S_B = \overline{B}$$

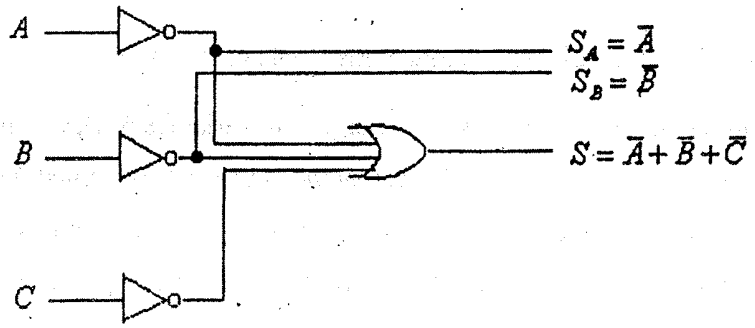
We may interpret these in words as:

(i) The signal for incoming train is turned green if platform A is not occupied or B is not occupied or C is not occupied. Equivalently, we may state that the signal is green if not all platforms A and B and C are occupied.

(ii) The point to platform A is set if platform A is not occupied.

(iii) The point to platform B is set if platform B is not occupied.

The corresponding switching circuit is given by the following



Example-2:

Problem Statement: A control unit is to be designed for a chemical process. Temperature and pressure are the two variables to be controlled. The control is exercised by switching on or off a heater and by opening or closing a valve. The control rules are given below:

(i) If the temperature and pressure are in the normal range, switch off heater and close the valve.

(ii) If the temperature is normal switch off the heater. Open the valve if the pressure is above normal and close it if it is below normal.

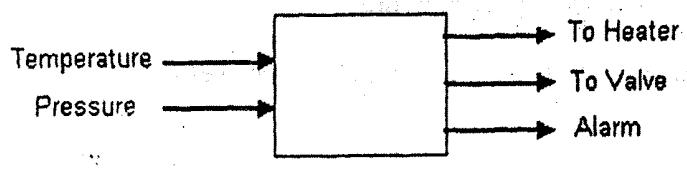
(iii) If the pressure is normal close the valve. Turn on the heater if the temperature is below normal and turn it off if the temperature is above normal.

(iv) If the pressure is above normal and the temperature is below normal open the valve, turn off the heater.

(v) If the temperature is above normal and the pressure is below normal, turn off the heater, close the valve.

(vi) If both temperature and pressure are above or below normal, ring an alarm and shut down the plant.

Given the above word statement we should first recognize the input and output variables. In this problem the input variables are temperature and pressure. The output variables are the signals to the heater, valve and alarm. The following figure shows the block diagram of the system.

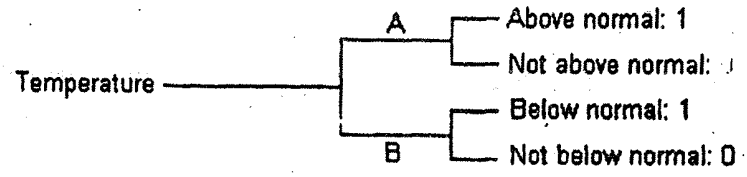


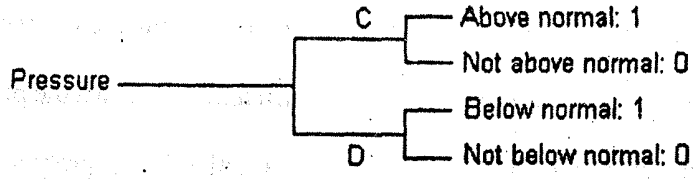
The next step is to assign truth values to the inputs and the outputs. The input conditions which determine controller action (namely the outputs) in this case are:

a) Temperature { above normal, normal, below normal }

b) Pressure { above normal
normal
below normal

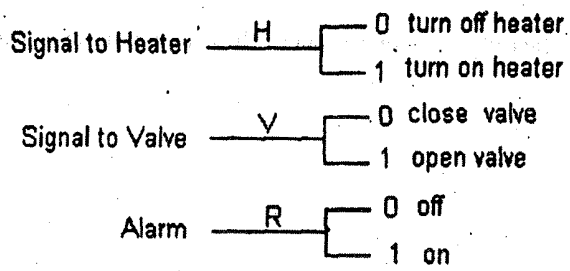
As there are three ranges for the input variables and we can represent only two valued variables in a truth table, we need to code each one of these input variables by two Boolean variables in appropriate coding as given below:





Observe that A = 0, B = 0 indicates temperature in the normal range, A=0, B=1 indicates temperature below normal; A = 1, B = 0 temperature above normal and A = 1, B 1 is an impossible combination. The same interpretation may be given to the variables C and D indicating the pressure.

The output variables may be coded in a straightforward manner as given below:



Using the above coding scheme, the word statement may be converted to the following truth table:

A	B	C	D	H	V	R
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	ϕ	ϕ	ϕ
0	1	0	0	1	0	0
0	1	0	1	ϕ	ϕ	1
0	1	1	0	0	1	0
0	1	1	1	ϕ	ϕ	ϕ
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	ϕ	ϕ	1
1	0	1	1	ϕ	ϕ	ϕ
1	1	0	0	ϕ	ϕ	ϕ
1	1	0	1	ϕ	ϕ	ϕ
1	1	0	1	ϕ	ϕ	ϕ
1	1	1	1	ϕ	ϕ	ϕ

Observe in the truth table that whenever AB = 11 or CD = 11 all the output variables have don't care values, that is, they may be 0 or 1. In developing this truth table, it has also been assumed that when the alarm conditions exist the plant . will be shut down and thus the heater and valve signals may be undefined, that is, they may be 0 or 1.

The Karnaugh map corresponding to the output variables H, V and R are shown below and using the don't care conditions; we obtain the following minimum expressions for the outputs:

$$H = B\bar{C}, V = C, R = AC + BD$$

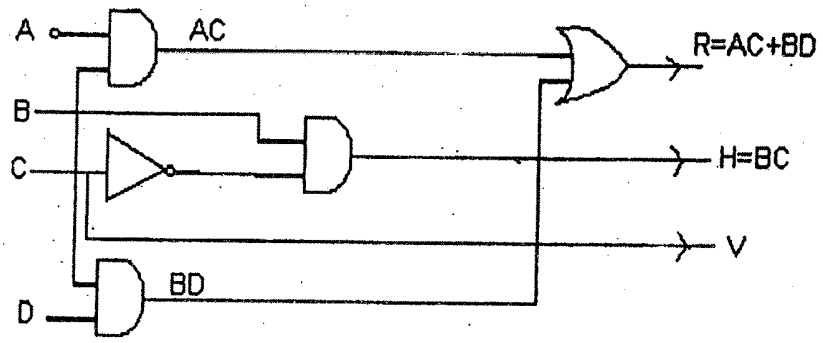
We may interpret them into the following word statements:

(i) The heater is turned on if the temperature is below normal and the pressure is not above normal.

(ii) The valve is opened if the pressure is above normal.

(iii) The alarm is turned on if the temperature is above normal and the pressure is above normal or if the temperature is below normal and the pressure is below normal.

After obtaining the minimized expressions, they may be realized using AND, OR, NOT gates as shown below:



5. Unit Summary

In this module, arithmetic Operations, complement representation of numbers, addition/ Subtraction in One's and two's complement, binary multiplication and multiplication of signed numbers, binary division, arithmetic with BCD numbers, floating point representation of numbers, floating-point addition and subtraction, floating-point multiplication, floating-point division, simplifying Boolean expressions by Veitch-Karnaugh Map Method, and combinatorial Circuit Design Procedure and implementation by binary operators and logic gates, are discussed.

6. Self Assessment Questions

1. Perform the subtraction with the following binary numbers using (i) 2's complement and (ii) 1's complement. Check the answer by straight subtraction.

   (a) 11010-1101, (b) 11010-1000, (c) 10010-10011, (d) 100-110000.

2. Represent the decimal number 8620 (a) in BCD, (b) in excess-3 code, (c) in 2, 4, 2, 1, and (d) as a binary number.

3. Obtain weighted binary code for the base-12 digits using weight of 5421.

4. Determine the odd-parity bit generated when the message consists of ten decimal digits in the 8, 4, -2, -1 code.

5. Explain floating-point representation of a real number with the help of a register with a capacity of six digits and a sign bit. Pictorially, illustrate the floating-point representation of -0.0801×10^{-4} by the said register.

6. What do you understand by normalized floating-point representation of a real number? Explain it. Evaluate $0.4568 \times 10^{8} - 1.234 \times 10^{7}$ using normalized floating-point representation and nine's complement.

7. Simplify the following Boolean functions by Karnaugh map method:

   (a) $f(x_1, x_2, x_3, x_4) = \sum(2,3,6,9,10,12,13,14) + \sum_{\phi}(3,11,15)$

   (b) $f(x_1, x_2, x_3, x_4) = \sum(4,6,7,15) + \sum_{\phi}(2,3,5,11)$

8. It is necessary to design an over heat alarm for an oil fired steam boiler. Three sensors are available. One sensor monitors the water temperature in the boiler, one monitors the chimney temperature and one follows on-off state of the burner. The following describes the logic operations of three sensors: An alarm signal should be generated whenever the burner is on and either the chimney or water temperature is too hot. Design the switching system of the above.

7. Suggested further Readings

1. M. Moris Mano, Digital Logic and Computer Design, Prentice-Hall of India, 2003.

2. V.Rajaraman and T.Radhakrishnan, An Introduction to Digital Computer Design, Prentice-Hall of India, 1987

VIDYASAGAR UNIVERSITY

DIRECTORATE OF DISTANCE EDUCATION
MIDNAPORE - 721 102

M.Sc. in Applied Mathematics with Oceanology and Computer Programming

Part-I
Group-C Paper-III
Module No. - 35

PROGRAMMING IN C-I

CONTENTS

Module-35: Programming in C

1.0 Introduction:

C is a higher-level language. It was developed in 1972 by Dennis Ritchie at Bell Telephone Laboratories. It gets its name from being the successor to the B language. Today there are C compilers available under every conceivable environment from DOS to Windows to Unix and on all sizes of computers from micros to mainframes. In 1988, it was standardized by the American National standards Institute (ANSI) and named as ANSI C.

C is a powerful language and it is used for many purposes like writing operating systems, business and scientific applications and even the C compiler itself. It is not tied to any particular hardware and programs written in C are portable across any system. The programs written in C are efficient and fast. It is one of the most popular computer language today.

2.0 Objectives

In this module, the main objectives are to study the followings

- character Set, the concept of constants and variables
- arithmetic, logical and relational operators
- conditional statements
- for statement, while statement, do-while statement, break and continue

3.0 Key Words and Study guides

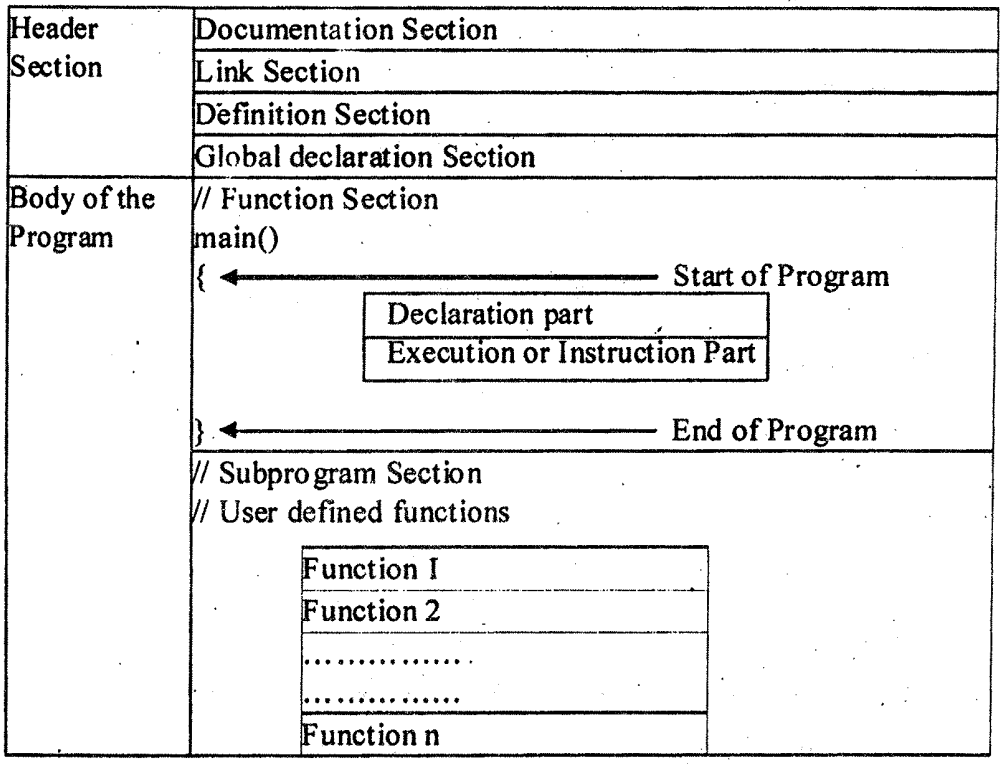
constant, variable, increment operator, decrement operator, type casting, ternary operator, swich statement, if statement, break and continue

4.0 Main Discussion

4.1 Basic of C Programming

4.1.1 Structure of C Programs:

A C program can be viewed as a group of building blocks called functions. A function is a subroutine that may include one or more statements designed to perform a specific task. To write a c-program, first functions are created and then they are put together. A c program may contain one or more section shown as follows:



(i) The **documentation section** consists of a set of comment lines giving the name of the program and other details. A comment is put in between

/* */

Example: /* Program to calculate roots of an algebraic equation */

(ii) The **link section** provides instructions to the compiler to link functions from the system library.

There is no keyword for the purpose of input/output in C language. Ready-made library functions are used for input/output. The library functions are classified into various groups. Each group is kept in a **header file.** The library used for I/O purposes are kept in the header file called ***studio.h*** (standard input/output). Similarly, mathematical functions are placed in a file called ***math.h***. These are placed in the link section at the beginning of the program. For example:

#include<studio. h>

#include<math. h>

Library functions of stdio.h are as follows:

scanf() printf() putchar() putc() puts() getchar()

Library functions of math.h are as follows:

cos() cosh() sin() sinh() tan() log( ) acos() asin() exp()

(iii) The **definition section** defines values to ail symbolic constants for use in the program. For example:

```
#define     PERIOD          10
#define     PRINCIPAL       2000.00
```

(iv) In the **global declaration section** some variables are declared which are used in more than one function.

(v) In the **body of the program** there is one ***main()*** function and one or more subfunctions which are optional. In ***main()*** function there is two parts, (i) declaration section and (ii) executable or instruction section.

In **declaration section**, it declares all the variables used in the executable part. Example: int roll; In executable section, there should beat least one statement in executable section. Normally, it consists of three parts-(i) input, (ii) processing and (iii) output.

These two sections of the main function must appear between the opening and the closing braces. The program execution begings at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. **All the statements in the declaration and executable sections end with a semicolon(;).**

The subprogram section contains all the user-defined functions that are called in the main function. User-defined functions are generally placed immediately after the main function, although they may appear in any order.

All sections, except the main function section may be absent when they are not required.

```
/* Sample program */
    #include<stdio.h>
    void main()
    {
    printf("Like C \n");
    exit ( 0 );
```

NOTE:

- C requires a semicolon at the end of every statement.
- printf is a standard C function — called from main.
- \n signifies newline. Formatted output — more later.
- Exit() is also a standard function that causes the program to terminate. Strictly speaking it is not needed here as it is the last line of main() and the program will terminate anyway.

Let us look at another printing statement:

printf(".\n..2\n ... 3\n");

The output of this would be:

.1

..2

... 3

4.1.2 Basic Structures:

A basic fact about computer programming is that all programs can be written using a combination of only three control structures:

- (i) Sequential structure
- (ii) Selective structure
- (iii) Repetitive structure

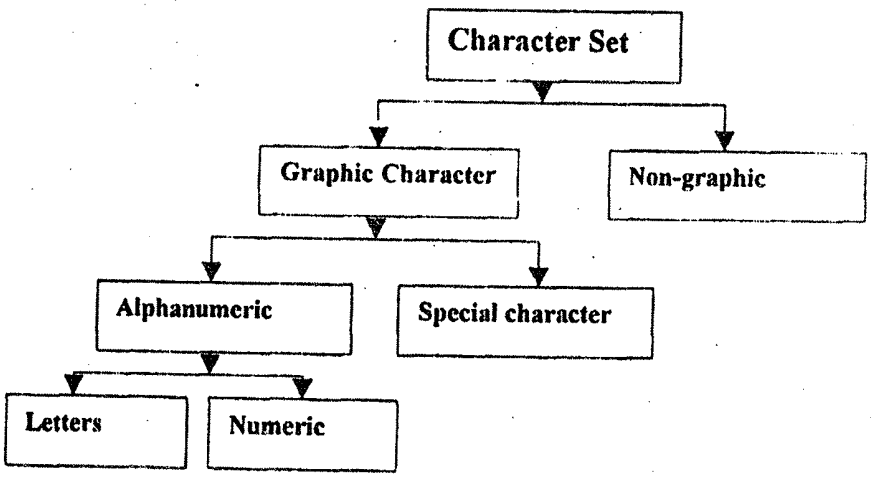
(i) **Sequential structure**: This structure consists of a sequence of program statements that are executed one after another in order

(ii) **Selective structure**: It consists of a test for a condition followed by alternative paths that the program can follow. The program selects one of the alternative paths depending upon the result of the test for the condition.

(iii) **Repetitive structure**: This structure consists of program statements that are repeatedly executed while some condition holds. It consists of an **entry point** that may include initialization of certain variables, a **loop continuation condition, a loop body**, and an **exit point.**

4.1.3 Character set:

The set of characters that may appear in a legal C program is called the character set. This set includes some (i) **graphic** as well as (ii) **nongraphic** characters. The graphic characters are those that may be printed and the nongraphic characters are represented by escape sequences consisting of a backslash (\) followed by a letter. The graphic characters other than decimal digits, letters, i.e., **alphanumeric characters**. Blank space, horizontal and vertical tabs, newlines, carriage return and formfeeds are called **whitespaces characters**.



Letters:							
Symbol				Meaning			
A-Z				Uppercase letters			
a-z				Lowercase letters			
Numeric:							
0-9				Decimal digits			
Special:							
,	comma	#	hash	~	tilde	)	Right parenthesis
.	period	'	apostrophe	>	Closing angle bracket	[	Left bracket
;	semicolon	"	Quotation mark	<	Opening angle braket	]	Right bracket
:	colon	|	Vertical bar	(	Left	{	Left brace

					parenthesis		
/	slash	_	underscore	%	Percent sign	&	ampersand
\	Back slash	$	Dollar sign	?	Question mark	^	caret
*	asterisk	-	Minus sign	+	Plus sign		
Non-graphic:(escape sequence)							
\a	Audible alert	\r	Carriage return	\b	Back space	\t	Horizontal tab
\f	Form feed	\v	Vertical tab	\n	newline	\\	Backslash
\'	singlequote	\''	doublequote	\?	Question mark	\0	null

Keywords:

C reserves certain names, called **keywords** for specific meanings and they can not used as variables names. The 32 standard keywords are given below.

auto, break, case, char, const, continuous, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, retun, short, signed, sizeof, static, struct, switch, typedef, union, unsiged, void, volatile, while.

Identifiers:

Identifiers are names given to various program entities. Examples of identifiers includes (1) function names, (ii) enumeration constants, (iii) symbolic constants, (iv) variables names, etc.

4.1.4 Constants:

A constant is an entity whose value does not change during program execution. Constants are of five different types: (1) interger, (ii) floating-point, (iii) character, (iv) string and (v) enumeration.

(i) interger constant: An integer constant is a number that h S an integer value. Integer constants may be specified in decimal, octal or hexadecimal notation. A decimal integer constant consists of a sequence of one or more decimal digits 0 through 9. The first digit of the sequence can not be 0, unless the decimal integer constant is 0. An **octal integer constant** consists of the digits 0, followed by a sequence of one or more octal digits 0 through 7. A **hexadecimal integer constant** consists of the digit 0, followed by one of the letters x or X, followed by a sequence of one or more hexadecimal digits 0 through 9, or letters a through f, or letters A through F .

Examples: 0 255 2147483647 012 077777 Ox1f OXABC

Note: Commas and spaces are not allowed in integer constants.

(ii) Floating-Point Constants: A floating-point constant is a number that has a real value which can be represented in two ways: (i) the **standard decimal form** of a floating-point constant is a number written with a deciaml point and (ii) the **scientific notation** is often used to express numbers that are very small or very large. This notation is as follows: **(coefficient) e (integer)** which is equivalent to **(coefficient) X $10^{integer}$**. The part appearing before e is called the mantissa and the part following e is called the **exponent**. The uppercase E can also be used instead of the lowercase e.

Examples: 1.0 0. .0 1.1e-8 2e10

(iii) Character constants: A character constant consists of a single character enclosed within apostrophes.

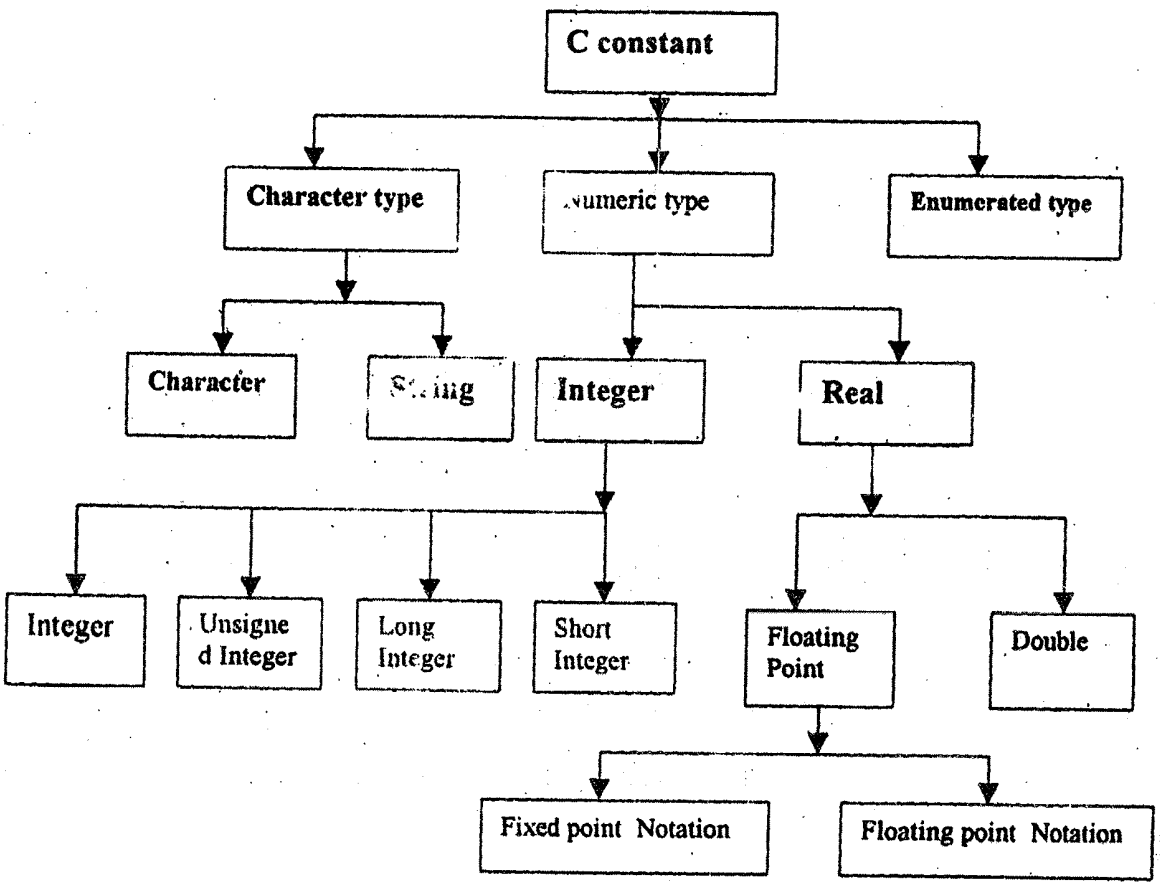
Examples: '0' 'a' 'Z' '?' '%'

(iv) String Constants: A string constant or simply a string consists of zero or more characters enclosed within double quotation marks. Non-graphic characters may also be used as part of a character string. The double quotation marks are not part of the string. They only serve to delimit it. If the double quotation mark is required to be one of the characters of the string, it must be preceded by a backslash (\). If the backslash is required to be a character of the string, it must be preceded by another backslash. The **length of a string** is the number of characters that form the string. There is no limit on the length of a string. Putting a backslash at the end of the line can continue a string.

Examples: "Programming in C is fun\n" "I'm a \" and I'm a\\"

"Great things are not done by impulse, \

but by a series of small things brought together"



4.1.5 Variables:

A variable is an entity used by a program to store values used in the computation. Simply it is tool to reserve space in computer's memory. The reserved space is given a name which we call a variable name. Constants are stored in this reserved space. The constants stored in the space reserved by the variable may vary from time to time, hence the variable name.

Rules for naming a variable name: C places some restrictions on what can be a variable name. These restrictions are as follows:

(i) A variable name must begin with a letter or underscore.

(ii) The remaining character except first character may be made up of any combination of letters, underscores, or digits 0-9.

(iii) Special characters and white spaces are not permitted within a variable name.

(iv) A variable name may use uppercase letters, lowercase letters, or both.

(v) The number of characters that can be used in a variable name is compiler-dependent. ANSI C permits at least 31 significant characters in variables names.

(vi) Keywords can not be used as variable names.

Examples: valid variable names are

Agent707 foo INTERSET interest

Every variable in C has a type and it must be declared before it is used. The different data types available in C are as follows:

C has the following simple data Types:

C type	Description	Size(bytes)	Lower bound	Upper bound
int	For an integer	2	-32768	+32767
unsigned int	For a positive integer	2	0	65536
long int	For a large integer	4	2	$+2^{31}-1$
char	For a character	1	—	—
unsigned char	For a character	1	0	255
float	For floating-point number	4	$-3.2 \times 10^{\pm 98}$	$+3.2 \times 10^{\pm 98}$
double	For floating-point number	8	$-1.7 \times 10^{+908}$	$+1.7 \times 10^{+908}$

Note-1: The qualifiers short and long may be applied to some of the above data types. Thus we have the following additional data types:

short int also abbreviated as short

long into also abbreviated as long

long double an extended-precision floating-point number

Note-2: The qualifiers signed and unsigned may be applied to char, short int, int, or long int. unsigned variables can only have nonnegative values while signed variables can have both positive and negative values. In the absence of explicit unsiged specification, int, short int and long int are considered to be signed.

Variables declarations: A declaration consists of a type name (data type) followed by a list of one or more variables of that type, separated by commas.

Examples: int mint;

char cherry;

double trouble;

float swim,salary;

This examples declare that the variable mint be of type int, the variable cherry be of type char etc.

4.1.6 Expression:

A combination of constants and variables together with operators is referred to as an expression. There are three types of operators: (i) arithmetic operators, (ii) relational operators and (iii) logical operators.

(1) Arithmetic Operators:

By arithmetic operators the arithmetic operations are done. The available arithmetic operators in C are as follows:

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder
+	Unary plus
-	Unary minus
++	increment
--	decrement

All these operators except reaminder (%) operate on operands of any arithmetic type. The reaminaer operator (%) takes only integral operands. The first five operators are binary operators and require two operands. When division is applied to integer data, the result is truncated to the integer value. The operator % obtains the remainder when one integer is divided by another. The increment and decrement operators are discussed latter.

Examples: 12+9=21 12.+9.=21.

12-9=3 12.-9.=3.

17*9=108 12.*9.=108.

12/9=1 12./9.=1.33

12%9=3

Arithmetic Expression: When an expression contain arithmetic operators then the expression is called arithmetic expression. Every expression has a value that can be determined by first binding the operands to the operators and then evaluating the expression. If an expression contains more than one operator and parentheses do not explicitly state the binding of operands, then according to the following precedence and associativity rule the operands are bounded to operators.

Operators	Type	Associativity
+ –	Unary	Right to Left
* / %	Binary	Left to Right
+ –	Binary	Left to Right

Example: (i) (x+2)*y/z-5, (ii) ((a-b)/c)%(d*(e+f)

(ii) Relational operators

C provides six relational operators shown below for comparing the values of two expressions and the expression so formed is called a relational expression.

Relational operator	Name	Relational Expression
<	Less than	expl < exp2
<=	Less than or equal to	expl <= exp2
>	Greater than	expl > exp2
>=	Greater than or equal to	expl >= exp2
= =	Equal to	exp1 = exp2
! =	Not equal to	expl exp2

Example: The exression

x==10;

test if the value of x is equal to 10.

(iii) Logical operators

C provides three logical operators for combining expressions into logical expression. The value of logical expression is either true (1) or false (0). In C, nonzero values indicate 1 and zero values indicate 0. A logical expression involves the following operators.

Operator	Name	Meaning
&&	Logical AND	Conjunction
||	Logical OR	Disjunction
!	Logical NOT	Negation

The actions of logical operators are as follows where T denotes true and F denotes False.

Example:

&&	T	F		||	T	F		!	
T	T	F		T	T	T		T	F
F	F	F		F	T	F		F	T

```
int a,b,c;
a=b=c=10;
```

the logical expression

```
a && (b+c)
```

has the value 1 (true) since both a and (b+c) are nonzero (true), whereas the logical expression

```
a && (b-c)
```

has the value 0 (false) since (b-c) is zer (false).

4.1.7 Assignment Statements:

An assignment statement is of the following form

```
variable=expression
```

Here the "=" symbol should not be interpreted in the same sense as in Mathematics. It is an operator that assigns the value of the expression on its right side to the variable on its left side. For example, the following two assignment statements

```
x=y;
```

and

```
y=x;
```

produce very different results. The first assigns the value of y to x leaving y unchanged, whereas the second assigns the value of x to y leaving x unchanged.

Example: x=i+j*y/4; is valid, but x+4=y*4+5*z; is unvalid.

4.1.8 Increment and decrement operators:

These two operators are used in two ways as: (i) prefix form and (ii) postfix form in the following manner:

```
++variable;
variable++,
—variable;
variable—;
```

Both in the prefix and postfix form, ++ adds 1 to its operand and — subtracts I from its operand. Difference is that in the prefix form, the value is incremented or decremented by 1 before it is used and in the postfix form, the value is incremented or decremented by 1 after the use. Example:

```
int i=1;
n=++i;
```

In this case, first increments the value of i to 2 and then sets n to the current value of i making n equal to 2. These statements are equivalent to the following:

```
int i=1;
n=i;
```

Example:

```
int i=1;
n=i++;
```

In these executions, first sets n to the present value of i,i.e., to 1 and then increments the value of i to 2. These are equivalent to

```
int i=1; n=i;
i=i+1;
```

4.1.9 Compound Assignment Operators:

C provides the following ten compound operators:

+= -= *= /= %= <<= >>= &= |= ^=

If we denote compound assignment operator by op=, then compressed form of the agginment statement

```
variable op= expression;
```

is equivalent to

```
variable=variable op expression;
```

Example:

i=i+1;	is	i+=i;
i=i*(a+1);	is	i*=a+1;
i=i<<1;	is	i<<1;

Note: The last five operators are from the bit operators.

4.1.10 Precedence of Operators:

The following table gives the procedence of all operators. Unary operators have the highest precedence, whereas the comma operator has the lowest precedence.

Description	Operator	Associativity
Increment/decrement	++ --	Right to left
Negation	!	Right to left
Unary minus	–	Right to left
Size in bytes	size of	Right to left
Multiplication	*	Left to right
Division	/	Left to right
Mod	%	Left to right
Addition	+	Left to right
Subtraction	–	Left to right
Less than	<	Left to right
Less than or equal to	<=	Left to right
Greater than Greater	>	Left to right
than or equal to	>=	Left to right
Equal to	= =	Left to right
Not equal to	! =	Left to right
Logical AND	&&	Left to right
Logical OR	| |	Left to right

Conditional	? :	Right to left
Assignment	= %= += *= /=	Right to left
Comma	,	Left to right

4.1.11 Input-Output:

We now present some input-output functions which interact with the standard input (usually keyboard) and the standard output (usually the screen).

(a) printf() function:

The general form of this function is

printf(control string, argument list);

where control string governs the conversion, formating and printing of the arguments. The control string include conversion specifications(format specifiers) that control the conversion of successive arguments before they are printed. So Control string consists of three types of items:

- Characters that will be printed on the screen as they appear
- Format specifications that define the output/input format for display of each item/for storing to the memory location
- Escape sequence characters

The argument list is the list of variables whose values are mormatted and printed according to the specifications of the control string. The arguments should match in number, order and type with the format specifications.

Format specifier	Type of argument	Output
%d	Integer	Signed decimal integer
%ld	Long integer	Decimal long integer
%o	Integer	Unsigned octal integer
%u	Integer	Unsigned decimal integer
%x	Integer	Unsigned hexadecimal integer
%X	Integer	Unsigned hexadecimal integer

%f	Floating Point	Signed value of the form [-] dddd.dddd
%e	Floating Point	Signed value of the form [-] d.dddd e[+/-]ddd
%c	Character	Character value
%s	string	String value

Example: printf("%f %c %d", a, b, c);

This statement results in the values assigned for a, b and c being displayed.

The control string contains the conversion specifications and thereby directs the entire operation of the printf function. The general form of format specification is as follows:

%w.d type_specifier

where w is an integer number that specifies the total number of columns for the output value and d is another integer that specifies the number of digits to the right of the decimal point of a real number or the number of characters to be printed from a string. Both w and d are optional. Example: The following program demonstrates on the possible variations of the %s conversion specification. It can be used with a field width specifier and the minus sign for left justification. The decimal places specifier is replaced by the maximum characters specifier which limits the number of characters from the string which are displayed.

```
#include<stdio.h>
void main(void)
{
    printf("%s\n","Vidyasagar University");
    printf("%25s\n","Vidyasagar University");
    printf("%-25s\n","Vidyasagar University");
    printf("%25.10s\n","Vidyasagar University");
    printf("%0.12s\n","Vidyasagar University");
    printf("%-25.10s\n","Vidyasagar University");
}
```

Output:

Vidyasagar University

Vidyasagar University

Vidyasagar University

Vidyasagar

Vidyasagar U

Vidyasagar

(b) scanf() function:

The general form of this function is

scanf (control string, argument list);

where control string contains governs the conversion specifications (format specifiers) according to which the characters from the standard input are interpreted and the results are assigned to the successive arguments. Each argument must be a pointer to the variable where the results of input are to stored. For the present, just we remember that if i is a variable, then &i is a pointer to i.

Examples: If the declarations are as follows

```
int    x;
float  y;
```

and we want to give the values 100 and 50.6 for x and y respectively, then the statement

scanf("%d%f",&x,&y);

results in x and y being assigned 100 and 50.6 respectively.

4.1.12 Coercion or Type-Casting:

C is one of the few languages to allow coercion that is forcing one variable of one type to be another type. C allows this using the cast operator (). So:

```
int integer number;
float float number=9.87;
integernumber(int)float number;
```

assigns 9 (the fractional part is thrown away) to integer number.

And:

```
int integernumber=10;
float floatnumber;
floatnumber=(float)integernumber;
```

assigns 10.0 to floatnumber.

Coercion can be used with any of the simple data types including char, so:

```
int integernumber;
char letter='A';
integernumber=(int)letter;
```

assigns 65 (the ASCII code for 'A') to integernumber.

Some typecasting is done automatically — this is mainly with integer compatibility.

A good rule to follow is: If in doubt cast.

Another use is the make sure division behaves as requested: If we have two integers internumber and anotherint and we want the answer to be a float then :

e.g.

```
floatnumber = (float) internumber / (float) anotherint;
```

ensures floating point division.

4.1.13 Enumerated Types:

An Enumeration a user defined type with values ranging over a finite set of identifiers called enumeration constants. Enumerated types contain a list of constants that can be addressed in integer values.

We can declare types and variables as follows.

```
enum days (mon, tues,..., sun) week;
enum days week1, week2;
```

NOTE: As with arrays first enumerated name has index value 0. So mon has value 0, rues 1, and so on. week 1 and week2 are variables.

We can define other values:

```
enum escapes 1 bell = '\a', backspace = '\b', tab ='\t', newline '\n', vtab '\v', return '\r'};
```

We can also override the 0 start value:

```
enum months Jan = 1, feb, mar, ..... dec};
```

Here it is implied that feb = 2 etc.

4.1.14 const Qualifier: The general syntax of it is as follows:

const data_type variable_name,

Example:

const float Tax_rate=0.3;

This informs the compiler that tax_rate is a constant floating point number, its value is 0.3 and it cannot be modified anywhere in the program. A statement such as

Tax_rate=0.99;

Will cause a compiler time error enabling you to detect the mistake early.

Macro: This is another way to achieve the effect of const qualifers, which is known as macro. The general form for a simple macro definition is

#define macro_name sequence_of_tokens

It associates with the macro_name whatever sequence_of_tokens appears from the first blank after the macro nameto the end of the line. These tokens constitute the body of the macro. For example:

#define ABC 100

Whenever a symbolic name i.e., macro is encountered, the compiler substitutes the value associated with the name automatically. To change the value, we have to simply change the definitions. A # define is a compiler directive and not a command. Therefore # define lines should not end with a semicolon. Symbolic constants are generally written in uppercase so that they are easily distinguished from lowercase variable names. # define instructions are usually placed at the beginning before the main() function. Symbolic constants are not declared in declaration section.

4.1.15 Comma operator:

A setoff expressions separated by commas is a valid construct in the C language. For example, i and j are declared by the statement:

int i,j;

Consider the following statement that makes use of the comma operator:

i=(j=3, J+2);

The right hand side consists of two expressions separated by commas. The first expression is j=3 and the second is j+2. These expressions are evaluated from left to right. That is, first the value 3 is assigned to j and then the

expression j+2 is evaluated, giving 5. The value of the entire comma-separated expression is the value of the right=most expression. In this example, the value assigned to I would be 5.

4.1.16 Programming Examples:

Problem-1: Write a program to convert of temperature from centigrade to farenheit and vice-versa.

```
#include<stdio.h>
void main()
{
        float c,f;
        printf("Enter temperature in celsius: ");
        scanf("%f",&c);
        f=1.8*c+32;
        printf("Equivalent fahrenheit=%f\n",f);
        printf("\nEnter temperature in fahrenheit:");
        scanf("%",&f);
        c=(f-32)/1.8;
        printf("Equivalent celsius=%f\n",c);
}
```

Input/Output:

```
Enter temperature in celsius: 30
Equivalent fahrenheit=86.000000
Enter temperature in fahrenheit: 40
Equivalent celsius=4.44445
```

Problem-2: Write a program to find the area of a triangle given the three sides.

```
#include<stdio.h>
#include<math.h>
void main()
{
    float a,b,c;
```

```
        float peri,s,area;
        printf("Enter the three sides:"),
        scanf("%f%f%f', &a,&b,&c),
        peri=a+b+c;
        s=peri/2;
        area=sqrt(s*(s-a)*(s-b)*(s-c));
        printf("Area of the triangle is %f\n',area);
}
```

Input/Output:

Enter the three sides. 2 3 4

Area of the triangle is 2.904737

Problem-3: Write a program to test a number is divisible or no', by 7.

```
include<stdio.h>
#include<conio.h>
void main()
{
        int x,r,m;
        clrscr();
        print("Enter the number
        scanf("%d".&x);
        r=x/7;
        m=r*7;
        if(x==m)
                printf("%d is a divisible by 7\n",x);
        else
                printf("%d is not divisible by 7",x);
}
```

Input/Output:

Enter the number

12

12 is not divisible by 7

Exercises

Exercise 1. Explain the basic data types in c and what is the size in bytes of each type.

Exercise 2. What are the different types of operators in c?

Exercise 3. What do you understand by operator precedence?

4.2 Conditionals:

This Chapter deals with the various methods that C can control the flow of logic in a program. Apart from slight syntactic variation they are similar to other languages. As we have seen following logical operations exist in C:==, !=,||,&&.

One other operator is the unitary - it takes only one argument - not !,

These operators are used in conjunction with the following statements.

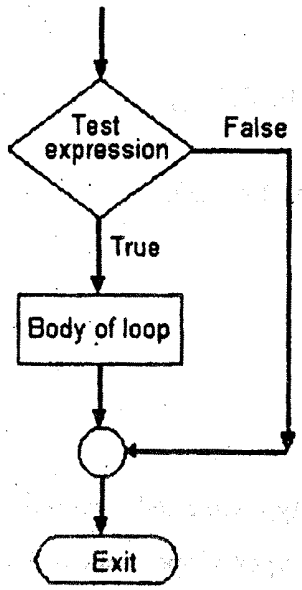
4.2.1 The if statement:

The if statement is used to specify conditional execution of a program statement or a group of statements enclosed in braces. The general format of this statement is:

```
if(expression)
    statement
```

When an if statement is encountered, expression is evaluated and if its value is nonzero (true), then statement is executed. After the execution of statement, the statement following the if statement is encountered next. If the value of expression is zero (false), statement is not executed and the execution continues from the statement immediately after the if statement.

The general flowchart is given below:



Example: The following program uses an if statement to perform this action.

```
#include<stdio.h>
void main(void)
{
int v,v2,larger;
scanf("%d %d",&v 1,&v2);
larger=v 1;
if(v2>v1)
  larger=v2;
printf("Large = %d\n",larger);
}
```

Input/Output:

34 50

Large = 50

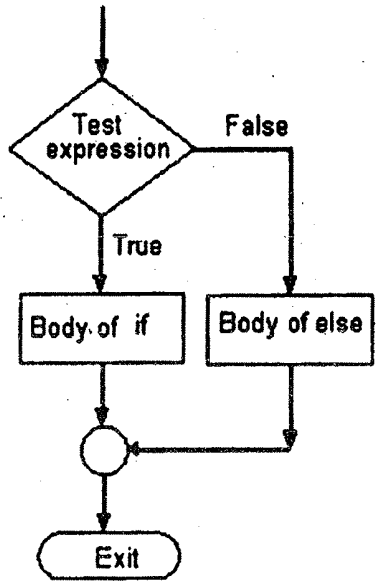
4.2.2 The if-else statement:

There are many situations when two groups of statements are given and it is desired that one of them be executed if some condition is true and the other be executed if the condition is false. Its general form is:

```
if (expression)
  statement,
else
  statement2
```

When an if-else statement is encountered, the value of expression is evaluated and if its value is nonzero (true), statement1 is executed. After the execution of statement1, the program execution continues from the statement immediately after statement2. If the value of expression is zero (false), statement2 is executed. After the execution of statement2, the program execution continues from the statement following statement2. In either case, one of statement1 or statement2 is executed, but not both.

The general flowchart is as follows:



Example: The following program uses an if-else statement to perform this action.

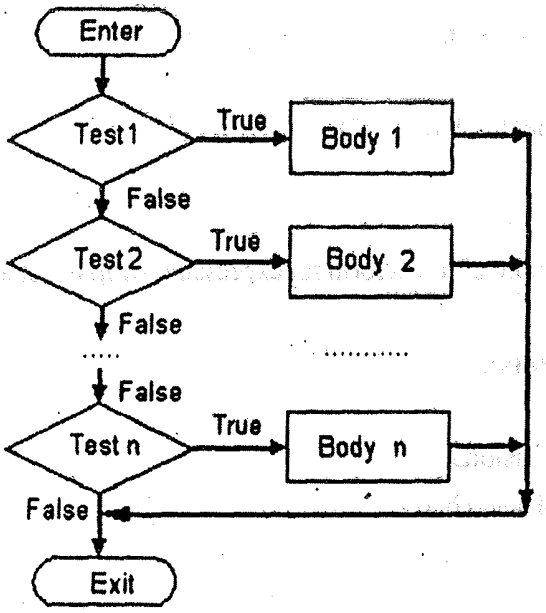
```
#include<stdio.h>
void main (void)
{
int v1, v2, larger,
scant("%d %d", &v1, & v2);
if(v1>v2)
        large=v1;
else
        large=v2'
print ("%d\n",larger);
}
```

4.2.3 The multi-way conditional statement:

The multi-way decision arises when there are multiple conditions and different statements are to be executed under each condition. The general format of this statement is:

```
if(expression-1)
        statement1
```

```
else if (expression-2)
        statement2
else if (expression-3)
        statement3
        .
        .
        .
else if (expression-(n-1)
```




```
        statement n
```

The general flowchart is given below:

In a multi-way conditional statement, conditional expressions are evaluated in order. If any of these expressions is found to be true, the statement associated with it is executed, and this terminates the whole chain. If none of the expressions is true the statement associated with the final else is executed. If no processing is required when none of the expression is true, this else along with the statement associated with it can be omitted.

For example:-

```
#include<stdio.h>
void main(void)
{
        int i,j,k;
        scanf("%d %d %d",&i,&j,&k)
        if(i==i)
          printf("I and j are equal.\n");
        else if(i==k)
          printf("I and k are equal.\n");
        else if(i==1)
          printf("I and 1 are equal.\n");
        else
          printf("I is not equal to j,k or 1.\n");
}
```

4.2.4 The ? operator:

The '? (*ternary condition*) operator is a more efficient form for expressing simple if statements. It has the following form:

$expression_1$? $expression_2$: $expression_3$

It simply states:

if $expression_1$ then $expression_2$ else $expression_3$

For example to assign the maximum of a and b to z:

```
z = (a>b) ? a : b;
```

which is the same as:

```
if (a>b)
  z = a;
else
  z=b;
```

4.2.5 The switch statement

The C switch is similar to Pascal's case statement and it allows multiple choice of a selection of items at one level of a conditional where it is a far neater way of writing multiple if statements. Its general form is:

```
switch (expression)
 {
```

```
case item1: statement1;
            break;
case item2: statement2;
            break;
            ,,
            ,,
            ,,
case item_r: statement_n;
            break;
default:    statement;
            break;
}
```

In each case the value of item, must be a constant, variables ate not allowed.

The break is needed if you want to terminate the switch after execution of one choice. Otherwise the next case would get evaluated. The default case is optional and catches any other cases.

For example:-

```
switch (letter)
    {
    case 'A'   :
    case 'E'   :
    case 'I'   :
    case 'O'   :
    case 'U'   : number of vowels++;
                 break;
    case       : numberofspaces++;
                 break;
    default    : numberofconstants++,
                 break;
    }
```

In the above example if the value of letter is 'A', 'E', 'I', 'O' or 'U' then number of vowels is incremented. If the value of letter is '' then number of spaces is incremented. If none of these is true then the default condition is executed, that is number of constants is incremented.

4.2.6 Compound Statement:

A compound statement is a set of statements enclosed within a pair of curly braces, { and }. An example of a compound statement is an if statement in which a set of statements enclosed within { and } are executed when the condition is true.

```
.....
.....
if (success!=0)
  {
    secs=years*365*24*60*60,
    printf("You have lived for %f seconds\n",secs);
    }
.....
.....
```

4.2.7 Programming Examples:

Problem-1: Write a program to find the real roots of a quadratic equation.

The roots of the quadratic equation aX2+bx+c=0, where a,b and c are real and a is not equal to zero, are given by

$$\left(-b \pm \sqrt{b^2 - 4ac}\right) \Big/ 2a$$

The roots are equal when the discriminant $b^2-4ac \geq 0$. If $b_2-4ac=0$, the roots are real and equal.

```
#include<stdio.h>
#include<math.h>
int main()
{
      float a,b,c;
      float dis, root, root 1, root2;
      printf("Enter the coefficients of a quadratic equation: ");
      scanf("%f %f %f",&a,&b,&c);
      if(a==0)
      {
            printf("Not a quadratic equation\n");
            return 1;
      }
      dis=b*b-4*a*c;
      if(dis<0)
            printf("No real roots\n");
      else if(dis==0)
      {
            root=-b/(2*a);
            printf("Two identical roots: %f/n",root);
}
else
{
```

```
            root 1=(–b+sqrt(dis))/(2*a);
            root2=(-b-sqrt(dis))/(2*a);
            printf("Two distinct roots: %f%f\n",rootl,root2);
        }
    return 0;
}
```

Input/Output:

Enter the coefficients of a quadratic equation: 1 6 2

Two distinct roots: -0.354249 -5.645751

Exercises

Exercise 1: Write a program to read two characters, and print their value when interpreted as a 2-digit hexadecimal number. Accept upper case fetters for values from 10 to 15.

Exercise 2: Read an integer value. Assume it is the number of a month of the year, print out the name of that month.

Exercise 3: Given as input three integers representing a date as day, month, year, print out the number day, month and year for the following day's date.

Typical input: 28 2 1992 typical output: Date following 28:02:1992 is 29:02:1992

Exercise 4: Write a program which reads two integer values. If the first is less than the second, print the message up. If the second is less than the first, print the message down If the numbers are equal, print the message equal If there is an error reading the data, print a message containing the word Error and perform exit(0);

4.3 Looping and Iteration:

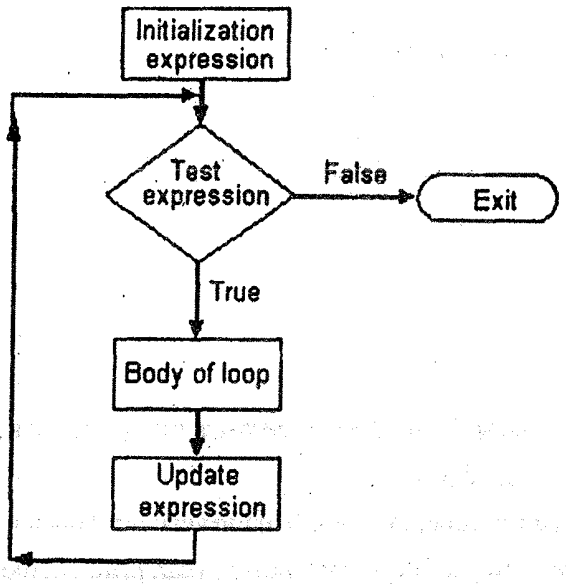
This chapter will look at C's mechanisms for controlling looping and iteration. Even though some of these mechanisms may look familiar and indeed will operate in standard fashion most of the time.

4.3.1 The for statement :

The C for statement has the following form:

```
for (expression-1; expression-2; expression-3)
    statement-1
    statement-2
```

The general flowchart is as follows:



expression-1 initializes i.e., defines the entry point of the structure; expression-2 provides the loop continuation condition, expression-3 is the modifier (which may be more than just simple increment). Stamens-1 is any valid simple or compound C statement whose execution is followed by a evaluation of expression-3. Statement-1 together with expression-3 constitute the loop body. Statement-2 is the exit point of the structure.

For example:

```
        main()
        {
        ins x;
          for (x=3;x>0;x-)
        {
        printf("x=%d\n",x);
        }
    }
```

...outputs:

```
        x=3
        x=2
        x=1
```

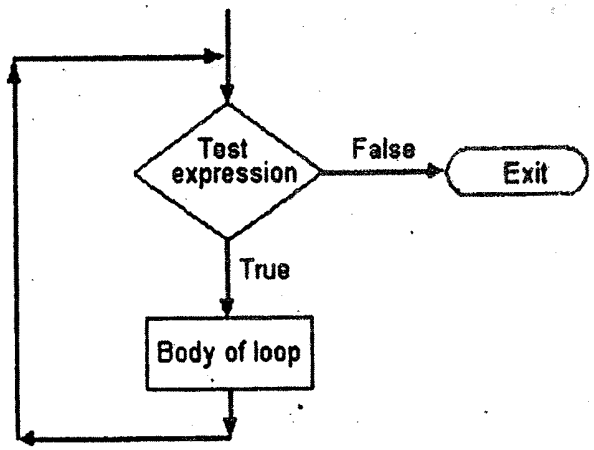
...to the screen

4.3.2 The while statement :

The while statement is pre-test loop whose general form is:

```
while (expression)
statement-1
statement-2
```

The flowchart is given below:



When the while statement is encountered, expression is evaluated. If expression evaluates to a nonzero value (true), statement-1 is executed. Expression is then again evaluated, and if the result of this evaluation is nonzero, statement-1 is again executed. This process continues until the result of expression evaluation becomes zero (false). The iteration is then terminated and the program execution continues from statement-2. If the result of expression evaluation is zero at the very outset, statement-1 is not executed at all and the program control immediately transfers to the statement-2.

For example:

```
main()
{
      int x=3;
      while (x>0)
      printf("x=%d\n",x);
            x-;
  }
}
```

...outputs:

x=3

x=2

x=1

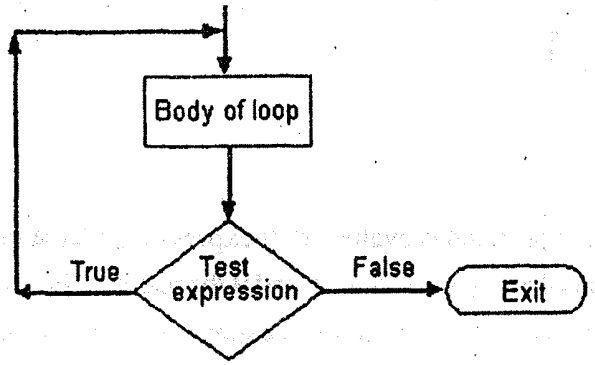
...to the screen.

4.3.3 The do-while statement :

The do-while loop is a post-test loop, whose general form is:

```
do
    statement-1
while (expression);
statement-2
```

The general flowchart is given below:



When the keyword do is encountered, statement-I is executed, followed by an evaluation of expression. If the result of the expression evaluation is nonzero, statement-1 is again executed. This process continues until the result of the expression evaluation becomes zero. The iteration is then terminated and the program execution continues from statement-2. If the result of the expression evaluation is zero at the very outset, the program control transfers to the statement-2. Thus, in this case, statement-1 is executed only once.

For example:

```
main()
{
    int x=3;
    do {
    printf("x=%d\n",x-);
     }
```

```
while (x>0);
}
```

..outputs:-

x=3

x=2

x=1

4.3.4 break and continue:

C provides two commands to control how we loop:

- break — exit form loop or switch.
- continue — skip 1 iteration of loop.

break statement

When a break statement of the form

break;

is encountered within a loop body, the execution of the loop body is interrupted and the program control transfers to the exit point of the loop.

For example, we consider the following example:

```
while(1)
{
scanf("%d",&j);
if(j==-1)
break;
sum+=j;
}
```

Whenever -1 is input, the condition j ==-1 evaluates to true, and the break statement is executed. This statement causes the execution of the loop to be terminated. Control passes to the statement following the while construct. Notice that the condition in the while loop has been specified as 1 which is nonzero and hence is always true. The condition specifies an infinite loop, but the break terminates its execution.

continue statement

The continue statement is another loop interruption statement but unlike break it does not terminate a loop, it only interrupts a particular iteration. the continue statement is of the form

continue;

A break and continue is usually associated with a if statement. The following examples shows break and continue at work.

Example:

We consider the following example where we read in integer values and process them according to the following conditions. If the value we have read is negative, we wish to print an error message and abandon the loop. If the value read is great than 100, we wish to ignore it and continue to the next value in the data. If the value is zero, we wish to terminate the loop. Therefore the programming code of this is as follows:

```
while (scanf("%d", &value) == 1 && value !=0) {
if (value < 0) {
 printfi("Illegal value\n");
break;
/* Abandon the loop */
}
if (value > 100) {
printf("Invalid value\n");
continue,
        /* Skip to start loop again */
}
/* Process the value read */
/* guaranteed between 1 and 100 */
        ……;
……;
} /* end while value != 0 */
```

goto Statement: The goto statement is used to alter the normal sequence of program execution by unconditionally transferring control to some other part of the program. The general syntax of this statement is as follows:

goto label;

where label isan identifier used to label the target statement to which the control would be transferred. The target statement must be labeled and it must be followed by a colon. The target statement will appear as follows:

label: statement;

Example: In this example, for and gota structures that determine whether two arrays x and y have an element in common. The array x has n elements and y has m elements.

```
for(i=0;i<n;i++)
for(j=0j<m;j++)
if(x[i] = =y[j]
goto found;
```

```
        .....
        .....
        found:
        printf("The both have common element");
        .....
        .....
```

4.3.5 Programming Example:

Problem-1: Write a program to find factorial of a number by non-recursive method.

```
#include<stdio.h>
void main()
{
        int n,i,fact;
        printf("Enter the number: ");
        scanf("%d",&n);
        if(n==0)
                printf("Factorial of 0 is 1 \n");
        else
                {
                fact= 1
        for(i=1;i<=n;i++)
                fact = fact*i;
        printf("Factorial of %4d is %5d",n,fact);
        }
}
```

Input/output:

Enter the number: 4

Factorial of 4 is 24

Problem-2: Write a program to generate PASCAL's triangle.

```
#include<stdio.h>
void main()
{
        int binom,p,q,r,x;
        binom=1;
```

```
        q=0;
        printf("Input the number of rows: "); scanf("%d",&p);
        printf("Pascals triangle An");
        while(q<p)
        {
        for(r=40-3*q;r>0;—r)
         printf(" ");
        for(x=0;x<=q;++x)
        {
                if((x==0) || (q==0))
                  binom=1;
                else
                  binom=(binom* (q-x+1))/x;
                printf("%6d",binom);
        }
    printf("\n"),
    ++q;
    }
}
```

Input/Output:

Input the number of rows: 4

Pascals triangle:

```
          1
        1   1
      1   2   1
    1   3   3   1
```

Problem-3: Write a program to check for a palindrome.

```
#include<stdio.h>
void main()
{
```

```
int n,num,digit,sum=0,rev=0;
printf("Input the number: ");
scanf("%d",&num);
n=num;
do
{
    digit=num%10;
    sum+=digit:
    rev.=rev* 10+digit;
    num/= 10;
    }while(num!=0);
    printf("Sum of the digits of the number—%4d\n",sum);
    printf("Reverse of the number=%7d\n",rev);
    if(n= =rev)
        printf("The number is a palindrome\n");
    else
        printf("The number is not a palindrome\n");
}
```

Input/Output:

Input the number : 12321
Sum of the digits of the number = 9
Reverse of the number = 12321
The number isapalindrome

5.0 Unit Summary

In this module, the structure of C Programs, the character Set, the concept of constants and variables, expression, assignment Statements, increment and decrement operators, compound assignment operators, the input/Output statement, type Casting, enumerated types, if statement, if-else statement, multi-way conditional statement, ternary condition, the switch statement, compound Statement, for statement, while statement, do-while statement, break and continue are discussed.

6.0 Self Assessment Questions

Exercises

Exercise 1: Write a program to read in 10 numbers and compute the average, maximum and minimum values.

Exercise 2: Write a program to read in numbers until the number -999 is encountered. The sum of all number read until this point should be printed out.

Exercise 3: Explain the difference between while and do-while.

Exercise 4: Explain the difference between a break statement and a continue statement with examples.

Exercise 5: Write a program to generate the first hundred terms of the Fibonacci series and print their sum and average.

Exercise 6: Explain the role of the initialization, test and update expressions in a for loop. When is each of them executed?

Exercise 7: Write a program to input a string, sum up the ASCII values of all its characters and output the resulting value.

7.0 Suggested further Readings

Balgurusamy, E., Programming in ANSI C, Tata McGraw Hill, 1992

Venugopal, K.R., Programming with C, Tata McGraw Hill, 2001

Kumar, R. and Agarwal, R., Programming in ANSI C, Tata MCGraw Hill, 1993

M.Sc. Course
in
Applied Mathematics with Oceanology
And
Computer Programming
Part-I

Paper-III **Group-C**

Module No. - 36
PROGRAMMING IN C-II

CONTENTS

Module-36 : Programming in C

1.0 Introduction

Very often, one needs to process a collection of related data items, such as test scores of students in a University. One way of handling such a situation would be to invent a new variable name for each of these data items. A notation, called subscript notation, exists in Mathematics to handle such situations C provides a capability similar to the subscript notation that enables you to structure and process a set of ordered data items. Pointers enable us to achieve parameter passing by reference, deal concisely and effectively with arrays, represent complex data structures, and work with dynamically allocated memory. C provides a facility, called structures, that allows a fixed number of data items, possibly of different types, to be treated as a single object.

2.0 Objectives

In this module, the following topics are covered .

* the concepts of arrays and strings,
* the concept of functions, the storage classes, the concept of recursion,
* the structures and union,
* the concept of pointers

3.0 Key Words and Study guides

Array, string, function, storage class, recursion, structure, union, and pointer.

4.0 Main Discussion

4.1 Arrays and strings

An ordered finite collection of data items, each of the same type, is called an **array** and the individual data items are its **elements.** Only one name is assigned to an entire array and the individual elements are referenced by specifying a subscript. A subscript is also called an **index.** In C subscripts start at 0, rather than 1, and can not negative. An array has the following properties:

* The type of an array is the data type of its elements.

* The location of an array is the location of its first element.
* The length of an array is the number of data elements in the array.
* The size of an array is the length of the array times the size of an element.

4.1.1 Single and Multi-dimensional Arrays :

Arrays whose elements are specified by oen subscript are called single-subscripted, linear or one-domensional arrays. Let us first look at how we define one-dimensional arrays in C :

int listofnumbers[50];

BEWARE : In C Array subscripts start at **0** and end one less than the array size. For example, in the above case valid subscripts range from 0 to 49. This is a **BIG** difference between C and other languages and does require a bit of practice to get in ***the right frame of mind.*** Elements can be accessed in the following ways:-

thirdnumber=listofnumbers[2];

listofnumbers[5]=100;

The array whose elements are specified by two or more subscripts are called two-dimensional or multi-dimensional array. **Multi-dimensional arrays can be defined as follows :**

int tableofnumbers[50][50];

for two dimensions.

For further dimensions simply add more [ ]:

int bigD[50][50][40][30]....[50];

Elements can be accessed in the following ways :

anumber=tableofnumbers[2][3];

tableofnumbers[25][16]=100;

Elements of an array can be assigned initial values by following the array definition with a list of initializers enclosed in braces and separated by commas. For example, the declaration

int mark[5]={34,90,95,86,56};

defines the array **mark** to contain five integer elements and initializes mark[0] to 34, mark[10] to 90, mark[2] to 95, mark[3] to 86 and mark[4] to 56.

4.1.2 Strings

In C Strings are defined as arrays of characters. For example, the following defines a string of 50 characters:

```
char name[50];
```

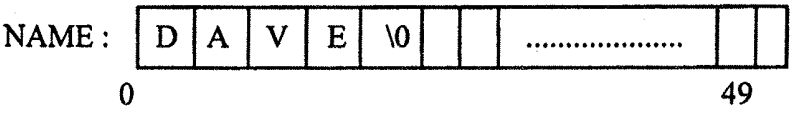
There are a special library of string handling routines which we will come across later.

To print a string we use printf with a special **%s** control character :

```
printf("%s", name);
```

NOTE : We just need to give the name of the string.

In order to allow variable length strings the \0 character is used to indicate the end of string. So we if we have a string, char NAME[50]; and we store the "DAVE" in it its contents will look like:



Now the different stirng handling functions commonly used are as follows:

(i) **Strcat() function :** This function joins two strings together. Its general form is

```
strcat(string1,string2);
```

where string1 and string2 are character arrays. When the function is executed, then string2 is appended to string1. It does so by removing the null character at the end of string1 and placing string2 from there. Here, it is to be sure that the size of string1 (to which string2 is appended) is large enough to accommodate the final string.

(ii) **strcmp() function :** This function compares two strings identified by the arguments and has a value 0 if they are equal. If they are not, then it has the numeric difference between the first non-matching characters in the strings. Its general form is

```
strcmp(string1,string2);
```

where string1 and string2 may be string variables or string constants. It compares string1 and string2.

(iii) **strcpy() function :** This function copies the contents of one string to another. It takes two arguments, the first being the pointer to the beginning of the destination string and the second is a pointer to the first element of the source string. Its general form is

```
strcpy (string1,string2);
```

This function assigns the contents string2 to string1. string2 may be a character array variable or a string constant.

Example :

```
#include<stdio.h>
#include<string.h>
void main()
{
char name1[25],name2[25];
printf("Enter one string: ");
gets(name1);
strcpy(name2,name1);
printf("The copies string is\n");
puts(name2);
}
```

Input /Output:

```
Enter one string mathematics
The copied string is
mathematics
```

(iv) **strlen() function :** Its general form is

n=strlen(string);

This function counts and returns the numbers of characters in a string. Here n is an integer variable, which receives the value of the length of the string. The argument may be a string constant. The counting ends at the first null character.

Example :

```
#include<stdio.h>
#include<string.h>
void main()
{
char name[25];
int length;
printf("Enter a name: ");
gets(name1);
length=strlen(name);
printf("The total number of characters in %s is %d", name, length);
}
```

Input/Output:

Enter a neme : Vidyasagar

The total number of characters in Vidyasagar is 10

Expect these there are further many string handling functions displayed in the following tables.

Sr. No.	Function	Action
1	strstr(str1,str2)	Checks if str2 is a substring of str1
2	strlwr(str)	Converts all characters in str to lower case
3	strupr(str)	Conversts all characters in str to upper case
4	strrev(str)	Reverses string str
5	strchr(str,c)	Searches for first occurrence of character c, in string str returns the pointer to c, if found, returns NULL otherwise
6	strset(str,c)	Initializes string str with all the characters as c

4.1.3. Character and String Input and Output :

The following are the standard character and string input and output functions which interact with the standard input i.e., keyboard and the standard output i.e., the screen.

(i) getchar() : This function reads a character from the standard input device (stdin). The geneal syntax is as follows

```
int getchar(void);
```

On success, getchar returns the character read, after converting it to an int without sign extension. On end-of-file or error, it returns EOF.

Example :

```
#include<stdio.h>
int main(void)
{
int c;
/*Note that getchar reads from stdin and is line buffered; this means it will not return until you press ENTER.*/
```

```
while ((c = getchar()) !'\n')
        printf("%c",c);
return 0;
}
```

Input / Output :

midnapur
midnapur

(ii) putchar() : It writes a character to the standard output device (stdout). putchar. The general form is

```
int putchar(int c);
```

On success, putchar returns the character c. On error, puchar returns EOF.

Example : /*define some box-drawing characters*/

```
#include<stdio.h>
#define LEFT_TOP 0xDA
#define RIGHT_TOP 0xBF
#define HORIZ 0xC4
#define VERT 0xB3
#define LEFT_BOT 0xC0
#define RIGHT_BOT 0xD9

int main (void)
{
        char i,j;

/* draw the top of the box */
putchar(LEFT_TOP);
for (i=0;i<10;i++)
        putchar(HORIZ);
. putchar(RIGHT_TOP);
putchar('\n);


/*draw the middle */
for (i=0, i<4; i++)
{
putchar(VERT);
for (j=0;j<10;j++)
putchar(' ');
```

```
puthar(VERT);

putchar('\n');
}
/* draw the bottom */
putchar(LEFT_BOT);
for(i=0;i<10;i++)
        putchar(HORIZ);
putchar(RIGHT_BOT);
putchar('\n');

return 0;
}
```

(iii) getc(): It gets character from stream. The general syntax is as follows

```
int getc(FILE *stream);
```

getc returns the next character on the given input stream and increments the stream's file pointer to point to the next character. On success, getc returns the character read, after converting it to an int without sign extension. On ed-of-file or error, it returns EOF.

Example :

```
#include<stdio.h>
int main(void)
{
char ch;
printf("Input a character:");
        /* read a character from the standard input stream */
ch = getc(stdin);
printf("The character input was: '%c'/n",ch);
return 0;
}
```

(iv) putch(): It outputs a character to a stream. The general form is

```
int putc(int c, FILE *stream);
```

putc is a macro that outputs the character c to the stream given by stream.

On success, putc returns the character printed. c. On error, putc returns EOF.

4.1.4 Programming Example :

Problem-1 : Write a program that cyclically permutes the elements of a given sequence.

Given a sequence $x_1, x_2, ..., x_n$, its cyclic permutation is defined to be the sequence $x_2, x_3, ..., x_n, x_1$. The desired program is as follows.

```
#include<stdio.h>
void main(void)
{
      int a[10],temp;
      int i;
      printf("Enter the sequence: ");
      for(i=0;i<10;i++)
      scanf("%d",&a[i]);
      temp=a[0];
      for(i=1;i<10;i++)
      a[i-1]=temp;
      for(i=0;i<10;i++)
      printf("%df\t",a[i]);
      }
```

Input/Output :

Enter the sequence : 1 2 3 4 5 6 7 8 9 10

2 3 4 5 6 7 8 9 10 1

Problem-2 : Write a program to find maximum and minimum from a list of numbers.

```
#include<stdio.h>
void main(void)
{
int n,i;
printf("Enter how many data you can store: ");
scanf("%d",&n);
float a[25],max,min;
printf("Enter the data");
for(i=0;i<n;i++)
      scanf("%f",&a[i]);
max=a[0];
min=a[0];
for(i=1,i<n;i++)
```

```
{
if(a[i]>=max)
        max=a[i];
if(a[i]<=min)
        min=a[i];
}
printf("max=%of\n",max);
printf("min=%f\n",min);
}
```

Input / Output :

Enter how many data you can store : 5
100 50 120 12 40
max = 120.000000
min = 12.000000

4.2 Functions :

C provides functions which are again similar most languages. One difference is that C regards main() as function. Also unlike some languages, such as Pascal, C does not have ***procedures*** -- it uses functions to service both requirements.

A function definition introduces a new function by declaring the type of value it returns and its parameters, and specifying the statements that are executed when the function is called. The general format of the function definition is :

returntype **fn_name(*parameterdef1*,...)**
{
localvariables
functioncode
}

where ***returntype*** specifies the type of the function and corresponds to the type of value returned by the function. A function that does not return any value but only causes some side effects is declared to be of type void. The specification of function type is optional, it is omitted, it is taken to be int.

fn-name is the name of the function being defined. ***parameterdef1*** specify the types and names of the parameters(also called formal parameters) of the functions, separated by commas. If a function does not have any

parameters, keyword void is written in place of parameter declarations. The rules for naming functions and parameters are the same as for naming variables.

Let us look at an example to find the average of two integers :

```
float findaverage(float a, float b)
    {
    float average;
    average=(a+b)/2;
    return(average);
    }
```

We would ***call*** the function as follows:

```
main()
{float a=5,b=15, result;
result=findaverage(a,b);
printf("average=%f\n",result);
}
```

Note: The return statement passes the result back to the main program.

4.2.1 Functions and Arrays :

Single dimensional arrays can be passed to functions as follows:-

```
float findaverage(int size, float list[])
{
        int i;
        float sum=0.0;
        for (i=0;i<size;i++)
            sum+=list[i];
        return(sum/size);
}
```

Here the declaration float list[] tells C that list is an array of float. **Note** we do not specify the dimension of the array when it is a ***parameter*** of a function.

Multi-dimensional arrays can be passed to functions as follows:

```
void printtable(int xsize,int ysize,float table[][5])
{ int x,y;
for (x=0;x<xsize;x++)
{ for (y=0;y<ysize;y++)
printf("\tf",table[x][y]);
printf("\n");
}
}
```

Here float table[][5] tells C that table is an array of dimension NX5 of float. **Note** we must specify the second (and subsequent) dimension of the array but not the first dimension.

4.2.2 Function Prototyping :

Before you use a function C must have ***knowledge*** about the type it returns and the parameter types the function expects.

The ANSI standard of C introduced a new (better) way of doing this than previous versions of C. (Note : All new versions of C now adhere to the ANSI standard.)

The importance of prototyping is twofold.

* It makes for more structured and therefore easier to read code.
* It allows the C compiler to check the ***syntax*** of function calls.

How this is done depends on the scope of the function. Basically if a functions has been defined before it is used (called) then you are ok to merely use the function.

If NOT then you must ***declare*** the function. The declaration simply states the type the function returns and the type of parameters used by the function. It is usual (and therefore **good**) practice to prototype all functions at the start of the program, although this is not strictly necessary.

To ***declare*** a function prototype simply state the type the function returns, the function name and in brackets list the type of parameters in the order they appear in the function definition.

e.g.

```
int strlen(char []);
```

This states that a function called strlen returns an integer value and accepts a single string as a parameter.

NOTE : Functions can be prototyped and variables defined on the same line of code. This used to be more popular in pre-ANSI C days since functions are usually prototyped separately at the start of the program. This is still perfectly legal though: order they appear in the function definition.

e.g.

```
int length, strlen(char []);
```

Here length is a variable, strlen the function as before.

4.2.3 return statement :

A **return** statement can be of one of the following two forms :

```
return expression;
return;
```

If the return statement is of the first form, the value of the *expression* is returned to the calling function.

For example, the function **larger**

```
float larger(float x, float y)
{
        return x>y?x:y;
}
```

returns the value of the larger of the two arguments in the function call.

The return statement of the second form returns no value to the calling function. This form of the return statement should be used only when the function is of type void.

4.2.4 Call by value and call by reference parameter :

C only provides *call by value* parameter passing, meaning thereby that the called function is only provided with the current values of the arguments but not their addresses and the corresponding parameters are assigned these values. Since the addresses of the arguments are not available to the called function, any change in the value of a parameter inside the called function does not cause a change in the corresponding argument. In *call by*

reference the address of the argument is supplied to the called function and any change in the value of a parameter is automatically reflected in the corresponding argument.

4.2.5 Storage Classes :

The storage class determines the lifetime of the storage associated with the variable. There are basically four types of storage classes which are as follows:

* static class
* auto class
* extern class
* register class

(i) static class : Global variables are classified in C as **static** variables, meaning they come into existence when the program is executed and continue to exist until the entire program terminates. Variables declared outside all blocks at the same level as function definitions are always static. Within a block, a variable can be declared to be static by prefixing its type declaration with the storage class specifier **static** in the follwoing manner :

static type variable_name;

For example, the declaration of *i* in

```
int fn(void)
{
static int i;
----------
---------
}
```

specifies that i is a static variable of type integer.

For example :

```
void stat();/* prototype fn*/
void main()
{ int i;
for (i=0;i<5;++i)
stat();
}
void stat()
{int auto_var = 0;
```

```
    static int static_var = 0;
    printf("auto =%d, static = %d\n", auto_var, static_var);
    ++auto_var;
    ++static_var;
    }
```

Output is :

```
    auto_var = 0, static_var = 0
    auto _ var = 0, static _ var = 1
    auto_var = 0, static_var = 2
    auto_var = 0, static_var = 3
    auto_var = 0, static_var = 4
```

Clearly the auto_var variable is created each time. The static_var is created once and remembers its value.

Note : If a local variable is declared as **static,** it remains in existence throughout the program execution. Since a global variable automatically possesses this quality, it is not necessary to declare global variables as **static.**

(ii) auto class : Local variables usually are created each time a function is executed and disappear when the function terminates. Local variables are by default classified as **auto** variables, because they are allocated automatically when a function is executed and deallocated automatically when the function terminates. A varable is specified to be automatic by prefixing its type declaration with the storage class specifier **auto** in the following manner :

```
    auto type variable_name;
```

For example, the declarations of the variables *i* and *result* in

```
int fact(int n)
{
int i, result;
---------
}
```

is equivalent to

```
int fact(int n)
{
auto int i, result;
```

```
----------
}
```

(iii) extern class : The extern storage class does not creat a variable but merely informs the compiler of its existence. If a variable is defined outside any function then it can be accessed by any statement following this definition in the rest of the source file by simply naming the variable. However if this variable is needed in the same file but at a point earlier than that at which it has been defined, it must be declared with the key word **extern** before it can be used. For example :

```
void main(void)
{
        extern int i;
        printf("i=%d",i);
}
int i=4;
```

This program prints *i=4*, since the *i* referred to in main is the global variable *i*; no local *i* is declared.

Note: An **extern** declaration may never include an initialization, since the variable is not created when it is declared as **extern**.

(iv) register class : A variable declaration with the **register** storage class is a suggestion to the compiler that the variable be assigned to a register rather that to a location in memory. The reason we say "suggestion" is that the number of **register** variables in existence at any time is limited by the number of registers available. The C compiler tries to accommodate as many **register** variables as possible. It must be declared with the key word **register** before the data type of the variable.

4.2.6 Recursion :

Recursion is the process whereby a construct operates on itself. A recursive function is one that calls itself, but which halts at some definite point to avoid infinite recursion. In C, functions may be defined recursively. For example, a factorial function defined as

Factorial(0)=

Factorial(n)=n*Factorial(n-1),n>0

is the classic example of recursion. This function can be coded as

```
int Factorial(int n(
{
```

```
if(n==0)
return I;     /* Termination condition*/
else
        return n*Factorial(n-1);          /* recursive condition*/
}
```

4.2.7 Some standard mathematical library function :

The following are some library functions :

i) acos(): This returns the arc cosine of arg. The argument to acos() must be in the range –1 to 1.

ii) asin(): This returns the arc since of arg. The argument to asin must be in the range –1 to 1.

iii) atan(): It returns the arc tangent of argument.

iv) atan2(): This function returns the arc tangent of y/x. It uses the signs of its arguments to compute the quadrant of the return value.

v) ceil(x): This returns the smallest integer not less than x.

vi) floor(x): It returns the largest integer not greater than x.

vii) fmod(x/y): It returns the remander of x/y.

viii) cos(): It returns the cosine of argument. The value of arg must be in radians.

ix) sin(): This function returns the sine of arg. The value of argument must be in radians.

x) exp(): It returns the natural logarithm base e raised to the arg power.

xi) log(num) : It returns the natural logarithms of num. A domain error occurs if num is negative and a range error occurs if the argument is zero.

xii) sqrt(num) : It returns the square root of num. For negative argument, a domain error will occur.

4.2.8 Programming Examples :

Problem-1 : Write a program to gerate fibonacci series using iteration.

```
#include<stdio.h>

void main()
{
void Fib(unsigned int n);
```

```
unsigned int n;
printf("Program to generate fibonicii numbers\n");
printf("Enter the total number to be generated:");
printf("Enter the total number to be generated:");
scanf("%d",&n);
Fib(n);
}
void Fib(unsigned into TotNum)
{
      unsigned int fLast=1,sLast=0;
      unsigned into CurNum;
      printf("0\n");
      do
      {
      CurNum=fLast+sLast;
      printf("%u\n",CurNum);
      sLast=fLast;
      fLast=CurNum;
      TotNum--;
      }while(TotNum>1);
return 0;
}
```

Prblem-2 : Write a program to convert the upper case to lower case and vice-versa.

```
#include<stdio.h>
#include<ctype.h>
void uper(char instr[])
{
      int i=0;
      while(instr[i])
```

```
{
        instr[i]=toupper(instr[i]);
i++;
}
}
void lower(char instr[])
{
int i=0;
while(instr[i])
{
instr[i]=tolower(instr[i]);
i++;
}
}
void main()
{
        char instr[80],inchar;
        printf("Input the string:");
        gets(instr);
        printf("To upper or lower case(u/l)?");
        scanf("%c",&inchar);
        switch(inchar)
        {
case 'u':
case 'U' : upper (instr);
         break;
case 'l' :
case 'L' : lower(instr);
         break;
default : printf("Invalid input!\n");
printf("%s\n",instr);
}
```

4.3 Structures :

A structure is a collection of logically related data items grouped together under a single name, called a structure tag. The data items that make up a structure are called its members, components, or fields, and can be of different types. The general format for defining a structure is

```
struct tag
{
        variable declarations
};
```

where **struct** is a key word that introduces a structure definition, tag is the name of the structure, and variable declarations is the set of type declarations for the members data items that make up the structure.

4.3.1 Structure Variables :

A structure definition defines a new type and variables of this type can be declared by including a list of variable names between the right brace and the terminating semicolon in the structure definition.

Example :

```
struct gun
{
char name[50];
int magazinesize;
float calibre;
};
struct gun armies;
```

definies a new structure gun and makes armies an instance of it.

NOTE : that gun is a ***tag*** for the structure that serves as shorthand for future declarations. We now only need to say struct gun and the body of the structure is implied as we do to make the armies variable. The tag is ***optional.***

Variables can also be declared between the} and; of a struct declaration, ***i.e.:***

```
struct gun
{
char name[50];
int magazinesize;
float calibre;
```

} **arnies;**

struct's can be pre-initialised at declaration :

struct gun arnies = {"Uzi",30,7};

which gives armies a 7mm. Uzi with 30 rounds of ammunition.

4.3.2 Accessing Structure Members :

To access a member (or field) of a struct, C provides the dot operator (.) by a construction of the form

structure variable.member name

For example, to give armies more rounds of ammunition:

arnies.magazineSize=100;

4.3.3 Structure initialization :

A variable of a particular structure type can be initialized by following its definition with an initializer for the corresponding structure type. For example, the declaration

structu date

{

int day,month,year;

} independence= {15,8,1947};

initializes the member variables *day, month and year* of the structure variable independence to 15, 8 and 1947 respectively and the declaration

struct date republic = {26,1,1950};

initializes the member varibales *day*, *month*, and *year* of the structure variable republic to 26, 1 and 1950 respectively.

4.3.4 Defining New Data Types :

typedef can also be used with structures. The following creates a new type a gun which is of type struct gun and can be initialized as usual :

typedef struct gun

{

char name[50];

```
int magazinesize;
float calibre;
} agun;
agun arimies={"Uzi",30,7};
```

Here gun still acts as a ***tag*** to the struct and is optional. Indeed since we have defined a new data type it is not really of much use, agun is the new data type. armies is a variable of type agun which is a structure.

C also allows arrays of structures:

```
typedef struct gun
{
char name[50];
int magazinesize;
float calibre;
} agun;
agun armiesguns[1000];
```

This gives armiesguns a 1000 guns. This may be used in the following way:

```
armiesguns[50].calibre=100;
```

gives Arnie's gun number 50 a calibre of 100mm, and:

```
itscalibre=armiesguns[0].calibre;
```

assigns the calibre of Arnie's first gun to itscalibre.

4.3.5 Structure and Array :

Array of structures are commonly used when a large number of similar records are required to be processed together. This is shown by the following example :

```
struct date birthdays[4]={{14,3,1879},{24,7,1969},(1,12,2002}};
```

4.3.6 Unions :

The union is a construct that allows different types of data items to share the same block of memory. The compiler automatically allocates sufficient space to hold the largest data item in the union. The syntax for defining and accessing a union is similar to that for structures except that the keyword union is used in place of **struct.**

For example :

```
union number
{
short shortnumber;
long longnumber;
double floatnumber;
} anumber;
```

defines a union called number and an instance of it called anumber, number is a union ***tag*** and acts in the same way as a tag for a structure.

Members can be accessed in the following way :

```
printf("%1d\n",anumber.longnumber);
```

This clearly displays the value of longnumber.

When the C compiler is allocating memory for unions it will always reserve enough room for the largest member (in the above example this is 8 bytes for the double). In order that the program can keep track of the type of union variable being used at a given time it is common to have a structure (with union embedded in it) and a variable

which flags the union type:

An example is :

```
typedef struct
{ int maxpassengers;
} jet;
typedef struct
{ int liftcapacity;
} helicopter;
typedef struct
{ int maxpayload;
} cargoplane;
typedef union
```

```
{ jet jetu;
helicopter helicopteru;
cargoplane cargoplaneu;
} aircraft;
typedef struct
{ aircrafttype kind;
int speed;
aircraft description;
} an_aircraft;
```

This example defines a base union aircraft which may either be jet, helicopter, or cargoplane. In the aircraft structure there is a kind member which indicates which structure is being held at the time.

4.3.7 Programming Example :

Problem : Write a program to find the total score of all students for five papers in M.Sc. Applied Mathematics in Part-I Examination.

```
#include<stdio.h>
void main(void)
{
struct name
{
int roll;
int Paper[5];
};
struct name student[10];
int i,j,totla;
printf("Enter all data:\n");
for(i=0;i<10;i++)
{
printf("\nRoll:");
scanf("%d",&student[i].roll);
for(j=0;j<5;j++)
{
printf("\nScore of Paper-%d",j);
scanf("%d",&student[i].paper[j]);
}
```

```
}
for(i=0;i<10;i++)
{
student[i].total=0;
for(j=0;j<5;j++)
student[i].total=student[i].total+student[i].Paper[j];
}
printf("Roll No.        Total\n");
for(i=0;i<10;i++)
printf("%d   %d\n",student[i].roll,student[i].total);
}
```

4.4 Pointers :

Pointer is a fundamental part of C. If you cannot use pointers properly then you have basically lost all the power and flexibility that C allows. The secret to C is in its use of pointers.

C uses ***pointers*** a lot. **Why":**

* It is the only way to express some computation.
* It produces compact and efficient code.
* It provides a very powerful tool.

C uses pointers explicitly with :

* Arrays,
* Structures,
* Functions.

NOTE. Pointers are perhaps the most difficult part of C to understand.

4.4.1 What is a Pointer ?:

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type.

The ***unary*** or ***monadic*** operator **&** gives the "address of a variable".

The ***indirection*** or dereference operator * gives the "contents of an object ***pointed to*** by a pointer".

To declare a pointer to a variable do :

int *pointer;

NOTE: We must associate a pointer to a particular type : You can't assign the address of a **short int** to a **long int**, for instance.

Consider the effect of the following code :

```
int x = 1, y = 2;
int *ip;
ip = &x;
y = *ip;
x = ip;
*ip = 3;
```

It is worth considering what is going on at the ***machine level*** in memory to fully understand how pointer works. Consider the following figure. Assume for the sake of this discussion that variable x resides at memory location 100, y at 200 and ip at 1000. **Note** A pointer is a variable and thus its values need to be stored somewhere. It is the nature of the pointers value that is ***new***.

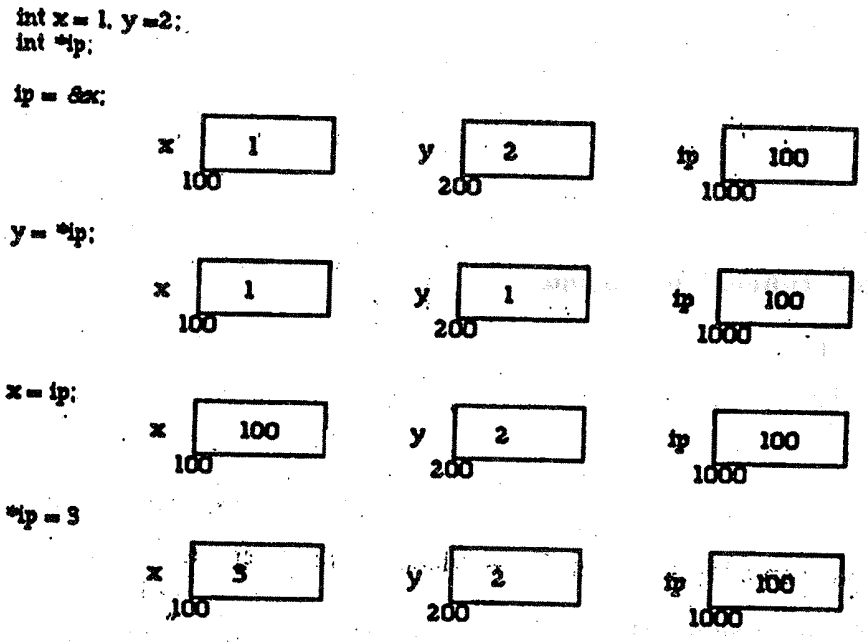


Fig. Pointer, Variables and Memory

Now the assignments x=1 and y=2 obviously load these values into the variables. ip is declared to be a ***pointed to an integer*** and is assigned to the address of x(&x). So ip gets loaded with the value 100. Next y gets assigned to the ***contents of*** ip. In this example ip currently ***points*** to memory location 100 -- the location of x. So y gets assigned to the values of x -- which is 1.

We have already seen that C is not too fussy about assigning values of different type. Thus it is perfectly **legal** (although not all that common) to assign the current value of ip to x. The value of ip at this instant is 100. Finally we can assign a value to the contents of pointer (*ip).

IMPORTANT : When a pointer is declared it does not point anywhere. You must set it to point anywhere. You must set it to point somewhere before you use it.

So

```
int*ip;
*ip=100;
```

will generate an error (program crash!!).

The correct use is :

```
int *ip;
int x;
ip = &x;
*ip = 100;
```

we can do **integer arithmetic on a pointer :**

```
float *flp, *flq;
flp = *flp + 10;
++*flp;
(*flp)++;
flq = flp;
```

NOTE : A pointer to any variable type is an address in memory -- which is an integer address. A pointer is definitely NOT an integer.

The reason we associate a pointer to a data type is so that it knows how many bytes the data is stored in. When we increment a pointer we increase the pointer by one "block" memory.

So for a character pointer ++ch_ptr adds 1 byte to the address.

For an integer or float ++ip or ++flp adds 4 bytes to the address.

Consider a float variable (fl) and a pointer to a float (flp) as shown in following Fig.

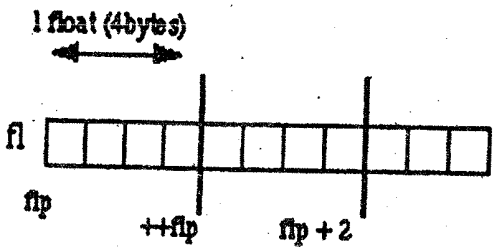


Fig. Pointer Arithmetic

Assume that flp points to fl then if we increment the pointer (++flp) it moves to the position shown 4 bytes on. If on the other hand we added 2 to the pointer then it moves 2 **float positions** ***i.e.*** 8 bytes as shown in the Figure.

4.4.2 Pointer and Functions :

Let us now examine the close relationship between pointers and C's other major parts. We will start with functions.

When C passes arguments to functions it passes them by value. There are many cases when we may want to alter a passed argument in the function and receive the new value back once to function has finished. Other languages do this (***e.g.*** var parameters in PASCAL). C uses pointers explicitly to do this. Other languages mask the fact that pointers also underpin the implementation of this. The best way to study this is to look at an example where we must be able to reeive changed parameters. Let us try and write a function to swap variables around?

The usual function ***call*** **:**

swap(a, b);// WONT WORK.

Pointers provide the solution : ***Pass the address of the variables to the functions and access address of function.***

Thus our function call in our program would look like this :

swap(&a, &b);

The Code to swap is fairly straightforward.

```
void swap(int *px, int *py)
{ int tem;
temp = *px;
/* contents of pointer */
*px = *py;
*py = temp;
}
```

We can return pointer from functions. A common example is when passing back structures ***e.g.*** :

```
typedef struct {float x,y,x;} COORD;
main()
{ COORD p1, *coord_fn();
/* declare fn to return ptr of COORD type*/
        .....
p1 = *coord_fn(...);
/* assign contents of address returned */
    ....
}

COORD *coord_fn (...)
{ COORD p;
....
p = ...;
/* assign structure values */
return &p;
/* return address of p */
}
```

Here we return a pointer whose contents are immediately ***unwrapped*** into a variable. We must do this straight away as the variable we pointed to was local to a function that has now finished. This means that the addres space is free and can be overwritten. It will not have been overwritten straight after the function has quit though so this is perfectly safe.

4.4.3 Pointers and Arrays :

Pointers and arrays are very closely linked in C.

Hint : think of array elements arranged in consecutive memory locations.

Consider the following :

```
int a[10], x;
int *pa;
pa = &a[0]; /* pa pointer to address of a[0] */
x = *pa;
/* x = contents of pa (a[0] in this case) */
```

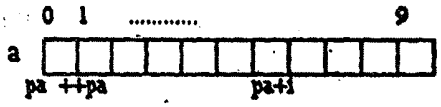


Fig. Arrays and Pointers

To get somewhere in the array (Fig.) using a pointer we could do :

pa + i ≡ a [i]

WARNING : There is no bound checking of arrays and pointers so you can easily go beyond array memory and overwrite other things.

C however is much more subtle in its link between arrays and pointers.

For example we can just type

```
pa = a;
```

instead of

```
pa = &a[0];
```

and

a[i] can be written as *(a+i).

i.e. &a[i] ≡ a + i.

We also express pointer addressing like this :

pa[i] ≡*(pa + i).

However pointers and arrays are different :

* A pointer is a variable. We can do
  pa = a and pa++.
* An Array is not a variable. a = pa and a++ ARE ILLEGAL.

This stuff is very important. Make sure you understand it. We will see a lot more of this.

We can now understand how arrays are passed to functions.

When an array is passed to a function what is actually passed is its initial elements location in memory.

So :

strlen(s) ≡ strlen(&s[0])

This is why we declare the function :

int strlen(char s[]);

An equivalent declaration is : **int strlen(char *s);**

since char s[]≡char *s.

strlen(s) is a ***standard library*** function that returns the length of a string. Let's look at how we may write a function:

```
int strlen(char *s)
{ char *p = s;
while (*p != '\0);
p++;
return p-s;
}
```

Now lets write a function to copy a string to another string. strcpy() is a standard library function that does this.

```
void strcpy(char *s, char *t)
{ while ( (*s++ = *t++) ! '\0);}
```

This uses pointers and assignment by value.

4.4.4 Arrays of Pointers :

We can have arrays of pointers since pointers are variables. We illustrate this by an example such as *Sort lines of text of different length.* **Arrays of Pointers** are a data representation that will cope efficiently and conveniently with variable length text lines.

How can we do this?

* Store lines end-to-end in one big char array (Fig.). \n will delimit lines.
* Store pointers in a different array where each pointer points to 1st char of each new line.
* Compare two lines using strcmp() standard library function.
* If 2 lines are out of order -- swap pointer in pointer array (not text).

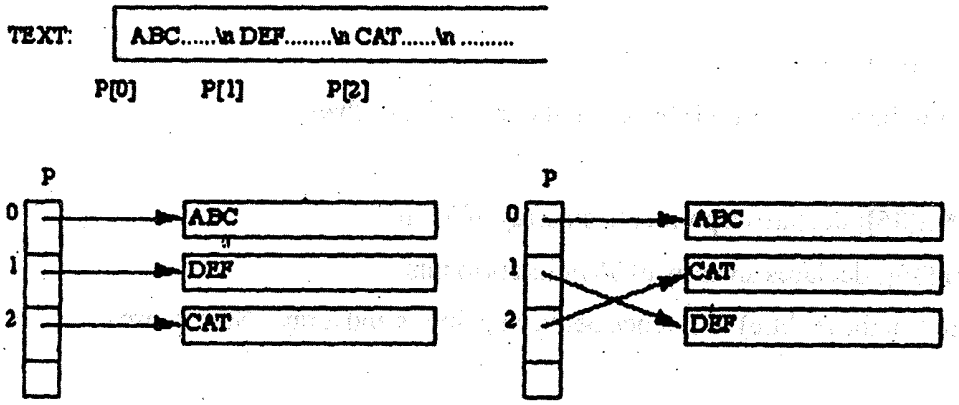


Fig. Arrays of Pointers (String Sorting Example)

This eliminates :

* complicated storage management.
* high overheads of moving lines.

4.4.5 Multidimensinoal arrays and pointers :

We should think of multidimensional arrays in a different way in C :

A 2D array is really a 1D array, each of whose elements is itself an array

Hence

a[n][m] notation.

Array elements are stored row by row.

When we pass a 2D array to a function we must specify the number of columns -- the number of rows is irrelevant.

The reason for this is pointers again. C needs to know how many columns in order that it can jump from row to roaw in memory.

Consider int a[5][35] to be passed in a function :

We can do :

f(int a[][35]) {.....}

or even :

f(int (*a)[35]) {.....}

We need parenthesis (*a) since [] have a higher precedence than *

So :

int (*a)[35]; declares a pointer to an array of 35 ints.

int *a[35]; declares an array of 35 pointers to ints.

Now lets look at the (subtle) difference between pointers and arrays. Strings are a common application of this.

Consider :

char *name[10];

char Aname[10][20];

We can legally do name[3][4] and Aname [3][4] in C.

However

* Aname is a true 200 element 2D char array.
* access elements via

  20*row + col + base_address

  in memory.

* name has 10 pointer elements.

NOTE : If each pointer in name is set to point to a 20 element array then and only then will 200 chars be set aside (+ 10 elements).

The advantage of the latter is that each pointer can point to arrays be of different length.

Consider :

```
char *name[] = {"no month", "jan", "feb", ...};
char Aname [][15] = {"no month", "jan", "feb", ...};
```

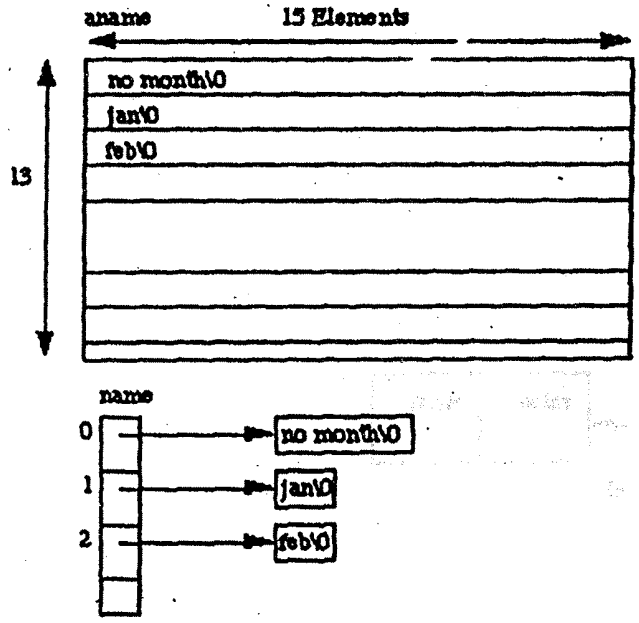


Fig. 2D Arrays and Arrays of Pointers

4.4.6 Static Initialisation of Pointer Arrays :

Initialization of arrays of pointers is an ideal application for an internal static array.

```
some_fn()
{static char *months = {"no month", "jan", "feb",          ....};
}
```

static reserves a private permanent bit of memory.

4.4.7 Pointers and Structures :

These are fairly straight forward and are easily defined. Consider the following :

```
struct COORD {float x,y,z;) pt;
struct COORD *pt_ptr;
pt_ptr = &pt;/* assigns pointer to pt */
```

the -> operator lets us access a member of the structure pointed to by a pointer ***i.e.*:**

```
pt_ptr->x = 1.0;
pt_ptr->y = pt_ptr->y -3.0;
```

Example : Linked Lists

```
typedef struct
{ int value;
ELEMENT *next;
} ELEMENT;
ELEMENT n1, n2;
n1.next = &n2;
```

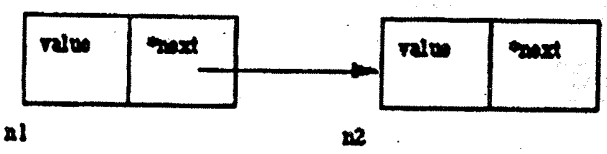


Fig. Linking Two Nodes

NOTE We can only declare next as a pointer to ELEMENT. We cannot have element of the variable type as this would set up a ***recursive*** definition which is **NOT ALLOWED.** We are allowed to set a pointer reference since 4 bytes are set aside for any pointer.

4.4.8 Programming Examples

Problem - 1 : Write a program to compare two strings.

```
#include<stdio.h>
#include<string.h>
int usrstrcmp(char *, char *);
```

```
void main()
{
        int cmp;
        static char s1[]="Good";
        static char s2[]="Morning";
        printf("Comparison of strings library function is");
        printf("%s\n",strcmp(s1,s2));
        cmp=usrstrcmp(s2,s1);
        printf("Comparison of strings using user defined function:");
        printf("%s\n",cmp);
}

int usrstrcmp(char *p1,char *q1)
{
        int k=0;
        char *p,*q;
        for(p=p1,q=q1;(*p==*q)&&((*p!='\0')&&(*q!='\0')));p++,q++)
        if((*p=='\0')&&(*q=='\0'))
        k=0;
        else
        k=1;
return k;
}
```

Note : **strcmp()** function compares two strings, character by character and returns a value 0 if both the strings are equal and a value 1 if the strings are different. strcmp() takes two arguments of the type pointer to character strings.

Problem -2 : Write a program to concatenate two strings in a single string.

```
#include<stdio.h>
#include<string.h>
void usrstrcat(char *, char *);
void main()
{
        static char s1[]="Good";
        static char s2[]="Morning";
        strcat(s1,s2);
        printf("Concatenation using standard library function is ");
        printf("%s\n",s1);
        usrstrcat(s2,s1);
```

```
        printf("Concatenation using user defined function is %s\n",s1);
}
void usrstrcat(char *p, char *q)
{
while (*p!='\0')
p++;
while(*p!='\0')
{
        *p=*q;
        p++;
        q++;
        }
        *p\'\0';
        }
```

Note : **strcat()** function concatenates two strings resulting in a single string. It takes two arguments, namely the pointers to hte two strings. The resulting string is stored in the first string specified in the argument list.

Problem - 3 : Write a program to find the length of a string.

```
#include<stdio.h>
#include<string.h>
int usrstrlen(char *);
void main()
{
        int len;
        static char s1[]="Good";
        printf("Length of string using standard library function \n");
        printf("%d",strlen(s1));
        len=usrstrlen(s1);
        printf("Lenght of string using user defined function %d\n",len);
        }

int usrstrlen(char *ptr)
{
int ln= 0;
while (*ptr!='\0');
ln+=1;
```

```
ptr++;
    }
return ln;
}
```

Note : strlen() function returns the length of a given string. It takes a single argument, namely, the pointer to the base address of the string.

5.0 Unit Summary

In tis module, implementations of arrays, and strings, the concept of functions, the storage classes, the concept of recursion, the structures and union, the concept of pointers and its uses are discussed.

6.0 Self Assessment Questions

Exercise 1 : What is the difference between a function declaration and a function definition?

Exercise 2 : Clearly differentiate between function prototype, function definition and function call.

Exercise 3 : Write a program to find the value $\int_a^b f(x)dx$ by Trapezoidal formula.

Exercise 4 : Write program using enumerated types which when given today's date will print out tommorow's date.

Exercise 5 : Write a simple database program that will store persons details such as age, date of birth, address etc.

Exercise 6 : Write a C program to read through an array of any type using pointers. Write a C program to scan through this array to find a particular value.

Exercise 7 : Write a program that takes three variable (a, b, b) in as separate parameters and rotates the values stored so that value a goes to be, b, to c and c to a.

7.0 Suggested further Readings

1. Balgurusamy, E., Programming in ANSI C, Tata McGraw Hill, 1992
2. Venugopal, K.R., Programming with C, Tata McGraw Hill, 2001
3. Kumar, R. and Agarwal, R., Programming in ANSI C, Tata McGraw Hill, 1993

--- 0 ---

"Learner's Feed-back"

After going through the Modules / Units please answer the following questionnaire.
Cut the portion and send the same to the Directorate.

To
The Director
Directorate of Distance Education,
Vidyasagar University
Midnapore - 721 102

1. The modules are : (give ✓ in appropriate box)
   ☐ easily understandable; ☐ very hard; ☐ partially understandable.

2. Write the number of the Modules/Units which are very difficult to understand :
   .....
   .....
   .....

3. Write the number of Modules / Units which according to you should be re-written :
   .....
   .....
   .....

4. Which portion / page is not understandable to you? (mention the page no. and portion)
   .....
   .....
   .....
   .....

5. Write a short comment about the study material as a learner.
   .....
   .....
   .....
   .....
   .....

Date : .....

.....
(Full Signature of the Learner)
Enrolment No. .....
Phone / Mobile No. .....


BADAL DINESH BHAVAN
DIRECTORATE OF DISTANCE EDUCATION